wolfTPM Manual

2026-11-13

Contents

1 wolfTPM **11**
1.1 Project Features . . . 11
1.2 Standards and Features . . . 12
1.3 Documentation map . . . 12
1.4 See Also . . . 13

2 TPM 2.0 Overview **14**
2.1 Protocol overview . . . 14
2.2 Hierarchies . . . 14
2.3 Platform Configuration Registers (PCRs) . . . 14
2.4 Terminology . . . 14
2.5 Device Identification . . . 15
2.6 See Also . . . 16

3 Project Structure **17**

3.1 Source tree . . . 17
3.2 Directory purposes . . . 17
3.3 Library layout . . . 18
3.4 See Also . . . 18

4 Getting Started 19
4.1 Prerequisites and building wolfSSL . . . 19
4.2 Build wolfTPM . . . 19
4.3 Run your first example . . . 19
4.4 See Also . . . 20

5 Building wolfTPM 21
5.1 Building wolfSSL . . . 21
5.2 Using an alternate wolfSSL directory . . . 21
5.3 Building with autotools . . . 21
5.4 Building using CMake . . . 22
5.5 Bare-metal build . . . 22
5.5.1 Prerequisites . . . 22
5.5.2 Step 1: Define preprocessor macros . . . 22
5.5.3 Step 2: Create a user_settings.h file . . . 22
5.5.4 Step 3: Configure include paths . . . 24
5.5.5 Step 4: Add source files . . . 24
5.5.6 Step 5: Implement the SPI HAL callback . . . 25
5.5.7 Step 6: Initialize and use wolfTPM . . . 26
5.5.8 Optional build configurations . . . 27
5.5.9 Cryptographic key storage . . . 27
5.5.10 Troubleshooting . . . 28
5.6 See Also . . . 28

6 Build Options 29
6.1 Reading the tables . . . 29
6.2 General and debug . . . 29
6.3 I/O layer and bus selection . . . 30
6.4 TPM vendors and modules . . . 32
6.5 Operating system transports and simulators . . . 32
6.5.1 Default simulator behavior . . . 33
6.5.2 fwTPM macros . . . 33
6.5.3 fwTPM-only and NV modes . . . 34
6.6 SPDM . . . 34
6.7 Post-quantum (v1.85) . . . 35
6.8 Preprocessor defines . . . 35
6.9 See Also . . . 37

7 System Interfaces 38
7.1 Software simulator (SWTPM) . . . 38
7.1.1 wolfTPM SWTPM support . . . 38
7.1.2 wolfTPM SWTPM UART support . . . 38
7.1.3 Using a SWTPM . . . 39
7.1.4 Running examples . . . 42
7.2 Linux kernel device (/dev/tpmX) . . . 42
7.2.1 Two device nodes . . . 42
7.2.2 Startup, shutdown, and shared state . . . 43
7.2.3 What the native API does on autodetect builds . . . 43
7.2.4 Transient handles do not outlive a process . . . 43
7.2.5 Building . . . 44

7.2.6 Permissions . . . 44
7.2.7 NVIDIA Jetson Orin (Tegra234) firmware TPM . . . 45
7.2.8 Testing . . . 46
7.2.9 CI coverage . . . 46
7.3 Windows TBS API . . . 46
7.3.1 Limitations . . . 46
7.3.2 Building in MSYS2 . . . 46
7.3.3 Building on Linux . . . 47
7.3.4 Running on Windows . . . 47
7.4 See Also . . . 47

8 Supported Hardware 49
8.1 Platform . . . 49
8.2 Tested Hardware . . . 49
8.3 Supported Parts . . . 50
8.4 Autodetect and the Kernel Device . . . 52
8.5 Building per Vendor . . . 52
8.5.1 Infineon . . . 52
8.5.2 STMicroelectronics ST33 . . . 53
8.5.3 Microchip ATTPM20 . . . 53
8.5.4 Nuvoton . . . 53
8.5.5 Nations Technologies . . . 53
8.5.6 SEALSQ QVault . . . 53
8.5.7 Espressif ESP-IDF . . . 54
8.6 See Also . . . 54

9 HAL IO Callback 55
9.1 Example HAL Implementations . . . 55
9.2 wolfHAL . . . 55
9.2.1 Board definitions . . . 55
9.3 HAL IO Callback Function . . . 56
9.4 Additional Build Options . . . 57
9.5 TPM Reset (nRST) HAL Macros . . . 57
9.6 Additional Compiler Macros . . . 57
9.7 See Also . . . 57

10 Examples Overview 58
10.1 Native API test . . . 58
10.2 Wrapper API test . . . 58
10.3 Crypto primitive examples . . . 58
10.4 Parameter encryption . . . 59
10.4.1 Post-quantum session keys (v1.85) . . . 59
10.5 Run flags and test handles . . . 59
10.6 See Also . . . 60

11 Key Management 61
11.1 Key generation overview . . . 61
11.2 Programs (examples/keygen/) . . . 63
11.3 Wrapper utilities (examples/wrap/) . . . 63
11.4 Storing keys in NV . . . 63
11.5 See Also . . . 63

12 Attestation 64
12.1 Remote attestation overview . . . 64
12.2 Creating keys for attestation . . . 64

COPYRIGHT ©2026 wolfSSL Inc.

12.3 MakeCredential and ActivateCredential . . . 65
12.3.1 Make credential . . . 65
12.3.2 Activate credential . . . 65
12.3.3 Certify . . . 66
12.4 Quote and PCR attestation . . . 66
12.4.1 PCR basics . . . 67
12.4.2 Bank allocation . . . 67
12.4.3 Quote . . . 67
12.4.4 Tool usage . . . 68
12.4.5 Typical demo output . . . 69
12.4.6 Measuring a system file (local attestation) . . . 70
12.4.7 Quote with encrypted qualifying data . . . 71
12.5 Signed timestamp (GetTime) . . . 71
12.6 Endorsement key certificates . . . 71
12.6.1 Example detail . . . 71
12.6.2 Example certificate chains . . . 71
12.7 Device identity . . . 72
12.8 See Also . . . 73

13 Sealing and NVRAM 74
13.1 Seal and unseal overview . . . 74
13.1.1 Sealing data into a TPM 2.0 key . . . 74
13.1.2 Sealing to PCRs with a signed policy . . . 75
13.2 Seal examples . . . 75
13.2.1 seal and unseal (password policy) . . . 75
13.2.2 seal_pcr (PCR-only policy) . . . 75
13.2.3 seal_policy_auth (PolicyAuthorize and PCR) . . . 75
13.2.4 seal_nv (NV storage and PCR policy) . . . 76
13.2.5 Testing . . . 76
13.2.6 Policy comparison . . . 76
13.3 Storing keys in NVRAM . . . 77
13.3.1 NVRAM programs . . . 78
13.4 Secure boot root of trust . . . 78
13.4.1 Secure boot ROT . . . 78
13.4.2 Secure boot encryption key storage . . . 79
13.5 See Also . . . 80

14 TLS and Certificates 81
14.1 CSR . . . 81
14.2 Certificate signing . . . 81
14.3 PKCS #7 . . . 82
14.4 TLS examples . . . 82
14.4.1 Generate the certificates . . . 83
14.4.2 TLS client . . . 83
14.4.3 TLS server . . . 84
14.5 See Also . . . 85

15 Firmware Update 86
15.1 Infineon firmware (extract and update) . . . 86
15.1.1 Extracting the firmware . . . 86
15.1.2 Updating the firmware . . . 86
15.2 ST33 firmware update . . . 88
15.2.1 Firmware format auto-detection . . . 88
15.2.2 Identifying the part . . . 88
15.2.3 Field upgrade command codes . . . 89

COPYRIGHT ©2026 wolfSSL Inc.

15.2.4 Updating the firmware . . . 89
15.2.5 LMS firmware . . . 90
15.2.6 Cancel an update . . . 91
15.3 Policy-based authorization (advanced) . . . 91
15.3.1 Destructive: provisioning replaces any existing platform policy . . . 93
15.3.2 Rollback of the example-provisioned policy . . . 93
15.4 See Also . . . 93

16 Management and GPIO 94
16.1 Management utilities . . . 94
16.2 GPIO control . . . 94
16.2.1 GPIO config (ST33) . . . 94
16.2.2 GPIO config (NPCT75xx) . . . 95
16.2.3 GPIO usage . . . 95
16.3 See Also . . . 96

17 Post-Quantum Cryptography 97
17.1 Overview . . . 97
17.2 Building . . . 97
17.2.1 wolfSSL . . . 97
17.2.2 fwTPM (software TPM) . . . 97
17.2.3 Hardware TPM: SEALSQ QVault . . . 98
17.2.4 Trimming the PQC footprint . . . 98
17.2.5 CMake . . . 99
17.3 Running the examples . . . 99
17.4 Examples . . . 99
17.4.1 pqc_ctrl . . . 99
17.4.2 pqc_mssim_e2e . . . 101
17.4.3 mlkem_encap . . . 101
17.4.4 mldsa_sign . . . 101
17.4.5 PQC keys via keygen and keyload . . . 101
17.4.6 PQC keys for parameter encryption . . . 102
17.4.7 create_primary with an ML-DSA primary . . . 102
17.4.8 Post-Quantum TLS 1.3 (ML-KEM and TPM ML-DSA) . . . 102
17.5 Benchmarks . . . 103
17.6 See Also . . . 103

18 SPDM Attestation and Secure Sessions 104
18.1 Quick start . . . 104
18.1.1 Nuvoton NPCT75x . . . 104
18.1.2 Nations NS350 . . . 104
18.2 Overview and how it works . . . 105
18.2.1 Identity key mode (Nuvoton and Nations) . . . 105
18.2.2 PSK mode (Nations only) . . . 106
18.2.3 SPDM-only mode (encrypted bus enforcement) . . . 106
18.3 Building . . . 107
18.3.1 1. Clone with the wolfSPDM submodule . . . 107
18.3.2 2. wolfSSL . . . 107
18.3.3 3. wolfTPM . . . 107
18.3.4 The wolfTPM SPDM profile . . . 107
18.3.5 Configure options . . . 108
18.3.6 fwTPM SPDM responder (no silicon required) . . . 108
18.3.7 Vendor selection in dual-vendor builds . . . 109
18.4 Usage and control commands . . . 109
18.4.1 One-time setup . . . 109

18.4.2 Establishing a session . . . 109
18.4.3 Lock and unlock SPDM-only mode . . . 110
18.4.4 PSK lifecycle (Nations) . . . 110
18.4.5 Command reference . . . 111
18.4.6 Usage example . . . 112
18.4.7 nv_bind . . . 112
18.4.8 TPM reset pin control . . . 113
18.5 TCG SPDM vendor commands . . . 113
18.6 Vendor specifics . . . 114
18.6.1 Nuvoton NPCT75x . . . 114
18.6.2 Nations NS350 . . . 114
18.6.3 Auto-SPDM . . . 114
18.6.4 Memory modes . . . 115
18.7 wolfSPDM API . . . 115
18.8 Troubleshooting . . . 116
18.8.1 Handshake fails after an interrupted session . . . 116
18.8.2 SPDM error codes . . . 116
18.9 Standard SPDM support . . . 116
18.10 Automated tests . . . 116
18.11 See Also . . . 117

19 fwTPM Overview 118
19.1 Features . . . 118
19.2 Architecture . . . 118
19.3 Supported TPM 2.0 Commands . . . 119
19.3.1 Startup and Self-Test . . . 120
19.3.2 Random Number Generation . . . 120
19.3.3 Capability . . . 120
19.3.4 Key Management . . . 120
19.3.5 Cryptographic Operations . . . 121
19.3.6 Hash Sequences . . . 121
19.3.7 Sealing . . . 121
19.3.8 PCR (Platform Configuration Registers) . . . 121
19.3.9 Clock . . . 121
19.3.10 Sessions and Authorization . . . 122
19.3.11 Policy . . . 122
19.3.12 Dictionary Attack (DA) Protection . . . 122
19.3.13 Non-Volatile Storage (NV) . . . 123
19.3.14 Attestation and Credentials . . . 123
19.4 Command Coverage . . . 124
19.4.1 Implemented (103 Revision 1.38 command codes) . . . 124
19.4.2 Missing Commands . . . 125
19.4.3 Coverage Summary . . . 127
19.5 Startup and Shutdown Lifecycle . . . 128
19.6 Primary Key Derivation . . . 128
19.7 See Also . . . 128

20 Building the fwTPM 129
20.1 Prerequisites . . . 129
20.2 Build the fwTPM Server . . . 129
20.3 Build Output . . . 129
20.4 Key Build Flags . . . 129
20.5 Command-Code Enforcement . . . 130
20.6 Configuration Macros . . . 130

20.6.1 Stack and Heap Control . . . 132
20.6.2 v1.85 Embedded RAM Impact . . . 132
20.6.3 Algorithm Feature Macros . . . 132
20.6.4 TPM Feature Group Macros . . . 133
20.7 See Also . . . 134

21 Using the fwTPM 136
21.1 Starting the Server . . . 136
21.2 Connecting wolfTPM Clients . . . 136
21.2.1 Using tpm2-tools . . . 137
21.3 NV Persistence . . . 137
21.4 Transport Modes . . . 137
21.4.1 Socket / SWTPM (Default) . . . 137
21.4.2 TIS / Shared Memory . . . 138
21.5 Testing . . . 139
21.5.1 CI Tests (fwtpm-test.yml) . . . 139
21.6 API Reference . . . 141
21.6.1 Core (`fwtpm.h`) . . . 141
21.6.2 Command Processor (`fwtpm_command.h`) . . . 141
21.6.3 IO Transport (`fwtpm_io.h`) . . . 141
21.6.4 NV Storage (`fwtpm_nv.h`) . . . 142
21.6.5 TIS Server (`fwtpm_tis.h`) . . . 142
21.6.6 TIS Client (`fwtpm_tis.h`, requires `WOLFTPM_FWTPM_HAL`) . . . 142
21.7 Real-world examples . . . 142
21.8 See Also . . . 143

22 HAL and Porting 144
22.1 IO HAL (Transport) . . . 144
22.2 NV HAL (Persistent Storage) . . . 145
22.2.1 NV Storage HAL for Embedded Ports . . . 145
22.3 Clock HAL . . . 146
22.4 Porting Example . . . 146
22.5 Available Ports . . . 147
22.6 Porting Guide . . . 147
22.7 See Also . . . 148

23 fwTPM Post-Quantum Support (TPM 2.0 v1.85) 149
23.1 Algorithms . . . 149
23.2 Commands . . . 149
23.3 Primary Key Derivation . . . 149
23.4 Sign and Verify Sequences . . . 150
23.5 Buffer Constants . . . 150
23.6 Limitations and Scope . . . 150
23.7 Deferred and Out of Scope . . . 151
23.8 Test Coverage . . . 151
23.9 See Also . . . 151

24 fwTPM SPDM Responder 152
24.1 How It Works . . . 152
24.2 Building . . . 152
24.3 Starting the Responder . . . 152
24.4 Responder Identity Key . . . 152
24.5 Testing . . . 153
24.6 See Also . . . 153

COPYRIGHT ©2026 wolfSSL Inc.

25 Rust Wrapper **154**
25.1 Requirements . . . 154
25.2 Step 1: Build the C Library . . . 154
25.3 Step 2: Build the Crate . . . 154
25.4 Step 3: Run the Tests . . . 154
25.5 What the Tests Check . . . 155
25.6 Example Test Output . . . 156
25.7 Step 4: Run the Examples . . . 156
25.8 Using the Wrapper . . . 157
25.9 What the Crate Covers . . . 158
25.10 Transport Security . . . 158
25.11 License . . . 158
25.12 See Also . . . 158

26 C# Wrapper **160**
26.1 Windows . . . 160
26.2 Linux . . . 160
26.3 API Overview . . . 161
26.3.1 Device Lifetime . . . 161
26.3.2 Errors and Return Values . . . 162
26.3.3 Keys, Blobs, and Sessions . . . 162
26.3.4 Template and Csr . . . 162
26.3.5 Other Device Methods . . . 163
26.3.6 Example . . . 163
26.4 See Also . . . 164

27 STM32CubeIDE **165**
27.1 Setup . . . 165
27.2 Notes . . . 165
27.3 See Also . . . 165

28 Embedded Integrations **166**
28.1 Espressif ESP-IDF . . . 166
28.1.1 Memory . . . 166
28.1.2 Pin Assignments (I2C) . . . 166
28.1.3 Troubleshooting I2C . . . 166
28.2 Zephyr . . . 167
28.2.1 Set Up as a Zephyr Module . . . 167
28.2.2 Build and Run the Wrap Test . . . 167
28.2.3 Build and Run the Capabilities Sample . . . 167
28.3 QNX . . . 168
28.3.1 Create a QNX Application . . . 168
28.3.2 QNX SPI Master Patch for Manual Chip Select . . . 169
28.4 IAR-EWARM . . . 170
28.4.1 IAR Project Settings . . . 171
28.4.2 IAR HAL . . . 171
28.5 Visual Studio . . . 171
28.5.1 Build Steps . . . 172
28.5.2 Role of user_settings.h . . . 172
28.5.3 Windows TPM Transport . . . 172
28.6 U-Boot . . . 173
28.6.1 U-Boot Commands . . . 173
28.6.2 Enable wolfTPM in U-Boot . . . 174
28.6.3 Build and Run with QEMU . . . 174
28.7 See Also . . . 176

COPYRIGHT ©2026 wolfSSL Inc.

29 API Reference **177**
29.1 Wrapper API . . . 177
29.2 Native API . . . 177
29.3 HAL IO . . . 177
29.4 Generated Reference . . . 177
29.5 See Also . . . 177
29.6 TPM2 Proprietary . . . 177
29.6.1 Functions . . . 177
29.6.2 Detailed Description . . . 180
29.6.3 Functions Documentation . . . 180
29.7 wolfTPM2 Wrappers . . . 202
29.7.1 Classes . . . 202
29.7.2 Types . . . 203
29.7.3 Functions . . . 203
29.7.4 Detailed Description . . . 222
29.7.5 Types Documentation . . . 223
29.7.6 Functions Documentation . . . 223
29.8 wolftpm/tpm2.h . . . 376
29.8.1 Classes . . . 376
29.8.2 Types . . . 383
29.8.3 Functions . . . 403
29.8.4 Attributes . . . 410
29.8.5 Types Documentation . . . 410
29.8.6 Functions Documentation . . . 444
29.8.7 Attributes Documentation . . . 480
29.8.8 Source code . . . 483
29.9 wolftpm/tpm2_wrap.h . . . 555
29.9.1 Classes . . . 555
29.9.2 Types . . . 556
29.9.3 Functions . . . 556
29.9.4 Attributes . . . 577
29.9.5 Types Documentation . . . 577
29.9.6 Functions Documentation . . . 579
29.9.7 Attributes Documentation . . . 714
29.9.8 Source code . . . 715
29.10 hal/tpm_io.h . . . 739
29.10.1 Functions . . . 739
29.10.2 Attributes . . . 740
29.10.3 Functions Documentation . . . 740
29.10.4 Attributes Documentation . . . 745
29.10.5 Source code . . . 745

30 Testing **749**
30.1 Running tests locally . . . 749
30.2 CI workflows . . . 749
30.3 See Also . . . 750

31 Benchmarks **751**
31.1 About these numbers . . . 751
31.2 TPM 2.0 benchmarks by device . . . 751
31.3 Post-quantum (SEALSQ QVault) . . . 752
31.4 See Also . . . 753

32 SBOM and Compliance **754**
32.1 SBOM and EU CRA compliance . . . 754

32.2 Generator internals . . . 755
32.2.1 Contents . . . 755
32.2.2 The driver contract . . . 755
32.2.3 The manifest contract . . . 756
32.3 See Also . . . 756

33 Release Notes **757**
33.1 Unreleased . . . 757
33.2 wolfTPM Release 4.2.0 (Sep 14, 2026) . . . 757
33.3 wolfTPM Release 4.1.0 (Jul 10, 2026) . . . 758
33.4 Full History . . . 760
33.5 See Also . . . 760

34 Cited Sources **761**
34.1 References . . . 761
34.2 Specifications . . . 761
34.3 See Also . . . 761

COPYRIGHT ©2026 wolfSSL Inc.

1 wolfTPM

Portable TPM 2.0 project designed for embedded use. This manual covers building wolfTPM, configuring it for your hardware, using the native and wrapper APIs, and running the firmware TPM (fwTPM) and the post-quantum and SPDM features.

1.1 Project Features

- This implementation provides all TPM 2.0 API's in compliance with the specification.
- Wrappers provided to simplify Key Generation/Loading, RSA encrypt/decrypt, ECC sign/verify, ECDH, NV, Hashing/HACM, AES, Sealing/Unsealing, Attestation, PCR Extend/Quote and Secure Root of Trust.
- Any TPM 2.0 compliant module is supported. Tested modules include Infineon SLB9670, SLB9672, SLB9673, STMicroelectronics ST33KTPM2XSPI, ST33KTPM2I, ST33TPHF2XSPI, ST33TPHF2XI2C, Microchip ATTPM20, Nations Technologies/NSING Z32H330, NS350, Nuvoton NPCT650, NPCT750, and SealSQ QVault TPM (first TPM with post-quantum ML-DSA/ML-KEM in silicon).
- wolfTPM uses the TPM Interface Specification (TIS) to communicate either over SPI, or using a memory mapped I/O range.
- On Linux, wolfTPM can auto-detect between the kernel TPM driver (`/dev/tpmX`) and direct SPI access at runtime. Enable it with `--enable-autodetect` (a bare `./configure` on a common host defaults to the software TPM instead).
- wolfTPM can also use the Linux TPM kernel interface (`/dev/tpmX`) to talk with any physical TPM on SPI, I2C and even LPC bus.
- Platform support for Raspberry Pi (Linux), MMIO, STM32 with CubeMX, Atmel ASF, Xilinx, QNX, Infineon TriCore, wolfHAL and Barebox.
- The design allows for easy portability to different platforms:
  - Native C code designed for embedded use.
  - Single IO callback for hardware SPI interface.
  - No external dependencies.
  - Compact code size and minimal memory use.
- Includes example code for:
  - Most TPM2 native API's
  - All TPM2 wrapper API's
  - PKCS 7
  - Certificate Signing Request (CSR)
  - TLS Client
  - TLS Server
  - Use of the TPM's Non-volatile memory
  - Attestation (activate and make credential)
  - Benchmarking TPM algorithms and TLS
  - Key Generation (primary, RSA/ECC and symmetric), loading and storing to flash (NV memory)
  - Sealing and Unsealing data with an RSA key or externally signed policy.
  - Time signed or set
  - PCR read/reset
  - GPIO configure, read and write.
  - Endorsement Key/Cert retrieval and validation.
- Parameter encryption support using AES-CFB or XOR.
- Support for salted unbound authenticated sessions.
- Support for HMAC Sessions.
- Support for reading Endorsement certificates (EK Credential Profile).
- Includes a portable firmware TPM 2.0 implementation (fwTPM, also known as fTPM / swtpm) for embedded platforms without a discrete TPM chip. See fwTPM Overview.
- **Post-quantum cryptography support** via TPM 2.0 Library Specification v1.85: ML-DSA (FIPS 204)

COPYRIGHT ©2026 wolfSSL Inc.

signing and ML-KEM (FIPS 203) key encapsulation, enabled with `--enable-v185` (full v1.85) or the leaner `--enable-pqc` (ML-DSA / ML-KEM only), with per-operation trimming via `--enable-mldsa`/`--enable-mlkem`. Auto-detected when `--enable-fwtpm` is built against a wolfCrypt that has ML-DSA + ML-KEM. Both the client library and the fwTPM server implement the eight new v1.85 PQC commands. See Post-Quantum Cryptography.

- **SPDM attestation support** (DMTF DSP0274) over the TCG SPDM-over-TPM binding, with a TCG certificate handshake and a DSP0274 pre-shared-key (PSK) handshake, enabled with `--enable-spdm`. The fwTPM server includes an SPDM 1.3 responder so the stack can be exercised end-to-end in CI without discrete silicon. See SPDM Attestation.

1.2 Standards and Features

Area	Status	Enable flag	Page
TPM 2.0 specification	Native API for the TPM 2.0 command set, with wrappers for common operations	Always built	API Reference
TCG TPM 2.0 Library Specification revision 1.85	Post-quantum commands implemented in the client library and fwTPM server	`--enable-v185` (full v1.85)	Build Options
Post-quantum: ML-DSA (FIPS 204) and ML-KEM (FIPS 203)	Client library and fwTPM server; SealSQ QVault in silicon	`--enable-pqc` (ML-DSA / ML-KEM only), `--enable-mldsa`, `--enable-mlkem`	Post-Quantum Cryptography
SPDM attestation (TCG certificate handshake and DSP0274 PSK handshake)	Client library, plus an SPDM 1.3 responder in fwTPM	`--enable-spdm`	SPDM Attestation
Parameter encryption (AES-CFB or XOR)	Supported, along with salted unbound and HMAC sessions	Set up per session at run time	API Reference
EK Credential Profile	Endorsement certificate reading (`examples/endorsement/get_ek_certs`)	Always built	Getting Started
Device Identity (IAK / IDevID)	Tested with the ST33KTPM; default keys are ECDSA SECP384R1 with SHA2-384 in NV	`WOLFTPM_MFG_IDENTITY`	Supported Hardware
Firmware TPM (fwTPM / fTPM / swtpm)	Portable TPM 2.0 server built on wolfCrypt	`--enable-fwtpm`	fwTPM Overview

1.3 Documentation map

- Getting Started: first steps after installing.
- Building: build wolfTPM from source.
- Build Options: configure flags and the defines they set.

COPYRIGHT ©2026 wolfSSL Inc.

- Supported Hardware: tested TPM modules and platforms.
- TPM 2.0 Overview: hierarchies, PCRs and device identification.
- Project Structure: source tree layout.
- Post-Quantum Cryptography: ML-DSA and ML-KEM support.
- SPDM Attestation: SPDM handshakes and the responder.
- fwTPM Overview: the firmware TPM server.
- API Reference: native and wrapper APIs.

1.4 See Also

- Getting Started
- Build Options
- Supported Hardware
- fwTPM Overview

2 TPM 2.0 Overview

This page describes what a TPM is, the hierarchies and PCRs that wolfTPM exposes, the terminology used in the code, and how to identify the TPM module you are talking to.

wolfTPM is a portable, open-source TPM 2.0 stack with backward API compatibility designed for embedded use. It is highly portable, due to having been written in native C, having a single IO callback for SPI hardware interface, no external dependencies, and its compacted code with low resource usage. wolfTPM offers API wrappers to help with complex TPM operations like attestation and examples to help with complex cryptographic processes like the generation of Certificate Signing Request (CSR) using a TPM.

2.1 Protocol overview

Trusted Platform Module (TPM, also known as ISO/IEC 11889) is an international standard for a secure crypto processor, a dedicated micro controller designed to secure hardware through integrated cryptographic keys. Computer programs can use a TPM to authenticate hardware devices, since each TPM chip has a unique and secret RSA key burned in as it is produced.

A TPM provides the following:

- A random number generator.
- Facilities for the secure generation of cryptographic keys for limited uses.
- Remote attestation: Creates a nearly unforgeable hash key summary of the hardware and software configuration. The software in charge of hashing the configuration data determines the extent of the summary. This allows a third party to verify that the software has not been changed.
- Binding: Encrypts data using the TPM bind key, a unique RSA key descended from a storage key.
- Sealing: Similar to binding, but in addition, specifies the TPM state for the data to be decrypted (unsealed).

A TPM can also be used for platform integrity, disk encryption, password protection, and software license protection.

2.2 Hierarchies

```
Platform    TPM_RH_PLATFORM
Owner       TPM_RH_OWNER
Endorsement TPM_RH_ENDORSEMENT
```

Each hierarchy has their own manufacture generated seed.

The arguments used on `TPM2_Create` or `TPM2_CreatePrimary` create a template, which is fed into a KDF to produce the same key based hierarchy used. The key generated is the same each time, even after reboot. The generation of a new RSA 2048 bit key takes about 15 seconds. Typically these are created and then stored in NV using `TPM2_EvictControl`. Each TPM generates their own keys uniquely based on the seed.

There is also an Ephemeral hierarchy (`TPM_RH_NULL`), which can be used to create ephemeral keys.

2.3 Platform Configuration Registers (PCRs)

PCRs hold hash digests at indices 0 to 23 in banks supported and allocated by the TPM. They can be extended to prove the integrity of a boot sequence (secure boot).

2.4 Terminology

This project uses the terms append vs. marshall and parse vs. unmarshall.

Acronyms:

- HAL: Hardware Abstraction Layer.
- NV: Non-Volatile memory.
- TPM: Trusted Platform Module.

2.5 Device Identification

The following lines are the identification output captured from each tested module. The `Caps/Did/Vid/Rid` line comes from the TIS bus registers.

```
Infineon SLB9670:
TPM2: Caps 0x30000697, Did 0x001b, Vid 0x15d1, Rid 0x10
Mfg IFX (1), Vendor SLB9670, Fw 7.85 (4555), FIPS 140-2 1, CC-EAL4 1

Infineon SLB9672:
TPM2: Caps 0x30000697, Did 0x001d, Vid 0x15d1, Rid 0x36
Mfg IFX (1), Vendor SLB9672, Fw 16.10 (0x4068), FIPS 140-2 1, CC-EAL4 1

Infineon SLB9673:
TPM2: Caps 0x1ae00082, Did 0x001c, Vid 0x15d1, Rid 0x16
Mfg IFX (1), Vendor SLB9673, Fw 26.13 (0x456a), FIPS 140-2 1, CC-EAL4 1

STMicro ST33KTPM2XSPI
TPM2: Caps 0x30000415, Did 0x0003, Vid 0x104a, Rid 0x 0
Mfg STM  (2), Vendor ST33KTPM2XSPI, Fw 9.256 (0x0), FIPS 140-2 1, CC-EAL4 0

STMicro ST33TPHF2XSPI
TPM2: Caps 0x1a7e2882, Did 0x0000, Vid 0x104a, Rid 0x4e
Mfg STM  (2), Vendor , Fw 74.8 (1151341959), FIPS 140-2 1, CC-EAL4 0

STMicro ST33TPHF2XSPI (newer firmware line)
TPM2: Caps 0x30000415, Did 0x0000, Vid 0x104a, Rid 0x4e
Mfg STM  (2), Vendor , Fw 1.258 (0x0), FIPS 140-2 1, CC-EAL4 0

STMicro ST33TPHF2XI2C
TPM2: Caps 0x1a7e2882, Did 0x0000, Vid 0x104a, Rid 0x4e
Mfg STM  (2), Vendor , Fw 74.9 (1151341959), FIPS 140-2 1, CC-EAL4 0

Microchip ATTPM20
TPM2: Caps 0x30000695, Did 0x3205, Vid 0x1114, Rid 0x 1
Mfg MCHP (3), Vendor , Fw 512.20481 (0), FIPS 140-2 0, CC-EAL4 0

Nations Technologies Inc. Z32H330 TPM 2.0 module
Mfg NTZ (0), Vendor Z32H330, Fw 7.51 (419631892), FIPS 140-2 0, CC-EAL4 0

Nations Technologies Inc. NS350 TPM 2.0 module
TPM2: Caps 0x30000615, Did 0x0701, Vid 0x9999, Rid 0x 1
Mfg NSG (0), Vendor NS350, Fw 30.30 (0x24042510), FIPS 140-2 1, CC-EAL4 0

Nuvoton NPCT650 TPM2.0
Mfg NTC (0), Vendor rlsNPCT , Fw 1.3 (65536), FIPS 140-2 0, CC-EAL4 0

Nuvoton NPCT750 TPM2.0
TPM2: Caps 0x30000697, Did 0x00fc, Vid 0x1050, Rid 0x 1
```

```
Mfg NTC (0), Vendor NPCT75x"!!4rls, Fw 7.2 (131072), FIPS 140-2 1, CC-EAL4 0

SealSQ QVault TPM 2.0
TPM2: Caps 0x30000797, Did 0x0083, Vid 0x2406, Rid 0x 3
Mfg SEAL (6), Vendor QVault TPM, Fw 2.1 (0x3010303), FIPS 140-3, CC-EAL4 0

NVIDIA Jetson Orin (Tegra234) OP-TEE firmware TPM, via /dev/tpmrm0
Mfg MSFT (7), Vendor SSE fTPM, Fw 8216.1808 (0x105300), FIPS 140-2, CC-EAL4 0
```

Early ST33TPHF2X 1.x firmware reports `TPM_PT_VENDOR_STRING_1..4` as binary rather than text, so the `Vendor` field prints empty. Later 1.x firmware reports ASCII such as `ST33TPHF2XSPI`. The firmware major version identifies the line instead: 1.x and 2.x are ST33TPHF2X (SPI and I2C firmware respectively), 9.x is ST33KTPM2X and 10.x is ST33KTPM2A. See `examples/firmware/README.md` in the wolfTPM source tree for how this selects the firmware update format and command codes.

!!! note The NVIDIA Jetson Orin entry has no `Caps/Did/Vid/Rid` line because those values come from TIS bus registers, which a firmware TPM does not have. The entry was captured with `--enable-autodetect`, where `wolfTPM2_Init_ex` returns as soon as the kernel device opens, so the debug line is never reached. An `--enable-devtpm` build still prints it, reading all zeros. `Fw 8216.1808` is `TPM_PT_FIRMWARE_VERSION_1` = `0x20180710`, which this implementation uses to carry a build date (2018-07-10) rather than a version number. Spec revision is 1.62, and all four PCR banks (SHA-1, SHA-256, SHA-384, SHA-512) are allocated with PCRs 0 to 23.

2.6 See Also

- Supported Hardware
- API Reference
- Getting Started
- Project Structure

3 Project Structure

This page lists the top-level directories of the wolfTPM source tree and the purpose of each, with notes on the firmware TPM and SPDM subdirectories and the library headers.

3.1 Source tree

```
wolfTPM/
  src/            TPM 2.0 core library and wrappers
    fwtpm/        firmware TPM (fwTPM) server
    spdm/         SPDM responder and vendor adapters
  wolftpm/        public headers
    fwtpm/        fwTPM headers
    spdm/         SPDM headers
  examples/       example applications
  hal/            tpm_io_* IO callback backends
  tests/          unit and API tests
  IDE/            IDE and board projects
  docs/           this manual and the Doxyfile
  certs/          example keys and certificates
  cmake/          CMake support files
  m4/             autoconf macros
  scripts/        test and helper scripts
  tools/          documentation and SBOM tooling
  wrapper/        language wrappers
  zephyr/         Zephyr module
```

3.2 Directory purposes

Directory	Purpose
`src/`	TPM 2.0 core (`tpm2.c`, `tpm2_packet.c`, `tpm2_tis.c`, `tpm2_param_enc.c`, `tpm2_crypto.c`, `tpm2_asn.c`) and the wrapper API (`tpm2_wrap.c`).
`src/fwtpm/`	The firmware TPM server (`fwtpm_server`): command handling, crypto, NV storage and IO.
`src/spdm/`	The SPDM responder and the vendor adapters.
`wolftpm/`	Public headers, including `tpm2.h`, `tpm2_wrap.h` and `tpm2_types.h`.
`wolftpm/fwtpm/`	Public headers for the fwTPM server.
`wolftpm/spdm/`	Public headers for SPDM (`spdm.h`, `spdm_tcg.h`, `spdm_psk.h`, `spdm_responder.h` and the vendor headers).
`examples/`	Example applications for the native and wrapper APIs.
`hal/`	IO callback backends (`tpm_io_*`) for Atmel, Barebox, Espressif, Infineon, Linux, Microchip, MMIO, QNX, ST, U-Boot, wolfHAL, Xilinx, Zephyr and fwTPM.
`tests/`	Unit tests and API tests.
`IDE/`	Projects for STM32CUBE, Espressif, QNX, IAR-EWARM and VisualStudio.

COPYRIGHT ©2026 wolfSSL Inc.

Directory	Purpose
`docs/`	This manual and the Doxygen configuration (`Doxyfile`).
`certs/`	Example keys and certificates used by the examples and tests.
`cmake/`	CMake support files.
`m4/`	Autoconf macros.
`scripts/`	Test and helper scripts.
`tools/`	Documentation tooling and the SBOM generator (`tools/sbom`).
`wrapper/`	Language wrappers: `rust` and `CSharp`.
`zephyr/`	Zephyr integration.

3.3 Library layout

wolfTPM header files are located in the following locations:

Library	Header location
wolfTPM	`wolftpm/`
wolfSSL	`wolfssl/`
wolfCrypt	`wolfssl/wolfcrypt`

The general header file that should be included from wolfTPM is shown below:

```
#include <wolftpm/tpm2.h>
```

Every example application that is included with wolfTPM includes the `tpm_io.h` header file, located in `hal/`. The `tpm_io.c` file sets up the example HAL IO callback necessary for testing and running the example applications with a Linux Kernel, STM32 CubeMX HAL or Atmel/Microchip ASF. The reference is easily modified, such that custom IO callbacks or different callbacks may be added or removed as desired.

3.4 See Also

- Building
- Build Options
- fwTPM Overview
- SPDM Attestation

4 Getting Started

wolfTPM is a portable TPM 2.0 library with a native API, a wrapper API, and a set of example applications that come ready to use after a successful build. The examples demonstrate features of a TPM 2.0 module and create RSA and ECC keys in NV storage for testing, using the handles defined in `examples/tpm_test.h`. This page covers the shortest path from a fresh checkout to a first working example.

4.1 Prerequisites and building wolfSSL

wolfTPM needs wolfSSL (wolfCrypt) built with the wolfTPM options. Build and install wolfSSL first:

```
git clone https://github.com/wolfSSL/wolfssl.git
cd wolfssl
./autogen.sh
./configure --enable-wolftpm --enable-pkcallbacks --enable-keygen
 ↪  CFLAGS="-DWC_RSA_NO_PADDING"
make
sudo make install
sudo ldconfig
```

`autogen.sh` requires automake and libtool: `sudo apt-get install automake libtool`.

See Building wolfTPM for using a wolfSSL installed in a different directory.

4.2 Build wolfTPM

```
git clone https://github.com/wolfSSL/wolfTPM.git
cd wolfTPM
./autogen.sh
./configure
make
```

!!! note On any non-Windows x86_64/aarch64 host (Linux, macOS, or BSD), a bare `./configure` automatically enables the software TPM backends (swTPM and fwTPM). This lets `make check` run without any TPM hardware attached. Selecting a hardware path turns this default off: any `--enable-<vendor>`, or `--enable-spi`, `--enable-i2c`, `--enable-mmio`, `--enable-devtpm`, `--enable-autodetect`, `--enable-winapi`, or `--enable-wintbs`. See System Interfaces.

For hardware specific build steps, see Supported Hardware.

4.3 Run your first example

A TPM must be reachable before you run any example. With the default build on a non-Windows x86_64/aarch64 host (Linux, macOS, or BSD), the examples talk to a software TPM over a socket, and `make` builds that TPM (`fwtpm_server`) but does not start it. In a separate terminal, start it from the wolfTPM directory:

```
./src/fwtpm/fwtpm_server --clear
```

The `--clear` option deletes any saved NV state so you begin with a fresh TPM. Leave the server running.

!!! note A software TPM must be running before `caps` or any other example can connect. If you built for hardware instead, connect the TPM module and skip this step.

The simplest example reads the TPM capabilities and searches for persistent handles:

```
./examples/wrap/caps
TPM2 Get Capabilities
wolfSSL Entering wolfCrypt_Init
Mfg NSG (0), Vendor NS350, Fw 30.30 (0x24042510), FIPS 140-2 1, CC-EAL4 0
Found 2 persistent handles
```

The output shown is illustrative. It reports the manufacturer and firmware details of the TPM you connect to, so it will differ between modules and for the software TPM.

Other examples go further. `./examples/native/native_test` calls the native TPM2_* APIs directly (startup, self test, random numbers, hashing, PCR operations, and more). The PKCS #7 and TLS examples require generating CSRs and signing them with a test script. See `examples/README.md` in the source tree for details.

To use parameter encryption in the examples, pass `-aes` for AES-CFB mode or `-xor` for XOR mode. Only some TPM commands and responses support parameter encryption.

!!! note To run the TLS server and client on the same machine, build with `WOLFTPM_TIS_LOCK` (`--enable-tislock`) to enable concurrent access protection.

4.4 See Also

- Building wolfTPM
- Build Options
- Supported Hardware
- System Interfaces

5 Building wolfTPM

wolfTPM is built on top of wolfSSL (wolfCrypt) and can be built with autotools, CMake, or directly into a bare-metal project using a `user_settings.h` file. This page covers each build method. Per-vendor build steps are on the Supported Hardware page, and the full list of configure switches is on the Build Options page.

5.1 Building wolfSSL

wolfSSL must be built and installed first. It can be downloaded from the downloads page or cloned from GitHub:

```
git clone https://github.com/wolfSSL/wolfssl.git
cd wolfssl
./autogen.sh
./configure --enable-wolftpm --enable-pkcallbacks --enable-keygen
 ↪  CFLAGS="-DWC_RSA_NO_PADDING"
make
sudo make install
sudo ldconfig
```

`autogen.sh` requires automake and libtool: `sudo apt-get install automake libtool`.

The `--enable-wolftpm` option is equivalent to passing these options:

```
./configure --enable-certgen --enable-certreq --enable-certext \
    --enable-pkcs7 --enable-cryptocb --enable-aescfb
```

5.2 Using an alternate wolfSSL directory

To build wolfTPM against a wolfSSL installed in a non-default location, install wolfSSL to a prefix and point wolfTPM at it with `--with-wolfcrypt`:

```
# cd /your-wolfssl-repo
./autogen.sh # as necessary
./configure --prefix=~/workspace/my_wolfssl_bin --enable-all
make install

# then for some other library such as wolfTPM:

# cd /your-wolftpm-repo
./configure --enable-swtpm --with-wolfcrypt=~/workspace/my_wolfssl_bin
```

5.3 Building with autotools

Once wolfSSL is installed, download wolfTPM from the downloads page or clone it from GitHub, then build it:

```
git clone https://github.com/wolfSSL/wolfTPM.git
cd wolfTPM
./autogen.sh
./configure
make
```

Add the options you need to `./configure`. For example, `--enable-devtpm` uses the Linux kernel TPM device, and `--enable-swtpm` uses a TPM simulator. See Build Options for the list, and Supported Hardware for the options each TPM module needs.

On Linux x86_64 and aarch64, a bare `./configure` auto-enables the software TPM backends so that `make check` works without hardware. See System Interfaces.

5.4 Building using CMake

CMake supports compiling in many environments, including Visual Studio if CMake support is installed. The commands below can be run in a `Developer Command Prompt`.

```
mkdir build
cd build
# to use installed wolfSSL location (library and headers)
cmake .. -DWITH_WOLFSSL=/prefix/to/wolfssl/install/
# OR to use a wolfSSL source tree
cmake .. -DWITH_WOLFSSL_TREE=/path/to/wolfssl/
# build
cmake --build .
```

Use `-DWITH_WOLFSSL=` when wolfSSL is already installed (library and headers), or `-DWITH_WOLFSSL_TREE=` to build against a wolfSSL source tree. The post-quantum (TPM 2.0 v1.85) options are on the Post-Quantum Cryptography page.

5.5 Bare-metal build

wolfTPM can be built for bare-metal embedded environments where no operating system is present. In this approach you compile the wolfTPM source files directly into your project instead of using auto-tools or CMake. It is common for microcontrollers such as ARM Cortex-M, RISC-V, UltraScale+/Versal, Microblaze, and others.

5.5.1 Prerequisites

- wolfCrypt library source code
- wolfTPM library source code
- A TPM 2.0 module connected via SPI (or I2C)

5.5.2 Step 1: Define preprocessor macros

Add these preprocessor macros to your project build settings or compiler command line:

```
WOLFTPM_USER_SETTINGS
WOLFSSL_USER_SETTINGS
```

These macros tell wolfTPM and wolfSSL to look for a `user_settings.h` file instead of using the autoconf-generated `options.h` file.

5.5.3 Step 2: Create a user_settings.h file

Create a `user_settings.h` file in your project that contains the build configuration options for both wolfSSL and wolfTPM. A reference configuration file is available in the wolfSSL repository: examples/-configs/user_settings_wolftpm.h.

Example `user_settings.h` for wolfTPM:

```
/* System */
#define WOLFSSL_GENERAL_ALIGNMENT 4
#define SINGLE_THREADED
#define WOLFCRYPT_ONLY
```

COPYRIGHT ©2026 wolfSSL Inc.

```
#define SIZEOF_LONG_LONG 8

/* Platform - bare metal */
#define NO_FILESYSTEM
#define NO_WRITEV
#define NO_MAIN_DRIVER
#define NO_DEV_RANDOM
#define NO_ERROR_STRINGS
#define NO_SIG_WRAPPER

/* wolfTPM required features */
#define WOLF_CRYPTO_CB
#define WOLFSSL_PUBLIC_MP
#define WOLFSSL_AES_CFB
#define HAVE_AES_DECRYPT

/* ECC options */
#define HAVE_ECC
#define ECC_TIMING_RESISTANT

/* RSA options */
#undef NO_RSA
#define WOLFSSL_KEY_GEN
#define WC_RSA_BLINDING

/* Big math library */
#define WOLFSSL_SP_MATH_ALL /* sp_int.c */
#define WOLFSSL_SP_SMALL
#define SP_INT_BITS 4096
/* #define SP_WORD_SIZE 32 */

/* SHA options: SHA-256 stays enabled, so do not define NO_SHA256 */
#define WOLFSSL_SHA512
#define WOLFSSL_SHA384

/* Disable unneeded features to reduce footprint */
#define NO_PKCS8
#define NO_PKCS12
#define NO_PWDBASED
#define NO_DSA
#define NO_DES3
#define NO_RC4
#define NO_PSK
#define NO_MD4
#define NO_MD5
#define WOLFSSL_NO_SHAKE128
#define WOLFSSL_NO_SHAKE256
#define NO_DH

/* Other interesting size reduction options */
#if 0
    #define RSA_LOW_MEM
    #define WOLFSSL_AES_SMALL_TABLES
    #define USE_SLOW_SHA
```

```
    #define USE_SLOW_SHA256
    #define USE_SLOW_SHA512
    #define NO_AES_192
#endif

/* Custom random seed source - implement your own */
#define HAVE_HASHDRBG
#define CUSTOM_RAND_GENERATE_SEED my_rng_seed
```

!!! warning The NO_* macros disable algorithms. wolfTPM wrappers and sessions need SHA-256, so never define NO_SHA256 in this file.

If you use CUSTOM_RAND_GENERATE_SEED, implement your own RNG seed function. This example gets the seed from the TPM with parameter encryption enabled:

```
int my_rng_seed(byte* seed, word32 sz)
{
    int rc;

    /* enable parameter encryption for the RNG request */
    rc = wolfTPM2_SetAuthSession(&wolftpm_dev, 0, &wolftpm_session,
        (TPMA_SESSION_decrypt | TPMA_SESSION_encrypt |
        TPMA_SESSION_continueSession));
    if (rc == 0) {
        rc = wolfTPM2_GetRandom(&wolftpm_dev, seed, sz);
    }
    wolfTPM2_UnsetAuthSession(&wolftpm_dev, 0, &wolftpm_session);
    return rc;
}
```

5.5.4 Step 3: Configure include paths

Add these directories to your project's include paths:

1. The wolfSSL root directory, for example /path/to/wolfssl
2. The wolfTPM root directory, for example /path/to/wolftpm
3. The directory that holds your user_settings.h

Example compiler flags:

```
-I/path/to/wolfssl
-I/path/to/wolftpm
-I/path/to/your/project/include
```

5.5.5 Step 4: Add source files

Add the required source files from wolfSSL and wolfTPM to your project.

wolfCrypt source files (minimum required for wolfTPM):

```
wolfssl/wolfcrypt/src/aes.c
wolfssl/wolfcrypt/src/asn.c
wolfssl/wolfcrypt/src/cryptocb.c
wolfssl/wolfcrypt/src/ecc.c
wolfssl/wolfcrypt/src/hash.c
wolfssl/wolfcrypt/src/hmac.c
wolfssl/wolfcrypt/src/random.c
wolfssl/wolfcrypt/src/rsa.c
```

```
wolfssl/wolfcrypt/src/sha.c
wolfssl/wolfcrypt/src/sha256.c
wolfssl/wolfcrypt/src/sha512.c
wolfssl/wolfcrypt/src/sp_int.c
wolfssl/wolfcrypt/src/wc_port.c
wolfssl/wolfcrypt/src/wolfmath.c
```

wolfTPM source files:

```
wolftpm/src/tpm2.c
wolftpm/src/tpm2_util.c
wolftpm/src/tpm2_packet.c
wolftpm/src/tpm2_tis.c
wolftpm/src/tpm2_wrap.c
wolftpm/src/tpm2_asn.c
wolftpm/src/tpm2_crypto.c
wolftpm/src/tpm2_param_enc.c
wolftpm/src/tpm2_cryptocb.c
wolftpm/src/tpm2_linux.c
```

This list matches `src_libwolftpm_la_SOURCES` in `src/include.am`. `tpm2_swtpm.c`, `tpm2_winapi.c` and `tpm2_spdm.c` are only needed for their respective optional builds. The HAL source (one of the `hal/tpm_io*.c` files, see below) is added separately.

5.5.6 Step 5: Implement the SPI HAL callback

wolfTPM needs a single SPI transmit and receive callback to communicate with the TPM module. Implement it for your hardware platform. Reference implementations are in the `hal/` directory of the wolfTPM repository:

- hal/tpm_io_xilinx.c for Xilinx Microblaze
- hal/tpm_io_st.c for STM32
- hal/tpm_io_infineon.c for Infineon Tricore
- hal/tpm_io_microchip.c for Microchip

5.5.6.1 Standard I/O callback

The standard SPI callback has this signature:

```c
typedef int (*TPM2HalIoCb)(
    TPM2_CTX* ctx,
    const byte* txBuf, byte* rxBuf,
    word16 xferSz,
    void* userCtx
);
```

Example implementation:

```c
#include <wolftpm/tpm2.h>
#include <wolftpm/tpm2_tis.h>

int TPM2_IoCb(TPM2_CTX* ctx,
    const byte* txBuf, byte* rxBuf, word16 xferSz,
    void* userCtx)
{
    int ret = TPM_RC_FAILURE;

    /* TODO: Assert SPI chip select */
```

```
    spi_cs_assert();

    /* Perform SPI transfer: send txBuf and receive into rxBuf */
    if (spi_transfer(txBuf, rxBuf, xferSz) == 0) {
        ret = TPM_RC_SUCCESS;
    }

    /* TODO: De-assert SPI chip select */
    spi_cs_deassert();

    (void)ctx;
    (void)userCtx;

    return ret;
}
```

5.5.6.2 Advanced I/O callback For platforms that need more control, enable `WOLFTPM_ADV_IO` to use the advanced callback:

```
typedef int (*TPM2HalIoCb)(
    TPM2_CTX* ctx,
    INT32 isRead, UINT32 addr,
    BYTE* xferBuf, UINT16 xferSz,
    void* userCtx
);
```

This gives access to the register address and the read or write direction, for platforms that need separate read and write operations.

5.5.7 Step 6: Initialize and use wolfTPM

After the setup is complete, initialize wolfTPM and start communicating with the TPM:

```
#include <wolftpm/tpm2_wrap.h>

int main(void)
{
    int rc;
    WOLFTPM2_DEV dev;

    /* Initialize wolfTPM */
    rc = wolfTPM2_Init(&dev, TPM2_IoCb, NULL);
    if (rc != TPM_RC_SUCCESS) {
        /* Handle error */
        return rc;
    }

    /* Get TPM capabilities */
    WOLFTPM2_CAPS caps;
    rc = wolfTPM2_GetCapabilities(&dev, &caps);
    if (rc == TPM_RC_SUCCESS) {
        /* Use TPM ... */
    }

    /* Cleanup */
```

```
    wolfTPM2_Cleanup(&dev);

    return 0;
}
```

5.5.8 Optional build configurations

To reduce the memory footprint in constrained environments, consider these options in `user_settings.h`:

```
/* Reduce stack usage */
#define WOLFTPM_SMALL_STACK

/* Disable wrapper layer if using native API only */
#define WOLFTPM2_NO_WRAPPER

/* Use smaller RSA key sizes only */
#define MAX_RSA_KEY_BITS 2048
```

If you know your TPM module type at compile time, select it. Select exactly one module variant, not several:

```
/* For Infineon, pick exactly one of these */
#define WOLFTPM_SLB9670
/* #define WOLFTPM_SLB9672 */
/* #define WOLFTPM_SLB9673 */

/* For ST ST33 */
#define WOLFTPM_ST33

/* For Nuvoton */
#define WOLFTPM_NUVOTON

/* For Microchip ATTPM20 */
#define WOLFTPM_MICROCHIP
```

If no module is specified, define `WOLFTPM_AUTODETECT` to detect the module at runtime. That is the default under `./configure`, but a `user_settings.h` build with no module and no `WOLFTPM_AUTODETECT` falls back to the Infineon SLB9672 (SPI) or SLB9673 (I2C), so define `WOLFTPM_AUTODETECT` explicitly for runtime detection.

For TPM modules connected via I2C instead of SPI:

```
#define WOLFTPM_I2C
#define WOLFTPM_ADV_IO
```

You must implement the advanced I/O callback for I2C communication.

5.5.9 Cryptographic key storage

In bare-metal environments, the TPM provides secure storage for cryptographic keys, isolated from main processor memory. Key material never leaves the TPM in plaintext form.

- Keys created with `TPM2_CreatePrimary` reside in the TPM and return a handle.
- Keys created with `TPM2_Create` return an encrypted blob that can be stored in non-volatile memory and reloaded using `TPM2_Load`.
- Use `TPM2_EvictControl` to store keys persistently in the TPM NVRAM.

This keeps cryptographic keys protected even if the main processor memory is compromised.

5.5.10 Troubleshooting

SPI communication issues:

1. Verify SPI clock polarity and phase (typically CPOL=0, CPHA=0 for a TPM).
2. Check the SPI clock speed. Start slow (1 to 10 MHz) and increase.
3. Verify chip select is asserted low during the entire send and receive.
4. Some TPMs require wait states during SPI operations, which means extra bytes are read until the MSB is set to signal response readiness (enabled with `WOLFTPM_CHECK_WAIT_STATE`).
5. Enable debug output with `DEBUG_WOLFTPM` (general), `WOLFTPM_DEBUG_VERBOSE` (detailed), or `WOLFTPM_DEBUG_IO` (SPI and I2C transactions).

Build errors:

1. Ensure `WOLFSSL_USER_SETTINGS` and `WOLFTPM_USER_SETTINGS` are defined.
2. Verify the include paths are correct.
3. Check that all required source files are included in the build.

5.6 See Also

- Getting Started
- Build Options
- Supported Hardware
- System Interfaces

6 Build Options

This page is the reference for the Autotools (`./configure`) options and the preprocessor defines that control how wolfTPM is built. Each configure switch lists the macro it defines when one exists. `configure.ac` in the wolfTPM source tree is the authoritative source.

!!! note This page documents the Autotools build. The CMake build differs: fwTPM is off by default and the TPM interface is chosen with the `WOLFTPM_INTERFACE` cache variable (`auto`, `SWTPM`, `WINAPI`, `DEVTPM`, `SPI`, `I2C` or `MMIO`) instead of `--enable-*` flags. See Building wolfTPM for CMake.

6.1 Reading the tables

- Every `--enable-X` flag also has a `--disable-X` form. The default column shows the state when the flag is not given.
- A few macros are opt-outs. `WOLFTPM2_NO_WRAPPER` is defined by `--disable-wrapper`, and `WOLFTPM2_NO_WOLFCRYPT` is defined by `--disable-wolfcrypt`. The enable forms do not define them.
- The generated `wolftpm/options.h` records the macros chosen at configure time.

6.2 General and debug

Option	Default	Effect and macros
`--enable-debug[=yes|no|verbose|io]`	no	Adds debug code and turns off optimizations. Defines `DEBUG_WOLFTPM`. `verbose` also defines `WOLFTPM_DEBUG_VERBOSE`, and `io` defines both `WOLFTPM_DEBUG_VERBOSE` and `WOLFTPM_DEBUG_IO`.
`--enable-examples`	enabled	Build the example programs.
`--enable-wrapper`	enabled	Build the wrapper API. `--disable-wrapper` defines `WOLFTPM2_NO_WRAPPER`.
`--enable-wolfcrypt`	enabled	Use wolfCrypt for RNG, authorization sessions and parameter encryption. `--disable-wolfcrypt` defines `WOLFTPM2_NO_WOLFCRYPT`.
`--with-wolfcrypt=PATH`	`/usr/local`	Path to the wolfSSL install. The directory must contain `lib` and `include`.

Option	Default	Effect and macros
`--enable-smallstack`	disabled	Defines `WOLFTPM_SMALL_STACK` to reduce stack usage. Also sets `MAX_COMMAND_SIZE=1024`, `MAX_RESPONSE_SIZE=1350` and `MAX_DIGEST_BUFFER=896`. With `--disable-wolfcrypt` it also sets `MAX_SESSION_NUM=1`.
`--enable-provisioning`	enabled	Support for provisioning Initial Device Identity (IDevID) and Attestation Identity Keys. Defines `WOLFTPM_PROVISIONING`.
`--enable-firmware`	enabled	TPM firmware upgrade support for Infineon SLB9672/SLB9673 and ST ST33. Defines `WOLFTPM_FIRMWARE_UPGRADE`. Use `--disable-firmware` to remove it.
`--enable-fuzz`	disabled	Build the fuzz targets.

!!! warning `WOLFTPM_DEBUG_SECRETS` is not set by any configure option and is off by default. Defining it manually prints sensitive material such as auth values, session keys, bind keys, HMAC keys, hierarchy auth and encryption secrets. Use it only for developer debugging. Never enable it in production builds or on devices that log stdout to persistent storage.

6.3 I/O layer and bus selection

Option	Default	Effect and macros
`--enable-spi`	not set	Intent signal for a SPI hardware build. SPI is the default transport when `--enable-i2c` is not set. Adds no macro, but counts as a hardware selection and so turns off the automatic swTPM and fwTPM defaults. Cannot be combined with `--enable-i2c`.
`--enable-i2c`	disabled	I2C TPM support. Defines `WOLFTPM_I2C` and automatically defines `WOLFTPM_ADV_IO`.

Option	Default	Effect and macros
`--enable-mmio`	disabled	Built-in memory-mapped I/O callbacks. Defines `WOLFTPM_MMIO` and automatically defines `WOLFTPM_ADV_IO`.
`--enable-advio`	disabled	Advanced I/O callback signature. Defines `WOLFTPM_ADV_IO`. You do not need to pass it separately with I2C or MMIO.
`--enable-wolfhal`	disabled	wolfHAL I/O callbacks. Defines `WOLFTPM_WOLFHAL`. Requires the wolfHAL headers and an application-provided `wolfHAL_board.h`. See `hal/README.md` for the required `BOARD_*` definitions.
`--enable-hal`	enabled	Build the example HAL interfaces. Defines `WOLFTPM_EXAMPLE_HAL`.
`--enable-hal-reset[=LINE]`	disabled	TPM nRST reset HAL through the Linux GPIO character device. Always defines `WOLFTPM_HAL_RESET`. A numeric `LINE` also defines `WOLFTPM_RESET_LINE`. Without a line the default is GPIO24 for ST33 and GPIO4 for Nuvoton. Drive it with `TPM2_IoCb_Reset()`.
`--enable-checkwaitstate`	depends on chip	TIS and SPI check-wait-state support. Defines `WOLFTPM_CHECK_WAIT_STATE`. Configure turns it on for autodetect and for every build that is not Infineon-only.
`--enable-tislock`	disabled	Defines `WOLFTPM_TIS_LOCK`. Uses a named semaphore to serialize TIS commands across processes. Linux only.

`--enable-hal-reset` needs the SPI or I2C hardware HAL. Configure rejects it together with swTPM or `--enable-devtpm`. Because swTPM is the default on common hosts, pass `--enable-spi` or `--enable-i2c` with it.

!!! note For I2C support on a Raspberry Pi you may need to enable I2C first:

```
1. Edit `/boot/firmware/config.txt` on current Raspberry Pi OS (for example `
   sudo vim /boot/firmware/config.txt`). Older images use `/boot/config.txt`.
2. Uncomment `dtparam=i2c_arm=on`.
3. Reboot with `sudo reboot`.
```

6.4 TPM vendors and modules

Option	Default	Effect and macros
`--enable-infineon[=slb9670|slb9672|slb9673|slb9678]`	disabled	Plain `--enable-infineon` selects SLB9672 and defines `WOLFTPM_SLB9672`. `--enable-infineon=slb9670` defines `WOLFTPM_SLB9670`. `--enable-infineon=slb9673` defines `WOLFTPM_SLB9673` and is I2C only: use `--enable-i2c` and do not pass `--enable-spi`. `--enable-infineon=slb9678` defines `WOLFTPM_SLB9678`, which takes its bus from a boot-time configuration pin, so pair it with either `--enable-spi` or `--enable-i2c`.
`--enable-st33`, `--enable-st`	disabled	ST ST33 support. Defines `WOLFTPM_ST33`. The two flags are equivalent.
`--enable-microchip`, `--enable-mchp`	disabled	Microchip ATTPM20 support. Defines `WOLFTPM_MICROCHIP`. The two flags are equivalent.
`--enable-nuvoton`	disabled	Nuvoton NPCT65x/NPCT75x support. Defines `WOLFTPM_NUVOTON`.
`--enable-nations`	disabled	Nations Technology NS350 support. Defines `WOLFTPM_NATIONS`.
`--enable-sealsq`	disabled	SealSQ QVault post-quantum TPM support. Defines `WOLFTPM_SEALSQ`.
`--enable-autodetect`	on when no vendor module is selected	Runtime module detection. Defines `WOLFTPM_AUTODETECT`.

Selecting an Infineon device with its argument looks like this:

```
./configure --enable-infineon=slb9670
./configure --enable-infineon=slb9673 --enable-i2c
```

6.5 Operating system transports and simulators

Option	Default	Effect and macros
`--enable-devtpm`	disabled	Use the Linux kernel driver (`/dev/tpmrm0` or `/dev/tpm0`). Defines `WOLFTPM_LINUX_DEV`. Cannot be combined with swTPM.
`--enable-swtpm`	see below	Talk to a simulator over the swtpm TCP protocol. Defines `WOLFTPM_SWTPM` and `TPM2_SWTPM_PORT`.
`--enable-swtpm=uart`	disabled	swtpm protocol over a UART serial port, for fwTPM on embedded targets such as STM32H5. Defines `WOLFTPM_SWTPM`, `WOLFTPM_SWTPM_UART` and `TPM2_SWTPM_PORT`, where the port value is the baud rate (default 115200).
`--with-swtpm-port=PORT`	2321	Sets `TPM2_SWTPM_PORT`. For `--enable-swtpm=uart` it sets the baud rate instead.
`--enable-fwtpm`	see below	Build the firmware TPM (fwTPM) server. Requires wolfCrypt.
`--enable-winapi` (alias `--enable-wintbs`)	disabled	Use the Windows TBS API. Defines `WOLFTPM_WINAPI`. Cannot be combined with swTPM or devtpm.

6.5.1 Default simulator behavior

`--enable-swtpm` and `--enable-fwtpm` are on by default when all of these hold:

- the host CPU is x86_64, amd64 or aarch64;
- the host OS is not Windows (mingw, cygwin, msys, win32);
- wolfCrypt is enabled;
- no hardware path was selected with any of `--enable-spi`, `--enable-i2c`, `--enable-mmio`, `--enable-devtpm`, `--enable-autodetect`, `--enable-winapi`, `--enable-infineon`, `--enable-st`, `--enable-st33`, `--enable-microchip`, `--enable-nuvoton`, `--enable-nations` or `--enable-sealsq`.

This applies to macOS and BSD as well as Linux. Everywhere else the default is off.

!!! warning A bare `./configure` on these hosts can define both `WOLFTPM_AUTODETECT` and `WOLFTPM_SWTPM`. `WOLFTPM_SWTPM` suppresses the kernel-device detection, so such a build never tries `/dev/tpmrm0` or `/dev/tpm0`. To use a real TPM, pass `--enable-autodetect`, `--enable-devtpm` or a vendor flag explicitly. Explicit `--enable-autodetect` prevents the simulator default and gives the kernel-first behavior: on Linux it tries `/dev/tpmrm0` or `/dev/tpm0` at runtime and falls back to SPI if the kernel driver is not available.

6.5.2 fwTPM macros

Macro	Where it is set
WOLFTPM_FWTPM_BUILD	Added to the generated options.h for any fwTPM build. This is the marker that test scripts look for.
WOLFTPM_FWTPM	Set only on the fwTPM server and fuzz targets. It gates server-side code in the shared sources.
WOLFTPM_FWTPM_HAL, WOLFTPM_ADV_IO	Added for TIS and shared-memory builds, that is fwTPM without --enable-swtpm. Not supported on Windows.

6.5.3 fwTPM-only and NV modes

Option	Default	Effect and macros
--enable-fwtpm-only	disabled	Build only the fwTPM server. Skips the client library, wrapper and examples, and defines WOLFTPM2_NO_WRAPPER. Implies --enable-fwtpm and needs wolfCrypt. Not compatible with --enable-spdm.
--enable-fwtpm-nv-appendonly	disabled	Append-only NV journal mode for write-once flash fwTPM ports. Defines WOLFTPM_FWTPM_NV_APPEND_ONLY.

6.6 SPDM

Option	Default	Effect and macros
--enable-spdm	disabled	SPDM support. Defines WOLFTPM_SPDM. Needs the wolfSPDM submodule: run git submodule update --init lib/wolfSPDM. With fwTPM it also defines WOLFTPM_SPDM_RESPONDER.
--enable-tcg	auto under --enable-spdm	SPDM TCG Binding mode. Defines WOLFTPM_SPDM_TCG. Turned on automatically with fwTPM, Nuvoton or Nations.
--enable-psk	auto with Nations	SPDM PSK mode. Defines WOLFTPM_SPDM_PSK. Requires --enable-tcg.

Related rules enforced by configure:

- --enable-tcg and --enable-psk require --enable-spdm.
- --enable-nuvoton with SPDM requires --enable-tcg and defines WOLFSPDM_NUVOTON.

- `--enable-nations` with SPDM requires both `--enable-tcg` and `--enable-psk` and defines `WOLFSPDM_NATIONS`.
- fwTPM with SPDM requires at least one of `--enable-tcg` or `--enable-psk`.
- `--with-wolfspdm` is removed and now fails configure. Use `--enable-spdm`.
- A debug build with SPDM also defines `WOLFSPDM_DEBUG`.

6.7 Post-quantum (v1.85)

Option	Default	Effect and macros
`--enable-v185`	auto-detect	Full TPM 2.0 v1.85 features: ML-DSA and ML-KEM, sign and verify sequence and digest commands, new response codes and capability properties. Defines `WOLFTPM_V185`. Auto-enabled for fwTPM builds when wolfCrypt has both ML-DSA and ML-KEM.
`--enable-pqc`	auto-detect	Lean post-quantum subset (ML-DSA and ML-KEM only). Defines `WOLFTPM_PQC`. A fwTPM build promotes it to full v1.85. `--enable-v185` wins if both are given.
`--enable-mldsa[=all|sign-only|verify-only|no]`	all	Limit ML-DSA. `sign-only` defines `WOLFTPM_NO_MLDSA_VERIFY`, `verify-only` defines `WOLFTPM_NO_MLDSA_SIGN`, and no defines `WOLFTPM_NO_MLDSA`.
`--enable-mlkem[=all|enc|dec|no]`	all	Limit ML-KEM. enc defines `WOLFTPM_NO_MLKEM_DECAP`, dec defines `WOLFTPM_NO_MLKEM_ENCAP`, and no defines `WOLFTPM_NO_MLKEM`.
`--disable-hash-mldsa`	pre-hash enabled	Drops pre-hash ML-DSA key support. Defines `WOLFTPM_NO_HASH_MLDSA`.

Use `--disable-v185` or `--disable-pqc` to turn off post-quantum support, including the auto-detect. Setting both `--enable-mldsa=no` and `--enable-mlkem=no` is an error. With wolfCrypt enabled, PQC needs wolfSSL 5.9.2-stable or later built with ML-DSA (`--enable-mldsa`, or the wolfSSL alias `--enable-dilithium`) and ML-KEM (`--enable-mlkem`). With `--disable-wolfcrypt`, PQC is command marshaling only.

6.8 Preprocessor defines

These are not configure options. Set them with `CFLAGS`, for example `./configure CFLAGS="-DWOLFTPM_MAX_RETRIES=3"`.

Macro	Effect
WOLFTPM_USE_SYMMETRIC	Enables symmetric AES, hashing and HMAC support for the TLS examples.
WOLFTPM2_USE_SW_ECDHE	Stops the TLS examples from using the TPM for ECC ephemeral key generation and the shared secret.
TLS_BENCH_MODE	Enables TLS benchmarking mode.
NO_TPM_BENCH	Disables the TPM benchmarking example.
WOLFTPM2_ECC_DEFAULT_CURVE	Curve used by the named wrapper templates that do not take an explicit curve, currently SRK and AIK. Defaults to TPM_ECC_NIST_P256, or the smallest enabled curve that meets ECC_MIN_KEY_SZ. Override with -DWOLFTPM2_ECC_DEFAULT_CURVE=TPM_ECC_NIST_P384. wolfTPM2_GetKeyTemplate_ECC and _ECC_ex take an explicit curve, so this macro does not remap them, except that P-256 is substituted when NO_ECC256 is set.
WOLFTPM_MAX_RETRIES	Default number of times a command is resubmitted when the TPM returns TPM_RC_RETRY (the TPM is momentarily busy, for example while persisting the daUsed flag on first authorization use of an externally provisioned key without noDA). Default 0, which is disabled. Opt in at runtime with TPM2_SetCommandRetries() or at build time with -DWOLFTPM_MAX_RETRIES=N. wolfTPM does not set noDA on every key it creates: the generic key-template APIs use the attributes the caller passes in, and the EK template omits noDA, so those keys can hit this condition.
WOLFTPM_NO_RETRY	Compiles out the TPM_RC_RETRY resubmit handling. TPM_RC_RETRY is returned to the caller. Conflicts with WOLFTPM_MAX_RETRIES greater than 0.
WOLFSSL_MLDSA_ALLOW_WEAK_PREHASH	Allows HashML-DSA to use a pre-hash weaker than the parameter set requires. This compatibility option is for pre-production TPM firmware and makes the build non-compliant with FIPS 204 section 5.4. It is disabled by default and unavailable in FIPS builds. Define it when building both wolfSSL and wolfTPM.
WOLFTPM_LOCALITY_DEFAULT	TIS locality requested at startup (default 0). Change it at runtime with wolfTPM2_SetLocality() on SPI, memory-mapped and swtpm transports. The wolfTPM I2C HAL does not implement locality selection and uses locality 0 only, so a non-zero wolfTPM2_SetLocality() on I2C returns NOT_COMPILED_IN.

Macro	Effect
WOLFTPM_TIS_RESET_STALE_LOCALITY	At startup, releases any active locality other than WOLFTPM_LOCALITY_DEFAULT so the default can be granted. Recovers a TPM left wedged when a previous session did not release its locality. Off by default. Use only on single-master buses, since on a shared bus it could clear a locality another master holds. The nRST reset HAL is the alternative.
WOLFTPM_LOCALITY_TIMEOUT_TRIES	Poll attempts when requesting a locality at runtime (default 1000). Kept small so a locality that cannot be granted fails fast.
WOLFTPM_RESET_LINE	nRST GPIO line number for the reset HAL. Set it with --enable-hal-reset=LINE.

6.9 See Also

- Building wolfTPM
- System Interfaces
- Supported Hardware
- Getting Started

7 System Interfaces

Besides talking directly to a TPM chip over SPI or I2C, wolfTPM can reach a TPM through an operating system interface or a software simulator. This page covers the three of them: the software TPM simulators (SWTPM), the Linux kernel device (`/dev/tpmX`), and the Windows TBS API. Only one transport can be enabled in a given build.

7.1 Software simulator (SWTPM)

wolfTPM can use a software TPM defined by section D.3 of TPM-Rev-2.0-Part-4-Supporting-Routines-01.38-code.

Software TPM implementations tested:

- Official TCG Reference: reference code from the specification maintained by TCG. See TCG TPM.
- IBM (ibmswtpm2) / Ken Goldman: a fork of the reference code maintained by IBM (93% identical to the official TCG code). See ibmswtpm2.
- Microsoft ms-tpm-20-ref: a fork of the reference code maintained by Microsoft (100% identical to the official TCG code). See ms-tpm-20-ref.
- libtpms/swtpm by Stefan Berger: uses the libtpms front end interfaces. See swtpm.

The software TPM transport is a socket connection by default. A UART is also supported. This implementation only uses the TPM command interface, typically on port 2321. It does not support the platform interface, typically on port 2322.

7.1.1 wolfTPM SWTPM support

To enable the socket transport for SWTPM use `--enable-swtpm`. By default all software TPM simulators use TCP port 2321.

```
./configure --enable-swtpm
make
```

!!! note It is not possible to enable more than one transport interface at a time. When building with the SWTPM socket interface, the built-in TIS and devtpm (`/dev/tpm0`) interfaces are not available.

Build options:

- `WOLFTPM_SWTPM`: use the socket transport (no TIS layer)
- `TPM2_SWTPM_HOST`: the socket host (default is localhost)
- `TPM2_SWTPM_PORT`: the socket port (default is 2321)

7.1.2 wolfTPM SWTPM UART support

To use the SWTPM protocol over a UART serial connection instead of TCP sockets, use `--enable-swtpm=uart`. This is intended for talking to a firmware TPM (fwTPM) running on an embedded target, such as the wolfTPM fwTPM server on STM32H5.

```
./configure --enable-swtpm=uart
make
```

The serial device path and baud rate can be set at compile time or at runtime:

```
# Runtime override via environment variable
TPM2_SWTPM_HOST=/dev/ttyACM0 ./examples/wrap/caps
```

Build options:

- WOLFTPM_SWTPM_UART: use the UART serial transport (set automatically by --enable-swtpm=uart)
- TPM2_SWTPM_HOST: the serial device path (default is /dev/ttyACM0 on Linux and /dev/cu.usbmodem on macOS). It can be overridden at runtime with the TPM2_SWTPM_HOST environment variable.
- TPM2_SWTPM_PORT: the baud rate (default is 115200)
- WOLFTPM_SWTPM_UART_RX_TIMEOUT_SECS: cumulative wall-clock timeout for a single receive (default is 60)
- WOLFTPM_SWTPM_UART_VTIME: per-read termios VTIME, in tenths of a second (default is 200, so 20 seconds)

The UART transport uses the same mssim protocol as the socket transport. The serial port is configured as 8N1 raw mode with no flow control. A receive waits for the device with poll() until WOLFTPM_SWTPM_UART_RX_TIMEOUT_SECS elapses, so a command taking longer than one VTIME period, such as RSA key generation on an MCU, still completes, while a device hangup fails immediately instead of waiting out the timeout. Like the socket transport, the serial port file descriptor is kept open across commands (no reconnect per command). Both transports close the connection during wolfTPM2_Cleanup. On the socket transport, any transmit or receive failure also closes the connection so the next command reconnects. The UART transport closes only when the per-command TPM_SESSION_END write fails.

7.1.2.1 Security note: environment variable override The TPM2_SWTPM_HOST environment variable is a development convenience that overrides the compile-time serial device path. On systems where untrusted local users share the environment with the TPM client, an attacker could redirect TPM I/O to a rogue device, such as a PTY they control. For production and hardened deployments:

- Unset TPM2_SWTPM_HOST in the process environment.
- Rely on the compile-time default (set via TPM2_SWTPM_HOST as a build -D macro) to pin the serial path.

The same guidance applies to TPM2_SWTPM_PORT (baud rate) and, for the socket transport, to using the environment variable to redirect the TCP host.

7.1.2.2 Example: wolfTPM fwTPM on STM32H5 The wolfTPM project includes a firmware TPM server port for STM32 Cortex-M33 targets with TrustZone support. See wolftpm-examples/STM32/fwtpm-stm32h5 for build, flash, and test instructions.

```
# Build host client with UART transport
./configure --enable-swtpm=uart
make

# Run examples against STM32 fwTPM (adjust device path as needed)
export TPM2_SWTPM_HOST=/dev/ttyACM0
./examples/wrap/caps
./examples/keygen/keygen -ecc
./examples/seal/seal
```

7.1.3 Using a SWTPM

7.1.3.1 SWTPM Power Up and Startup The TCG TPM and Microsoft ms-tpm-20-ref implementations require power up and startup commands on the platform interface before the command interface is enabled. Use these commands to issue the required power up and startup:

```
echo -ne "\x00\x00\x00\x01" | nc 127.0.0.1 2322
echo -ne "\x00\x00\x00\x0B" | nc 127.0.0.1 2322
```

7.1.3.2 TCG TPM

```
git clone git@github.com:TrustedComputingGroup/TPM.git
cd TPM
cd TPMCmd
./bootstrap
./configure
make
```

Run with `./Simulator/src/tpm2-simulator`, then run power on and self test. See SWTPM Power Up and Startup.

7.1.3.3 ibmswtpm2

```
git clone https://github.com/kgoldman/ibmswtpm2.git
cd ibmswtpm2/src/
make
```

Run with `./tpm_server`.

!!! note You can use the `-rm` switch to remove the cache file NVChip. Alternatively, delete the NVChip file (`rm NVChip`).

7.1.3.4 ms-tpm-20-ref

```
git clone https://github.com/microsoft/ms-tpm-20-ref
cd ms-tpm-20-ref/TPMCmd
./bootstrap
./configure
make
```

Run with `./Simulator/src/tpm2-simulator`, then run power on and self test. See SWTPM Power Up and Startup.

7.1.3.5 swtpm Build libtpms:

```
git clone git@github.com:stefanberger/libtpms.git
cd libtpms
./autogen.sh --with-tpm2 --with-openssl --prefix=/usr
make install
```

Build swtpm:

```
git clone git@github.com:stefanberger/swtpm.git
cd swtpm
./autogen.sh
make install
```

On macOS, do the following first:

```
brew install openssl socat
pip3 install cryptography

export LDFLAGS="-L/usr/local/opt/openssl@1.1/lib"
export CPPFLAGS="-I/usr/local/opt/openssl@1.1/include"

# libtpms had to use --prefix=/usr/local
```

Run swtpm:

```
mkdir -p /tmp/myvtpm
swtpm socket --tpmstate dir=/tmp/myvtpm --tpm2 --ctrl type=tcp,port=2322
 ↪  --server type=tcp,port=2321 --flags not-need-init
```

7.1.3.6 swtpm with QEMU This demonstrates using wolfTPM in QEMU to communicate through the Linux kernel device /dev/tpmX. You need to install or build swtpm. A short build method is below. You may need to consult the instructions for libtpms and swtpm.

```
PREFIX=$PWD/inst
git clone git@github.com:stefanberger/libtpms.git
cd libtpms/
./autogen.sh --with-openssl --with-tpm2 --prefix=$PREFIX && make install
cd ..
git clone git@github.com:stefanberger/swtpm.git
cd swtpm
PKG_CONFIG_PATH=$PREFIX/lib/pkgconfig/ ./autogen.sh --with-openssl --with-tpm2
 ↪  \
    --prefix=$PREFIX && \
  make install
cd ..
```

Set up a basic Linux installation. Other installation bases can be used. This step takes some time to install the base Linux system.

```
# download mini install image
curl -O http://archive.ubuntu.com/ubuntu/dists/bionic-updates/main/installer-
 ↪  amd64/current/images/netboot/mini.iso
# create qemu image file
qemu-img create -f qcow2 lubuntu.qcow2 5G
# create directory for tpm state and socket
mkdir $PREFIX/mytpm
# start swtpm
$PREFIX/bin/swtpm socket --tpm2 --tpmstate dir=$PREFIX/mytpm \
  --ctrl type=unixio,path=$PREFIX/mytpm/swtpm-sock --log level=20 &
# start qemu for installation
qemu-system-x86_64 -m 1024 -boot d -bios bios-256k.bin -boot menu=on \
  -chardev socket,id=chrtpm,path=$PREFIX/mytpm/swtpm-sock \
  -tpmdev emulator,id=tpm0,chardev=chrtpm \
  -device tpm-tis,tpmdev=tpm0 -hda lubuntu.qcow2 -cdrom mini.iso
```

Once a base system is installed, start QEMU again and build wolfSSL and wolfTPM in the QEMU instance.

```
# start swtpm again
$PREFIX/bin/swtpm socket --tpm2 --tpmstate dir=$PREFIX/mytpm \
  --ctrl type=unixio,path=$PREFIX/mytpm/swtpm-sock --log level=20 &
# start qemu system to install and run wolfTPM
qemu-system-x86_64 -m 1024 -boot d -bios bios-256k.bin -boot menu=on \
  -chardev socket,id=chrtpm,path=$PREFIX/mytpm/swtpm-sock \
  -tpmdev emulator,id=tpm0,chardev=chrtpm \
  -device tpm-tis,tpmdev=tpm0 -hda lubuntu.qcow2
```

In the QEMU terminal, check out and build wolfTPM:

```
sudo apt install automake libtool gcc git make

# get and build wolfSSL
```

```
git clone https://github.com/wolfssl/wolfssl.git
pushd wolfssl
./autogen.sh && \
  ./configure --enable-wolftpm --disable-examples --prefix=$PWD/../inst && \
  make install
popd

# get and build wolfTPM
git clone https://github.com/wolfssl/wolftpm.git
pushd wolftpm
./autogen.sh && \
  ./configure --enable-devtpm --prefix=$PWD/../inst --enable-debug && \
  make install
sudo make check
popd
```

You can now run examples such as sudo ./examples/wrap/wrap within QEMU. sudo may be required for access to /dev/tpm0.

7.1.4 Running examples

```
./examples/wrap/caps
./examples/pcr/extend
./examples/wrap/wrap_test
```

See examples/README.md in the source tree for additional example usage.

7.2 Linux kernel device (/dev/tpmX)

On Linux the kernel TPM driver stack exposes a TPM through a character device, and wolfTPM can use it directly instead of driving SPI or I2C itself. This is the right transport whenever the kernel already owns the TPM: a discrete chip bound to a kernel driver, a Windows-style firmware TPM, or a TEE-resident firmware TPM such as the one on NVIDIA Jetson platforms.

With --enable-devtpm there is no TIS layer and no HAL IO callback. hal/tpm_io.c is compiled out entirely and TPM2_IoCb is NULL (see hal/tpm_io.h), so pass NULL for the callback argument of TPM2_Init and wolfTPM2_Init.

With --enable-autodetect this is not the case. The TIS/SPI HAL stays compiled in on purpose, because it is the fallback, and TPM2_IoCb is a real function. Keep passing it, or the SPI fallback that build exists to provide is unreachable.

7.2.1 Two device nodes

The kernel presents up to two nodes per TPM:

- /dev/tpm0: the raw device. One user at a time, no resource management. Whatever you send reaches the TPM.
- /dev/tpmrm0: the in-kernel resource manager (kernel 4.12 and later, practical from 5.12). It virtualizes handles, swaps transient objects and sessions in and out as needed, and flushes everything belonging to a connection when that connection closes.

wolfTPM prefers /dev/tpmrm0 and falls back to /dev/tpm0. The resource manager is the better default: a TPM has very few transient object slots, and without it a program that leaks a handle wedges the TPM for everything else on the system.

Build-time overrides, honored by both --enable-devtpm and --enable-autodetect:

- `-DWOLFTPM_USE_TPMRM`: use `/dev/tpmrm0` only, with no fallback to the raw device.
- `CFLAGS='-DTPM2_LINUX_DEV="/dev/tpm1"'`: use a specific node. The inner quotes are required, because the macro is used directly as a C string literal and an unquoted value does not compile.

7.2.2 Startup, shutdown, and shared state

The TPM is started by firmware long before Linux runs, and on the resource manager it is shared with every other process on the system. Restarting or shutting it down is therefore not an individual caller's decision, so wolfTPM stays out of the way on this transport:

- `wolfTPM2_Init` skips the startup and self-test sequence.
- `wolfTPM2_Reset` and `wolfTPM2_Shutdown` send no TPM command and return `NOT_COMPILED_IN` (-174), the same way `wolfTPM2_SetLocality` does on this transport. A `wolfTPM2_Reset(dev, 0, 0)` that asked for neither a shutdown nor a startup still returns `TPM_RC_SUCCESS`, since nothing was declined. Treat `NOT_COMPILED_IN` here as "the OS owns this", not as a failure.
- `wolfTPM2_SetLocality` returns `NOT_COMPILED_IN` because the kernel owns the locality.

The kernel does not reliably stop you here. Command filtering on `/dev/tpmrm0` is primarily about handle isolation, not about blocking global state changes, and behavior varies by kernel version and TPM implementation. On Linux 5.15 with the Jetson OP-TEE fTPM, a `TPM2_Shutdown(TPM_SU_CLEAR)` sent through the resource manager is passed straight through and returns success, both from wolfTPM and from `tpm2_shutdown`. So this is a case where the library declining to send the command is what protects other users of the TPM, rather than the kernel doing it for you.

If you need to control TPM startup state, you need `/dev/tpm0` and exclusive use of the TPM, or direct SPI access with the wolfTPM TIS driver.

7.2.3 What the native API does on autodetect builds

Two behaviors matter if you use `TPM2_Init` or `TPM2_Init_ex` directly rather than the `wolfTPM2_*` wrapper.

The kernel device wins over your callback. If `/dev/tpmrm0` or `/dev/tpm0` opens, every command is routed there and the HAL IO callback you passed is never invoked. On a host that has both a kernel-bound TPM and a discrete SPI part, that means you now talk to a different TPM than a pre-autodetect build did. `--enable-spi` on its own does not pin SPI, because it leaves autodetect on and the kernel device still wins. Pin the part you want with `--enable-devtpm`, `--enable-<vendor>`, `--enable-spi --disable-autodetect`, or `-DTPM2_LINUX_DEV`.

Init now acquires a descriptor. `TPM2_Init*` opens the device on autodetect builds, and `TPM2_Cleanup()` is what closes it. Native callers that skipped cleanup previously leaked nothing, but now they leak a descriptor per context. This matters most on hosts exposing only the raw `/dev/tpm0`, which permits a single open. A context that merely initialized holds the TPM exclusively for its lifetime, and a second context in the same process falls through to a different transport.

`TPM2_Init_minimal()` is unaffected: it performs no IO and still succeeds with no device present.

7.2.4 Transient handles do not outlive a process

This is the difference most likely to break an existing application.

On `/dev/tpmrm0` the kernel gives each open file description its own handle space. Transient object handles are virtualized (the value the TPM assigned is not the value you get back), and everything in that space is flushed when the file descriptor closes. A transient key created by one process is gone by the time a second process runs, and the handle number it printed is meaningless to anyone else.

Creating a primary key on the Jetson fTPM through the resource manager returns:

```
Create Primary Handle: 0x80ffffff
```

This is not the `0x80000000` a raw device would report. Query the transient handles from a separate process afterwards and the list is empty:

```
tpm2_getcap handles-transient      # no output, the space was torn down
```

Two practical consequences:

- A "create a key, keep it, use it from the next command" workflow does not work across processes. Do the whole sequence in one process, or make the object persistent with `TPM2_EvictControl` so it gets a stable `0x81xxxxxx` handle that does survive.
- Passing a hard-coded transient handle such as `0x80000000` on a command line fails. The kernel rejects the reference before it reaches the TPM, and because that happens at the file-descriptor layer the error surfaces as `errno 22 = Invalid argument` on `read()`, which wolfTPM reports as `TPM_RC_FAILURE` rather than as a handle error. If you see `TPM_RC_FAILURE` alongside `Failed to read from /dev/tpmrm0 ... errno 22`, suspect a stale or cross-process transient handle before suspecting the TPM.

The wolfTPM script `examples/run_examples.sh` hits exactly this: its provisioning section creates IAK and IDevID primaries with `-keep` in one process and then references `0x80000000` and `0x80000001` from another. That block cannot pass on the resource manager by construction. Everything on either side of it is unaffected. Use `/dev/tpm0` with exclusive access if you need to run it as written.

7.2.5 Building

```
./autogen.sh
./configure --enable-devtpm
make
```

`--enable-devtpm` uses the kernel node only. Use `--enable-autodetect` instead if you want wolfTPM to try `/dev/tpmrm0`, then `/dev/tpm0`, and finally fall back to probing SPI. This is useful for one binary that has to run on several boards.

Only one transport can be enabled at a time. `--enable-devtpm` conflicts with `--enable-swtpm` and `--enable-winapi`, and configure stops if you ask for more than one.

7.2.5.1 The x86_64 and aarch64 default A bare `./configure` on Linux x86_64 or aarch64 does not produce a build that talks to `/dev/tpmX`. On those hosts wolfTPM auto-enables the software TPMs (swTPM and fwTPM) so that `make check` passes with no hardware attached, and defining `WOLFTPM_SWTPM` suppresses the kernel-device autodetect path. The result talks to a simulator on TCP port 2321.

Selecting any hardware path explicitly turns that default back off: `--enable-autodetect`, `--enable-devtpm`, `--enable-spi`, `--enable-i2c`, `--enable-mmio`, `--enable-winapi`, `--enable-wintbs`, or any `--enable-<vendor>`. Configure prints a notice when the software default is taken, so check the end of its output if a build unexpectedly fails to find your TPM.

This matters most on single-board aarch64 machines with a firmware TPM, where the kernel device is the only transport there is.

7.2.6 Permissions

The TPM character devices are not world-accessible. On a typical system they are mode `0660` owned by group `tss`:

```
crw-rw---- 1 tss root  10,   224 /dev/tpm0
crw-rw---- 1 tss tss  252, 65536 /dev/tpmrm0
```

wolfTPM detects EACCES and reports it plainly:

```
Permission denied on /dev/tpm0
Use sudo or add tss group to user.
```

The fix is to put your user in the owning group and start a new login session:

```
sudo usermod -aG tss $USER
```

The `tss` group is created by tpm2-tss. On distributions that ship it, the group frequently exists with no members, so this step is required even though the group looks correctly set up.

To use a group of your own instead, add a udev rule.

1. Create the group and add your user:

   ```
   sudo addgroup wolftpm
   sudo adduser [username] wolftpm
   ```

2. Create `/etc/udev/rules.d/wolftpm-udev.rules` containing:

   ```
   KERNEL=="tpm[0-9]*", TAG+="systemd", MODE="0660", GROUP="wolftpm"
   ```

3. Reload the rules with `sudo udevadm control -R`, then re-plug or reboot.

7.2.7 NVIDIA Jetson Orin (Tegra234) firmware TPM

Jetson Orin platforms carry a TPM 2.0 implemented in firmware, running as a trusted application inside OP-TEE rather than as a discrete package on a bus. Linux reaches it through the `tpm_ftpm_tee` driver, which speaks to the TA over the TEE interface and registers an ordinary TPM chip. From the point of view of wolfTPM it is just another `/dev/tpmrm0`.

Confirm the device is present before building:

```
lsmod | grep tpm_ftpm_tee
ls -l /dev/tpm*
cat /sys/class/tpm/tpm0/tpm_version_major     # expect 2
```

If the module is missing, try `sudo modprobe tpm_ftpm_tee` and check that the kernel was configured with `CONFIG_TCG_FTPM_TEE`. On NVIDIA Jetson Linux (L4T) images the driver is present and an `fTPM Device Provisioning Service` systemd unit runs at boot. You can see it complete in the boot log.

An OP-TEE boot message about silicon-identity fTPM provisioning not being enabled refers to a separate NVIDIA feature. It does not mean the TPM 2.0 device is unavailable.

Build as above with `--enable-devtpm` or `--enable-autodetect`, then confirm with:

```
./examples/wrap/caps
```

Because this is a firmware TPM, expect two differences from a discrete part. There is no TIS bus, so the `TPM2: Caps/Did/Vid/Rid` values do not exist and the device is identified purely from `TPM2_GetCapability` properties. Under `--enable-devtpm` the `DEBUG_WOLFTPM` line is still printed but reads all zeros. Under `--enable-autodetect`, `wolfTPM2_Init_ex` returns as soon as the kernel device opens, before that printf, so the line is absent entirely. Also, the algorithm coverage of a firmware TPM is set by its firmware build rather than by a datasheet, so check it rather than assuming. Where an operation is absent, the benchmark reports it as unsupported rather than failing. The Jetson Orin fTPM supports every operation the benchmark exercises.

7.2.8 Testing

The examples run unchanged on this transport:

```
./examples/wrap/caps
./examples/native/native_test
./examples/wrap/wrap_test
./examples/bench/bench
./examples/run_examples.sh
```

`run_examples.sh` already skips the locality test on backends that do not support it.

7.2.9 CI coverage

Both `--enable-devtpm` and `--enable-autodetect` are build-tested in CI, but not run, because GitHub-hosted runners have no `/dev/tpm*` node. Runtime coverage of this transport requires a self-hosted runner with a real TPM bound to the kernel driver.

7.3 Windows TBS API

wolfTPM can be built to use the Windows native TBS (TPM Base Services). When using the Windows TBS interface, NV access is blocked by default. TPM NV storage space is very limited, and when it fills up it can cause undefined behavior, such as failures loading key handles. NV space is not managed by TBS.

The TPM is designed to return an encrypted private key blob on key creation using `TPM2_Create`, which you can safely store on disk and load when needed. The symmetric encryption key used to protect the private key blob is only known by the TPM. When you load a key using `TPM2_Load` you get a transient handle, which can be used for signing and for encryption and decryption.

For primary keys created with `TPM2_CreatePrimary` you get back a handle. No encrypted private data is returned. That handle remains loaded until `TPM2_FlushContext` is called.

For normal key creation using `TPM2_Create` you get back a `TPM2B_PRIVATE outPrivate`, which is the encrypted blob that you can store and load at any time using `TPM2_Load`.

7.3.1 Limitations

wolfTPM has been tested on Windows 10 with TPM 2.0 devices. Windows does support TPM 1.2, but functionality is limited and wolfTPM does not support it.

The presence of a TPM 2.0 can be checked by opening PowerShell and running `Get-PnpDevice -Class SecurityDevices`:

```
Status     Class           FriendlyName
------     -----           ------------
OK         SecurityDevices Trusted Platform Module 2.0
Unknown    SecurityDevices Trusted Platform Module 2.0
```

7.3.2 Building in MSYS2

Tested using MSYS2:

```
export PREFIX=$PWD/tmp_install

cd wolfssl
./autogen.sh
```

```
./configure --prefix="$PREFIX" --enable-wolftpm
make
make install

cd wolftpm/
./autogen.sh
./configure --prefix="$PREFIX" --enable-winapi
make
./examples
```

To install the development base tools on MSYS2 use `pacman -S base-devel` and `pacman -S mingw-w64-x86_64-toolchain`.

7.3.3 Building on Linux

Tested using mingw-w32-bin_x86_64-linux_20131221.tar.bz2 from the MinGW-w64 Win32 toolchain builds.

Extract the tools and add them to the PATH:

```
mkdir mingw_tools
cd mingw_tools
tar xjvf ../mingw-w32-bin_x86_64-linux_20131221.tar.bz2
export PATH=$PWD/bin/:$PWD/i686-w64-mingw32/bin:$PATH
cd ..
```

Build:

```
export PREFIX=$PWD/tmp_install
export CFLAGS="-DWIN32 -DMINGW -D_WIN32_WINNT=0x0600 -DUSE_WOLF_STRTOK"
export LIBS="-lws2_32"

cd wolfssl
./autogen.sh
./configure --host=i686 CC=i686-w64-mingw32-gcc --prefix="$PREFIX"
 ↪  --enable-wolftpm
make
make install

cd ../wolftpm/
./autogen.sh
./configure --host=i686 CC=i686-w64-mingw32-gcc --prefix="$PREFIX"
 ↪  --enable-winapi
make
cd ..
```

7.3.4 Running on Windows

To confirm the presence and status of the TPM on the machine, run `tpm.msc`. See `examples/README.md` in the source tree for running the examples.

7.4 See Also

- Getting Started
- Building wolfTPM
- Build Options

- Supported Hardware

8 Supported Hardware

wolfTPM talks to TPM 2.0 parts over SPI or I2C through a single HAL I/O callback, or to a system TPM through the operating system driver. This page lists the platforms wolfTPM has HAL backends for, the hardware it has been tested with, the configure flag for each part, and the build steps for each vendor.

8.1 Platform

The hardware examples are most often run on a Raspberry Pi using the Linux `spidev` interface. That is the example platform, not a requirement. The Linux HAL picks a default SPI chip select per vendor: Infineon builds use `/dev/spidev0.1`, while Microchip, ST, Nuvoton, Nations Technologies and SEALSQ builds use `/dev/spidev0.0`. Override the device with `TPM2_SPI_DEV_PATH` and `TPM2_SPI_DEV_CS` if your wiring differs.

To interface with your hardware bus (SPI or I2C), wolfTPM uses a single HAL callback. You pass it during initialization when calling `TPM2_Init` or `wolfTPM2_Init`. See HAL IO Callback for the callback model.

Example HAL implementations are provided in the `hal` directory for:

- Atmel ASF (`tpm_io_atmel.c`)
- Barebox (`tpm_io_barebox.c`)
- Espressif ESP-IDF (`tpm_io_espressif.c`)
- Firmware TPM (`tpm_io_fwtpm.c`)
- Infineon TriCore and PSoC/CyHAL (`tpm_io_infineon.c`)
- Linux SPI and I2C (`tpm_io_linux.c`)
- Memory-mapped I/O (`tpm_io_mmio.c`)
- Microchip Harmony (`tpm_io_microchip.c`)
- QNX (`tpm_io_qnx.c`)
- STM32 CubeMX (`tpm_io_st.c`)
- U-Boot (`tpm_io_uboot.c`)
- wolfHAL (`tpm_io_wolfhal.c`)
- Xilinx (`tpm_io_xilinx.c`)
- Zephyr (`tpm_io_zephyr.c`)

An advanced I/O option (`--enable-advio` or `WOLFTPM_ADV_IO`) adds the register address and a read-/write flag as parameters to the I/O callback. This is required for I2C support, and `--enable-i2c` turns it on.

8.2 Tested Hardware

wolfTPM has been tested with:

- Infineon OPTIGA(TM) Trusted Platform Module 2.0 SLB9670 (SPI), SLB9672 (SPI), SLB9673 (I2C) and SLB9678 (SPI or I2C).
  - LetsTrust is a vendor of TPM development boards.
- STMicroelectronics ST33KTPM2XSPI, ST33KTPM2I, ST33TPHF2XSPI (SPI) and ST33TPHF2XI2C (I2C).
- Microchip ATTPM20 module.
- Nuvoton NPCT650 and NPCT750 TPM 2.0 modules.
- Nations Technologies Z32H330 and NS350 TPM 2.0 modules.
- SEALSQ QVault TPM 2.0 module (SPI, post-quantum ML-DSA and ML-KEM).
- NVIDIA Jetson Orin (Tegra234) firmware TPM: a TPM 2.0 running as an OP-TEE trusted application, reached through the Linux kernel driver rather than a bus. See System Interfaces.

The firmware updater also recognizes the ST33KTPM2A firmware line, but that part is not in the tested list.

COPYRIGHT ©2026 wolfSSL Inc.

Device identification is printed in two steps. A direct bus connection first prints a `TPM2: Caps ... Did ... Vid ... Rid` line read from the TIS registers. A second `Mfg ...` line then reports the manufacturer, vendor string, firmware version and certification flags. A firmware TPM has no TIS registers, so it prints only the second line. See TPM 2.0 Overview for captured output from each tested module.

8.3 Supported Parts

Vendor	Part(s)	Bus	configure flag	Notes
Infineon	SLB9670	SPI	`--enable-infineon=slb9670`	Library SPI clock default 43 MHz. AES key size limited to 128 bits.
Infineon	SLB9672	SPI	`--enable-infineon`	Default for SPI. Library SPI clock default 33 MHz. Supports firmware upgrade.
Infineon	SLB9673	I2C	`--enable-infineon=slb9673 --enable-i2c --enable-advio`	I2C only, so no SPI clock applies.
Infineon	SLB9678	SPI or I2C	`--enable-infineon=slb9678`	Bus selected by a boot-time configuration pin, not a build option, so pair with either `--enable-spi` or `--enable-i2c`. Add `--enable-spdm` for the TCG asymmetric secure channel.
STMicroelectronics	ST33KTPM2XSPI, ST33TPHF2XSPI	SPI	`--enable-st33`	Library SPI clock default 33 MHz. Wait states required.

Vendor	Part(s)	Bus	configure flag	Notes
STMicroelectronics	ST33KTPM2I, ST33TPHF2XI2C	I2C	`--enable-st33 --enable-i2c`	Wait states required. Firmware upgrade support is on by default.
Microchip	ATTPM20	SPI	`--enable-microchip`	Library SPI clock default 33 MHz. Wait states required.
Nuvoton	NPCT650, NPCT750	SPI	`--enable-nuvoton`	Library SPI clock default 43 MHz. Wait states required.
Nations Technologies	Z32H330, NS350	SPI	`--enable-nations`	Wait states required (`WOLFTPM_CHECK_WAIT_STAT` enabled for Nations builds).
SEALSQ	QVault TPM 2.0	SPI	`--enable-sealsq`	Library SPI clock default 33 MHz. Wait states required. Post-quantum commands also need `--enable-pqc` or `--enable-v185`.
NVIDIA	Jetson Orin (Tegra234) firmware TPM	None (kernel driver)	`--enable-autodetect` or `--enable-devtpm`	Accessed through `/dev/tpmrm0`, falling back to `/dev/tpm0`. No bus flag.

The SPI clocks in this table are library defaults from `wolftpm/tpm2_types.h`, not the electrical limits of each part. The part limits are set by the vendor datasheets and can be lower or higher:

- **Infineon SLB9670:** 43 MHz is allowed only at 3.3 V with a sufficiently fast SCLK edge. The limit is

lower at 1.8 V or with slower edges.
- **Infineon SLB9672:** the datasheet gives 33 MHz nominal and 34.65 MHz maximum.
- **STMicroelectronics:** ST33KTPM2X supports up to 66 MHz, ST33KTPM2I up to 48 MHz and ST33TPHF2XSPI up to 33 MHz. wolfTPM uses 33 MHz for all ST parts.
- **Microchip ATTPM20:** rated to 36 MHz, but wolfTPM uses 33 MHz because of issues at the higher rate.
- **SEALSQ QVault:** the datasheet lists 33 MHz in its summary and 36 MHz in its timing table. wolfTPM uses the conservative 33 MHz.

!!! note The part limits above come from vendor datasheets and were not checked against code in this repository. Confirm them against the current datasheet for your part.

To change the SPI clock, define `TPM2_SPI_MAX_HZ` at build time, for example `CFLAGS="-DTPM2_SPI_MAX_HZ=20000000"`. The default for a selected vendor is set in `wolftpm/tpm2_types.h`. I2C builds on Linux default to 400 kHz (`TPM2_I2C_HZ`).

8.4 Autodetect and the Kernel Device

When no vendor flag is given, `--enable-autodetect` is on by default. It detects the module at run time. With autodetect, wolfTPM turns on wait state checking and caps the SPI clock at 33 MHz, the lowest default among the supported parts. This cap applies when wolfTPM falls back to direct SPI access, where it tries `/dev/spidev0.0` through `/dev/spidev0.4`.

On Linux, autodetect and `--enable-devtpm` first try the kernel TPM device. The driver opens `/dev/tpmrm0` (the resource manager, kernel 5.12 and later) and falls back to `/dev/tpm0` if it is not present. Define `WOLFTPM_USE_TPMRM` to use only the resource manager, or set `TPM2_LINUX_DEV` to pin a specific device. See System Interfaces.

8.5 Building per Vendor

All builds start from a clone of the repository:

```
git clone https://github.com/wolfSSL/wolfTPM.git
cd wolfTPM
```

8.5.1 Infineon

Supports SLB9670 or SLB9672 (SPI) and SLB9673 (I2C). Pick one of the following.

SLB9672 on SPI (the default for `--enable-infineon`):

```
./autogen.sh
./configure --enable-infineon
make
```

SLB9670 on SPI:

```
./autogen.sh
./configure --enable-infineon=slb9670
make
```

SLB9673 on I2C:

```
./autogen.sh
./configure --enable-infineon=slb9673 --enable-i2c --enable-advio
make
```

8.5.2 STMicroelectronics ST33

SPI parts (ST33KTPM2XSPI, ST33TPHF2XSPI):

```
./autogen.sh
./configure --enable-st33
make
```

I2C parts (ST33KTPM2I, ST33TPHF2XI2C):

```
./autogen.sh
./configure --enable-st33 --enable-i2c
make
```

Firmware upgrade support is enabled by default (`--enable-firmware`), which builds the `st33_fw_update` example tool. Pass `--disable-firmware` to leave it out.

Raspberry Pi wiring: the ST33KTPM2X SPI device is `/dev/spidev0.0`, with nRST (active low) on GPIO24 (pin 18). Nuvoton uses GPIO4. You can optionally drive nRST from code with `--enable-hal-reset` and `TPM2_IoCb_Reset()`. See HAL IO Callback.

8.5.3 Microchip ATTPM20

```
./autogen.sh
./configure --enable-microchip
make
```

8.5.4 Nuvoton

```
./autogen.sh
./configure --enable-nuvoton
make
```

8.5.5 Nations Technologies

Pass `--enable-nations`. Without it, a default `./configure` does not define `WOLFTPM_NATIONS`, so the Nations settings and vendor commands are not built. Z32H330 and NS350 are the tested modules. The NS350 Raspberry Pi TPM 2.0 module uses `/dev/spidev0.0`. Wait states are required and are turned on for Nations builds through `WOLFTPM_CHECK_WAIT_STATE`.

```
./autogen.sh
./configure --enable-nations
make
```

8.5.6 SEALSQ QVault

Build with `--enable-sealsq` for the ordinary TPM commands. The ML-DSA and ML-KEM commands are gated separately, so also pass `--enable-pqc` (lean post-quantum subset) or `--enable-v185` (full v1.85 command set), which require a wolfSSL build with ML-DSA and ML-KEM. The post-quantum build options are covered in Post-Quantum Support.

```
./autogen.sh
./configure --enable-sealsq --enable-pqc
make
```

COPYRIGHT ©2026 wolfSSL Inc.

8.5.7 Espressif ESP-IDF

The ESP-IDF component needs a wolfSSL source tree. If CMake cannot find one, it stops with the error "Could not find wolfssl". Place a wolfSSL checkout in a parent directory named `wolfssl`, `wolfssl-master` or `wolfssl-<user>`, or point the `WOLFSSL_ROOT` variable at it. The wolfSSL ESP Registry managed component also works as an alternative.

The wolfTPM-specific settings are in the wolfSSL `user_settings.h` file, typically found in `[project]/components/wolfssl/include`.

```
git clone https://github.com/wolfSSL/wolfssl.git
git clone https://github.com/wolfSSL/wolfTPM.git
cd wolfTPM/IDE/Espressif

# set your path to ESP-IDF, shown here for VisualGDB using v5.2
WRK_IDF_PATH=/mnt/c/SysGCC/esp32/esp-idf/v5.2

. "${WRK_IDF_PATH}/export.sh"
idf.py build
```

8.6 See Also

- HAL IO Callback
- System Interfaces
- TPM 2.0 Overview
- Post-Quantum Support

9 HAL IO Callback

A single hardware abstraction layer (HAL) callback must be registered to handle communication to the TPM hardware. This page describes the callback, the example implementations distributed with wolfTPM, and the build options that control them.

Examples for several platforms are provided to help with initial setup. If you use one of the built-in, system-provided hardware interfaces, `NULL` can be supplied for the HAL IO callback.

The available system TPM interfaces are:

- Linux `/dev/tpm0`: enabled with `WOLFTPM_LINUX_DEV` or `--enable-devtpm`.
- Windows TBS: enabled with `WOLFTPM_WINAPI` or `--enable-winapi`.
- Software TPM simulator: enabled with `WOLFTPM_SWTPM` or `--enable-swtpm`.

If you use a HAL IO callback, it is registered on library initialization with:

- TPM2 native API: `TPM2_Init`
- wolfTPM wrappers: `wolfTPM2_Init`

9.1 Example HAL Implementations

Platform	Example File	Build Option
Atmel ASF	`tpm_io_atmel.c`	`WOLFSSL_ATMEL`
Barebox	`tpm_io_barebox.c`	`__BAREBOX__`
Infineon	`tpm_io_infineon.c`	`WOLFTPM_INFINEON_TRICORE`
Linux	`tpm_io_linux.c`	`__linux__`
Microchip	`tpm_io_microchip.c`	`WOLFTPM_MICROCHIP_HARMONY`
QNX	`tpm_io_qnx.c`	`__QNX__`
ST Cube HAL	`tpm_io_st.c`	`WOLFSSL_STM32_CUBEMX`
wolfHAL	`tpm_io_wolfhal.c`	`WOLFTPM_WOLFHAL`
Xilinx	`tpm_io_xilinx.c`	`__XILINX__`

9.2 wolfHAL

Enabled with `WOLFTPM_WOLFHAL` or `--enable-wolfhal`. The wolfHAL headers must be on the include path.

This HAL is placed last in the platform selection chain, so it is used only when no other platform macro is defined. For example, building for an STM32 target with the CubeMX headers present selects `tpm_io_st.c` instead.

9.2.1 Board definitions

wolfTPM does not ship board definitions. `tpm_io_wolfhal.c` includes `"wolfHAL_board.h"`, which the application provides on its include path. A wolfHAL project already has one, so in most cases only the TPM-specific entries below need to be added to it.

For SPI:

Macro	Type	Description
`BOARD_SPI_DEV`	`whal_Spi*`	SPI instance the TPM is connected to
`BOARD_SPI_COM_CFG`	`whal_Spi_ComCfg*`	SPI session parameters
`BOARD_GPIO_DEV`	`whal_Gpio*`	GPIO instance driving chip select

Macro	Type	Description
BOARD_CS_PIN	pin number	Chip select pin, driven active low

For I2C (also requires WOLFTPM_ADV_IO, which --enable-i2c sets):

Macro	Type	Description
BOARD_I2C_DEV	whal_I2c*	I2C instance the TPM is connected to
BOARD_I2C_COM_CFG	whal_I2c_ComCfg*	I2C session parameters, including the TPM target address

The TPM target address goes in the addr field of BOARD_I2C_COM_CFG. Most TPM 2.0 I2C parts use 0x2e. The TPM2_I2C_ADDR macro that the other I2C HALs use has no effect here, so defining it is a compile-time error.

A TPM 2.0 I2C part takes roughly 80 us to wake and NAKs until it is ready, so each transfer is retried up to TPM_I2C_TRIES times (default 10). Define TPM_I2C_TRIES to override this.

A missing entry is reported at compile time, naming the macro that is required. Only the macros needed by the selected bus are checked.

Example additions to an existing wolfHAL_board.h:

```
/* TPM on SPI1, chip select on PA15 */
extern whal_Spi_ComCfg g_tpmSpiComCfg;
#define BOARD_SPI_COM_CFG  (&g_tpmSpiComCfg)
#define BOARD_CS_PIN       15
```

For I2C, where the session config carries the TPM address:

```
/* board.c */
whal_I2c_ComCfg g_tpmI2cComCfg = {
    .freq   = 400000, /* Hz */
    .addr   = 0x2e,   /* TPM target address */
    .addrSz = 7,      /* bits */
};

/* wolfHAL_board.h */
extern whal_I2c_ComCfg g_tpmI2cComCfg;
#define BOARD_I2C_COM_CFG  (&g_tpmI2cComCfg)
```

9.3 HAL IO Callback Function

The prototypes for the HAL callback function:

```
#ifdef WOLFTPM_ADV_IO
typedef int (*TPM2HalIoCb)(struct TPM2_CTX*, INT32 isRead, UINT32 addr,
    BYTE* xferBuf, UINT16 xferSz, void* userCtx);
#else
typedef int (*TPM2HalIoCb)(struct TPM2_CTX*, const BYTE* txBuf, BYTE* rxBuf,
    UINT16 xferSz, void* userCtx);
#endif
```

Example function definitions:

```
#ifdef WOLFTPM_ADV_IO
int TPM2_IoCb(TPM2_CTX*, int isRead, word32 addr, byte* buf, word16 size,
    void* userCtx);
#else
int TPM2_IoCb(TPM2_CTX* ctx, const byte* txBuf, byte* rxBuf,
    word16 xferSz, void* userCtx);
#endif
```

9.4 Additional Build Options

- `WOLFTPM_CHECK_WAIT_STATE`: Enables checking of the wait state during a SPI transaction. Most TPM 2.0 chips require this and typically need only 0 to 2 wait cycles depending on the command. Only the Infineon TPMs guarantee no wait states.
- `WOLFTPM_ADV_IO`: Enables advanced IO callback mode, which includes the TIS register and a read/write flag. This is required for I2C, but can be used with SPI also.
- `WOLFTPM_DEBUG_IO`: Enables logging of the IO (if using the example HAL).
- `WOLFTPM_HAL_RESET`: Optional TPM hardware reset (nRST) control in the example HAL (`--enable-hal-reset`). On Linux, `TPM2_IoCb_Reset(&dev->ctx, userCtx)` pulses nRST (active low) through the GPIO char device (raw GPIO v2 uAPI, no libgpiod).

9.5 TPM Reset (nRST) HAL Macros

These apply when `WOLFTPM_HAL_RESET` is set.

- `WOLFTPM_RESET_GPIOCHIP`: GPIO char device. Default: `/dev/gpiochip0`.
- `WOLFTPM_RESET_LINE`: GPIO line wired to nRST. Default: ST33 uses 24 (GPIO24, Pi pin 18), Nuvoton uses 4 (GPIO4). Also settable with `--enable-hal-reset=<line>`.
- `WOLFTPM_RESET_HOLD_US` and `WOLFTPM_RESET_SETTLE_US`: reset hold time and post-reset settle time in microseconds. Defaults: `300000` and `1000000`.

9.6 Additional Compiler Macros

- `TPM2_SPI_DEV_PATH`: The device string to be opened by the Linux IO callback. Default: `"/dev/spidev0."`.
- `TPM2_SPI_DEV_CS`: The chip select number string to use. Default: `"0"`.

These can be set during configure:

```
./configure CPPFLAGS="-DTPM2_SPI_DEV_PATH=\"/dev/spidev0.\"
 ↪  -DTPM2_SPI_DEV_CS=\"0\""
```

Autodetect uses `TPM2_SPI_DEV_PATH[0..4]` for the searched device paths.

9.7 See Also

- Supported Hardware
- TPM 2.0 Overview

10 Examples Overview

The wolfTPM examples show how to use a TPM 2.0 module through both the native TPM2_* API and the wolfTPM2_* wrapper API. They build with the library and are ready to run after a successful install. To connect them to your hardware platform, see the TPM2_IoCb function in tpm_io.c and the HAL I/O callback guide.

The examples create RSA and ECC keys in NV for testing, using the handles defined in ./examples/tpm_test.h (see Run flags and test handles). The PKCS #7 and TLS examples need CSRs generated and signed with a test script. See the CSR and certificate signing section of examples/README.md for the steps.

Some examples are vendor specific, such as the extra GPIO examples for the ST33 and NPCT75x TPMs. These only work on that hardware.

10.1 Native API test

Demonstrates calling the native TPM2_* APIs.

```
./examples/native/native_test
```

10.2 Wrapper API test

Demonstrates calling the wolfTPM2_* wrapper APIs.

```
./examples/wrap/wrap_test
```

10.3 Crypto primitive examples

Small, focused examples for common TPM crypto operations:

```
./examples/wrap/getrandom [bytes]
./examples/wrap/hash [-sha384|-sha512]
./examples/wrap/encrypt_decrypt [-aescfb|-aesctr|-aescbc]
./examples/keygen/ecdh
```

Command	What it does
getrandom [bytes]	Gets random bytes from the TPM RNG (default 32).
hash	Hashes a message with a TPM hash sequence. SHA-256 is the default; use -sha384 or -sha512 to change it.
encrypt_decrypt	Symmetric encrypt/decrypt round trip. AES-CFB is the default.
ecdh	ECDH (P-256) key agreement that produces a shared secret.

!!! note Many TPMs disable TPM2_EncryptDecrypt entirely because of export controls. The encrypt_decrypt example skips gracefully when the command is unavailable.

10.4 Parameter encryption

To enable parameter encryption in the examples, use `-aes` for AES-CFB mode or `-xor` for XOR mode. Only some TPM commands and responses support it. If the `TPM2_` API entry has `CMD_FLAG_ENC2` or `CMD_FLAG_DEC2` set in its flags, the command uses parameter encryption or decryption.

Only the first parameter of a TPM command can be encrypted, and it must be of type `TPM2B_DATA`. Examples are the password auth of a TPM key or the qualifying data of a TPM2.0 Quote. The request and the response can be encrypted together or separately. The `sessionAttributes` control this:

- `TPMA_SESSION_decrypt` for the command request
- `TPMA_SESSION_encrypt` for the command response

Either one can be set alone, or both can be set in the same authorization session. This is up to the developer.

Examples that use parameter encryption:

- Key generation with an encrypted authorization value. See Key Management.
- A secure vault for keys with encrypted NV authorization. See Sealing and NVRAM.
- A TPM2.0 Quote with encrypted user data. The qualifying data supplied for a Quote is arbitrary data that goes into the signed Quote structure. With parameter encryption the host sends it to the TPM in encrypted form, which protects it from man-in-the-middle attacks. See Attestation.

10.4.1 Post-quantum session keys (v1.85)

On a v1.85 PQC capable TPM, the parameter encryption session can be keyed with a post-quantum primary instead of an RSA or ECC storage key. ML-KEM is decrypt capable and is used as the session salt key. ML-DSA is sign only and is used as the session bind key. The RSA or ECC storage key, where one is needed (for example as the parent of a created child), is unchanged.

Pass `-mlkem[=512|768|1024]` to salt the session with an ML-KEM key, or `-mldsa[=44|65|87]` to bind it to an ML-DSA key. These flags are accepted by `wrap_test`, `pcr/quote`, `nvram/store` and `nvram/counter`. The keygen example uses `-paramkey=mlkem[=...]` and `-paramkey=mldsa[=...]` instead, because `-mlkem` and `-mldsa` there select the child key type.

```
./examples/wrap/wrap_test -aes -mlkem=768
./examples/pcr/quote 16 quote.blob -ecc -xor -mldsa=65
./examples/nvram/counter -aes -mldsa=65
./examples/keygen/keygen keyblob.bin -ecc -aes -paramkey=mlkem=768
```

10.5 Run flags and test handles

The handles used by the examples are defined in `./examples/tpm_test.h`.

Define	Value	Purpose
`TPM2_DEMO_STORAGE_KEY_HANDLE`	`0x81000200`	Persistent storage key (RSA)
`TPM2_DEMO_STORAGE_EC_KEY_HANDLE`	`0x81000201`	Persistent storage key (ECC)
`TPM2_DEMO_PERSISTENT_KEY_HANDLE`	`0x81000202`	Persistent key for common use
`TPM2_DEMO_HMAC_KEY_HANDLE`	`0x81000210`	Persistent HMAC key

The RSA and ECC test keys and certificates use an index offset added to a base address:

Define	Index	Handle	Type
`TPM2_DEMO_RSA_KEY_HANDLE`	`0x20`	`0x81000000 + 0x20`	Persistent key

Define	Index	Handle	Type
TPM2_DEMO_RSA_CERT_HANDLE	0x20	0x01800000 + 0x20	NV index
TPM2_DEMO_ECC_KEY_HANDLE	0x21	0x81000000 + 0x21	Persistent key
TPM2_DEMO_ECC_CERT_HANDLE	0x21	0x01800000 + 0x21	NV index

!!! warning To run the TLS server and client examples on the same machine, build wolfTPM with `WOLFTPM_TIS_LOCK` (`./configure --enable-tislock`). It adds a named semaphore that protects concurrent access to the SPI device between processes.

10.6 See Also

- Key Management
- Attestation
- Sealing and NVRAM
- HAL I/O callback guide

11 Key Management

wolfTPM includes example programs for creating TPM keys, storing them to disk as key blobs, importing external keys, and loading them back into a temporary TPM handle. This page walks through the key generation examples and lists the programs in `examples/keygen/` and the wrapper utilities in `examples/wrap/`.

11.1 Key generation overview

The keygen example creates a TPM key under the storage key (SRK) and writes the key blob to disk. The keyload example reads that blob and loads it into a temporary TPM handle.

```
$ ./examples/keygen/keygen keyblob.bin -rsa
TPM2.0 Key generation example
Loading SRK: Storage 0x81000200 (282 bytes)
Creating new RSA key...
Created new key (pub 280, priv 222 bytes)
Wrote 840 bytes to keyblob.bin

$ ./examples/keygen/keyload keyblob.bin
TPM2.0 Key load example
Loading SRK: Storage 0x81000200 (282 bytes)
Reading 840 bytes from keyblob.bin
Loaded key to 0x80000001


$ ./examples/keygen/keygen keyblob.bin -ecc
TPM2.0 Key generation example
Loading SRK: Storage 0x81000200 (282 bytes)
Creating new ECC key...
Created new key (pub 88, priv 126 bytes)
Wrote 744 bytes to keyblob.bin

$ ./examples/keygen/keyload keyblob.bin
TPM2.0 Key load example
Loading SRK: Storage 0x81000200 (282 bytes)
Reading 744 bytes from keyblob.bin
Loaded key to 0x80000001
```

Symmetric and keyed hash keys use the same flow:

```
$ ./examples/keygen/keygen -sym=aescfb128
TPM2.0 Key generation example
    Key Blob: keyblob.bin
    Algorithm: SYMCIPHER
         aescfb mode, 128 keybits
    Template: Default
    Use Parameter Encryption: NULL
Loading SRK: Storage 0x81000200 (282 bytes)
Symmetric template
Creating new SYMCIPHER key...
Created new key (pub 50, priv 142 bytes)
Wrote 198 bytes to keyblob.bin

$ ./examples/keygen/keyload
```

```
TPM2.0 Key load example
    Key Blob: keyblob.bin
    Use Parameter Encryption: NULL
Loading SRK: Storage 0x81000200 (282 bytes)
Reading 198 bytes from keyblob.bin
Reading the private part of the key
Loaded key to 0x80000001

$ ./examples/keygen/keygen -keyedhash
TPM2.0 Key generation example
    Key Blob: keyblob.bin
    Algorithm: KEYEDHASH
    Template: Default
    Use Parameter Encryption: NULL
Loading SRK: Storage 0x81000200 (282 bytes)
Keyed Hash template
Creating new KEYEDHASH key...
TPM2_Create key: pub 48, priv 158
Public Area (size 48):
  Type: KEYEDHASH (0x8), name: SHA256 (0xB), objAttr: 0x40460, authPolicy sz: 0
  Keyed Hash: scheme: HMAC (0x5), scheme hash: SHA256 (0xB), unique size 32
TPM2_Load Key Handle 0x80000001
New key created and loaded (pub 48, priv 158 bytes)
Wrote 212 bytes to keyblob.bin
```

When no filename is given, the default `keyblob.bin` is used. That means keygen and `keyload` can run without extra parameters for a quick demonstration. Use one of the `--help` switches to see the full list of algorithms and options keygen supports.

The `keyimport` example takes a private key, wraps it as a TPM key blob and stores it to disk. It can then be loaded with `keyload`:

```
$ ./examples/keygen/keyimport keyblob.bin -rsa
TPM2.0 Key import example
Loading SRK: Storage 0x81000200 (282 bytes)
Imported key (pub 278, priv 222 bytes)
Wrote 840 bytes to keyblob.bin

$ ./examples/keygen/keyload keyblob.bin
TPM2.0 Key load example
Loading SRK: Storage 0x81000200 (282 bytes)
Reading 840 bytes from keyblob.bin
Loaded key to 0x80000001


$ ./examples/keygen/keyimport keyblob.bin -ecc
TPM2.0 Key Import example
Loading SRK: Storage 0x81000200 (282 bytes)
Imported key (pub 86, priv 126 bytes)
Wrote 744 bytes to keyblob.bin

$ ./examples/keygen/keyload keyblob.bin
TPM2.0 Key load example
Loading SRK: Storage 0x81000200 (282 bytes)
Reading 744 bytes from keyblob.bin
Loaded key to 0x80000001
```

keyload takes only one argument, the filename of the stored key. It does not need to be told the key type because the RSA or ECC scheme is stored inside the key blob.

To protect the authorization value while creating a key, add -aes or -xor to keygen. See Examples Overview.

11.2 Programs (examples/keygen/)

Program	Description
create_primary.c	Creates and stores primary keys, including endorsement hierarchy keys such as the IAK and IDevID.
keygen.c	Creates a new RSA, ECC, symmetric or keyed hash key under the SRK and writes the key blob to disk.
keyload.c	Reads a key blob from disk and loads it into a temporary TPM handle.
keyimport.c	Imports an existing private key as a TPM key blob and writes it to disk.
external_import.c	Imports an external private key (built into the example) under the SRK. Use -rsa or -ecc for the SRK type, and -load to load the saved keyblob.bin to a third level key.
ecdh.c	ECDH key agreement using a TPM key, producing a shared secret.

11.3 Wrapper utilities (examples/wrap/)

Program	Description
wrap_test.c	Exercises the wolfTPM2_* wrapper APIs.
caps.c	Reads and prints TPM capabilities.
getrandom.c	Gets random bytes from the TPM RNG.
hash.c	Hashes a message with a TPM hash sequence.
hmac.c	Computes an HMAC with a persistent TPM HMAC key, creating the key if it is not found.
encrypt_decrypt.c	Symmetric encrypt/decrypt round trip with a TPM key.

11.4 Storing keys in NV

Keys and secrets can also be stored in the TPM's NV memory, optionally with an encrypted authorization value. See Sealing and NVRAM.

11.5 See Also

- Examples Overview
- Sealing and NVRAM
- Attestation

12 Attestation

wolfTPM includes examples for local and remote attestation, TPM signed timestamps, endorsement key certificates and device identity keys. This page covers the remote attestation challenge flow, PCR quotes, signed timestamps, EK certificate validation and the manufacturer identity keys.

12.1 Remote attestation overview

Remote attestation is the process of a client giving evidence to an attestation server, which checks that the client is in a known state. For this to work, the client and server must first establish trust. This is done with the standard TPM 2.0 commands `TPM2_MakeCredential` and `TPM2_ActivateCredential`.

1. The client sends the server the public parts of a TPM 2.0 Primary Attestation Key (PAK) and a quote signing Attestation Key (AK).
2. `MakeCredential` uses the public part of the PAK to encrypt a challenge (a secret). Typically the challenge is a digest of the public part of the AK. Only the TPM that can load the private part of the PAK and AK can decrypt it. Both keys carry the `fixedTPM` attribute, so the only TPM that can load them is the one where they were created.
3. The server sends the challenge to the client.
4. `ActivateCredential` uses the loaded PAK and AK to decrypt the challenge and recover the secret. The client can then respond to the server.

This proves to the server that the client holds the expected TPM identity and attestation key.

!!! note The transport used to exchange the challenge and response is up to the developer, because it is implementation specific. One option is a TLS 1.3 client-server connection using wolfSSL.

The examples used in this flow:

Program	Role
`./examples/attestation/make_credential`	Used by a server to create a remote attestation challenge.
`./examples/attestation/activate_credential`	Used by a client to decrypt the challenge and respond.
`./examples/attestation/certify`	Certifies (attests) that an object with a given name is loaded in the TPM.
`./examples/keygen/create_primary`	Creates a primary key (PK) and attestation key (AK).

All of these examples accept `-eh` to use the Endorsement Key and an Attestation Key under the Endorsement Hierarchy. The private part of the EK never leaves the TPM, and the EK is unique to each TPM chip, so a challenge encrypted to the EK can only be opened by that TPM. The drawback is privacy: the EK identifies the TPM, so the identity of the host under attestation is always known. The examples support both an AK under the SRK and an AK under the EK. The developer chooses which to use.

12.2 Creating keys for attestation

Use the keygen example to create the TPM 2.0 Attestation Key and the Primary Storage Key that serves as the Primary Attestation Key (PAK).

```
$ ./examples/keygen/keygen -rsa
TPM2.0 Key generation example
    Key Blob: keyblob.bin
```

```
    Algorithm: RSA
    Template: AIK
    Use Parameter Encryption: NULL
Loading SRK: Storage 0x81000200 (282 bytes)
RSA AIK template
Creating new RSA key...
New key created and loaded (pub 280, priv 222 bytes)
Wrote 508 bytes to keyblob.bin
Wrote 288 bytes to srk.pub
Wrote AK Name digest
```

The files written here (`keyblob.bin`, `srk.pub` and the AK name digest) are the inputs for the next two steps. See Key Management for more on keygen.

12.3 MakeCredential and ActivateCredential

12.3.1 Make credential

An attestation server uses `make_credential` to generate the challenge. The secret is 32 bytes of random data that can serve as a symmetric key seed in some attestation schemes.

```
$ ./examples/attestation/make_credential
Using public key from SRK to create the challenge
Demo how to create a credential challenge for remote attestation
Credential will be stored in cred.blob
wolfTPM2_Init: success
Reading 288 bytes from srk.pub
Reading the private part of the key
Public key for encryption loaded
Read AK Name digest success
TPM2_MakeCredential success
Wrote credential blob and secret to cred.blob, 648 bytes
```

The transfer of the PAK and AK public parts between the client and server is not part of this example.

12.3.2 Activate credential

A client uses `activate_credential` to decrypt the challenge. The secret is exposed in plain text and can be sent to the attestation server.

```
$ ./examples/attestation/activate_credential
Using default values
Demo how to create a credential blob for remote attestation
wolfTPM2_Init: success
Credential will be read from cred.blob
Loading SRK: Storage 0x81000200 (282 bytes)
SRK loaded
Reading 508 bytes from keyblob.bin
Reading the private part of the key
AK loaded at 0x80000001
Read credential blob and secret from cred.blob, 648 bytes
TPM2_ActivateCredential success
```

Sending the response containing the secret (in plain text or as a symmetric key seed) back to the server is also outside this example.

12.3.3 Certify

The `certify` example uses `TPM2_Certify` to sign attestation information for another key. This proves that an object with a specific name is loaded in the TPM. A common use is having the restricted IAK sign the attestation information for the IDevID.

`create_primary` can create RSA or ECC IDevID and IAK keys. They are created under the endorsement hierarchy and follow the TCG "TPM 2.0 Keys for Device Identity and Attestation" specification for primary key policies. The IDevID key is used for external, non-restricted signing. The IAK is used for internal attestation. Here the IAK certifies the IDevID.

```
% ./examples/keygen/create_primary -rsa -eh -iak -keep
TPM2.0 Primary Key generation example
        Algorithm: RSA
        Unique: IAK
        Store Handle: 0x00000000
        Use Parameter Encryption: NULL
Creating new RSA primary key...
Create Primary Handle: 0x80000000

% ./examples/keygen/create_primary -rsa -eh -idevid -keep
TPM2.0 Primary Key generation example
        Algorithm: RSA
        Unique: IDEVID
        Store Handle: 0x00000000
        Use Parameter Encryption: NULL
Creating new RSA primary key...
Create Primary Handle: 0x80000001

% ./examples/attestation/certify -rsa -certify=0x80000001 -signer=0x80000000
Certify 0x80000001 with 0x80000000 to generate TPM-signed attestation info
EK Policy Session: Handle 0x3000000
TPM2_Certify complete
Certify Info 172
RSA Signature: 256

% ./examples/management/flush 0x80000001
Preparing to free TPM2.0 Resources
Freeing 80000001 object

% ./examples/management/flush 0x80000000
Preparing to free TPM2.0 Resources
Freeing 80000000 object
```

For ECC, use the same steps and replace `-rsa` with `-ecc`.

12.4 Quote and PCR attestation

The `examples/pcr/` folder has tools for working with Platform Configuration Registers (PCRs) and for generating a TPM 2.0 Quote. Build with `./configure --enable-debug` to see more log output.

Program	Purpose
`./examples/pcr/reset`	Clears the content of a PCR (restrictions apply, see below).

Program	Purpose
`./examples/pcr/extend`	Modifies the content of a PCR with an extend operation.
`./examples/pcr/quote`	Generates a TPM 2.0 Quote with the PCR digest and a TPM signature.
`./examples/pcr/allocate`	Reports which PCR banks the TPM implements and has allocated, and changes the allocation.
`./examples/pcr/demo.sh`	Script that demonstrates the tools above.
`./examples/pcr/demo-quote-zip.sh`	Script that measures a system file and generates a TPM signed proof of that measurement.

12.4.1 PCR basics

A PCR can only be changed by an extend operation. At power-up the TPM resets all PCRs to their default value (all zeros or all ones, depending on the PCR). The same PCR value can only be reached by extending with the same digests in the same order. Extending with A, B, C gives a different result than C, B, A, but each order is reproducible.

`TPM2_Extend` uses a SHA-1 or SHA-256 hash operation to combine the current PCR value with the new digest.

Every PCR can be extended, but only some can be reset at runtime:

- PCR0 to 15 are reset at boot and can be cleared again only by a reboot.
- PCR16 is for debugging. All the tools above use it by default, and it is safe to test with.
- PCR17 to 22 are reserved for Dynamic Root of Trust Measurement (DRTM).

Reset locality follows TCG PC Client: PCR16 and 23 reset at localities 0 to 3, PCR20 to 22 at localities 2 to 4, and PCR17 to 19 at locality 4. Use `-loc=n` to select the locality (see `wolfTPM2_SetLocality`). It applies to the built-in TIS/SPI driver and the fwTPM.

12.4.2 Bank allocation

A TPM keeps a separate set of PCRs per hash algorithm, called a bank. Which banks exist is fixed in silicon, but which are allocated is provisioned with `TPM2_PCR_Allocate` and can be changed. Many parts, including the Infineon SLB9672 and later, allocate only one bank at a time, so moving from SHA-256 to SHA-384 means deallocating SHA-256. SHA-1 is deprecated and is not allocated on current parts.

!!! warning The selection replaces the allocation. Any bank not named in the request is deallocated. The command needs the platform hierarchy, which platform firmware has usually disabled under an OS (the TPM answers `TPM_RC_HIERARCHY` regardless of the authorization supplied). Use `wolfTPM2_AllocatePCRBanks_ex` with a session where platform auth is not the empty password. The change takes effect at the next TPM reset, so power cycle the TPM or restart the simulator, then re-read the banks. Changing banks invalidates every `PolicyPCR` digest, and anything sealed to PCR values can no longer be unsealed.

12.4.3 Quote

`TPM2_Quote` puts the PCR digest in a TCG defined `TPMS_ATTEST` structure together with a TPM signature. The signature comes from an Attestation Identity Key (AIK) that only the TPM can use, which gives assurance about the source of the Quote and the PCR digest.

12.4.4 Tool usage

```
$ ./examples/pcr/reset -?
Incorrect arguments
Expected usage:
./examples/pcr/reset [pcr] [-loc=n]
* pcr is a PCR index between 0-23 (default 16)
* -loc=n switch to TPM locality n (0-4) before reset
    (PCR 17-19 need locality 4; 20-22 need locality 2-4;
     enforced by the fwTPM and by discrete TPMs like the ST33)
Demo usage without parameters, resets PCR16.
```

```
$ ./examples/pcr/extend -?
Incorrect arguments
Expected usage:
./examples/pcr/extend [pcr] [filename]
* pcr is a PCR index between 0-23 (default 16)
* filename points to file(data) to measure
    If wolfTPM is built with --disable-wolfcrypt the file
    must contain SHA256 digest ready for extend operation.
    Otherwise, the extend tool computes the hash using wolfcrypt.
Demo usage without parameters, extends PCR16 with known hash.
```

```
$ ./examples/pcr/quote -?
Incorrect arguments
Expected usage:
./examples/pcr/quote [pcr] [filename]
* pcr is a PCR index between 0-23 (default 16)
* filename for saving the TPMS_ATTEST structure to a file
Demo usage without parameters, generates quote over PCR16 and
saves the output TPMS_ATTEST structure to "quote.blob" file.
```

```
$ ./examples/pcr/allocate -?
Expected usage:
./examples/pcr/allocate [-sha1] [-sha256] [-sha384] [-sha512]
                        [-restore]
* no algorithm flags: report the current allocation and exit
* -shaN: include that bank in the new allocation (repeatable)
* -restore: put the original allocation back before exiting
Demo usage without parameters, reports the PCR banks.

WARNING: the algorithm flags REPLACE the allocation. Banks not
named are deallocated, every PolicyPCR digest changes, and blobs
sealed to PCR values become unsealable. Many TPMs support only
one active bank at a time.

The new allocation takes effect at the next TPM reset, so power
cycle the TPM (or restart the simulator) and re-run to confirm.
```

With no flags, `allocate` reports the banks the TPM has. The list comes from the TPM's own `TPM_CAP_PCRS` response, so a bank this build has no name for prints as its hash algorithm id, and `pcrSelect` is the raw bitmap:

```
$ ./examples/pcr/allocate
PCR banks:
  Bank       Allocated  pcrSelect
  SHA-256    yes        FFFFFF
```

```
  SHA-384     yes         FFFFFF
  SHA-1       no          000000
```

To move to a SHA-384 only allocation and confirm it after a reset:

```
$ ./examples/pcr/allocate -sha384
TPM reported: allocationSuccess YES, maxPCR 24, sizeNeeded 1152, sizeAvailable
 ↪  4608
PCR allocation staged. It takes effect at the next TPM reset
(Startup(CLEAR) after a _TPM_Init) - power cycle the TPM, or
restart the simulator process, then re-run to confirm.

$ ./examples/pcr/allocate
PCR banks:
  Bank        Allocated   pcrSelect
  SHA-256     no          000000
  SHA-384     yes         FFFFFF
  SHA-1       no          000000
```

On a TPM that keeps one bank active, asking for two is rejected with `TPM_RC_PCR`. A TPM that accepts the command but lacks space reports `allocationSuccess = NO`, which the wrapper returns as `BUFFER_E` with `sizeNeeded` greater than `sizeAvailable`. Neither case is a wolfTPM error.

Use `-restore` in scripts so a run leaves the banks as it found them. It replays the exact selection read at startup, bitmaps included, so a partially selected bank comes back partial.

12.4.5 Typical demo output

All PCR examples run without arguments. This is the output of `./examples/pcr/demo.sh`:

```
$ ./examples/pcr/reset
Demo how to reset a PCR (clear the PCR value)
wolfTPM2_Init: success
Trying to reset PCR16...
TPM2_PCR_Reset success
PCR16 digest:
    00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 | ................
    00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 | ................
```

PCR16 is back to all zeros, so later PCR digests are predictable. This is similar to PCR7 right after boot, but PCR16 lets you test without rebooting.

```
$ ./examples/pcr/extend
Demo how to extend data into a PCR (TPM2.0 measurement)
wolfTPM2_Init: success
Hash to be used for measurement:
000102030405060708090A0B0C0D0E0F101112131415161718191A1B1C1D1E1F
TPM2_PCR_Extend success
PCR16 digest:
    bb 22 75 c4 9f 28 ad 52 ca e6 d5 5e 34 a9 74 a5 | ."u..(.R...^4.t.
    8c 7a 3b a2 6f 97 6e 8e cb be 7a 53 69 18 dc 73 | .z;.o.n...zSi..s
```

The new value comes from the old PCR content (all zeros) and the supplied SHA-256 digest. It is always the same if `reset` runs before `extend`. To use your own data, pass the PCR index first (16 is recommended) and a file second.

```
$ ./examples/pcr/quote
Demo of generating signed PCR measurement (TPM2.0 Quote)
```

```
wolfTPM2_Init: success
TPM2_CreatePrimary: 0x80000000 (314 bytes)
wolfTPM2_CreateEK: Endorsement 0x80000000 (314 bytes)
TPM2_CreatePrimary: 0x80000001 (282 bytes)
wolfTPM2_CreateSRK: Storage 0x80000001 (282 bytes)
TPM2_StartAuthSession: sessionHandle 0x3000000
TPM2_Create key: pub 280, priv 212
TPM2_Load Key Handle 0x80000002
wolfTPM2_CreateAndLoadAIK: AIK 0x80000002 (280 bytes)
TPM2_Quote: success
TPM with signature attests (type 0x8018):
    TPM signed 1 count of PCRs
    PCR digest:
    c7 d4 27 2a 57 97 7f 66 1f bd 79 30 0a 1b bf ff | ..'*W..f..y0....
    2e 43 57 cc 44 14 7a 82 11 aa 76 3f 9f 1b 3a 6c | .CW.D.z...v?..:l
    TPM generated signature:
    28 dc da 76 33 35 a5 85 2a 0c 0b e8 25 d0 f8 8d | (..v35..*...%...
    1f ce c3 3b 71 64 ed 54 e6 4d 82 af f3 83 18 8e | ...;qd.T.M......
    (remaining signature bytes omitted)
```

Before the TPM signs the quote, the example creates an Endorsement Key (EK), which acts as the primary key for the others. It then creates a Storage Key (SRK) and, under it, an Attestation Identity Key (AIK) that signs the quote structure.

12.4.6 Measuring a system file (local attestation)

A system administrator wants to confirm that the `zip` tool on a user's system is genuine and unmodified. The administrator resets PCR16, extends it with a hash of the binary, and generates a quote that serves as a reference for later comparison. This is what `./examples/pcr/demo-quote-zip.sh` does.

```
$ ./examples/pcr/reset 16
...
Trying to reset PCR16...
TPM2_PCR_Reset success
...
```

The `extend` tool hashes `/usr/bin/zip` with wolfCrypt (SHA-256) and wolfTPM issues a `TPM2_PCR_Extend` on PCR16:

```
$ ./examples/pcr/extend 16 /usr/bin/zip
...
TPM2_PCR_Extend success
PCR16 digest:
    2b bd 54 ae 08 5b 59 ef 90 42 d5 ca 5d df b5 b5 | +.T..[Y..B..]...
    74 3a 26 76 d4 39 37 eb b0 53 f5 82 67 6f b4 aa | t:&v.97..S..go..
```

Then the administrator creates a quote as proof of the measurement in PCR16:

```
$ ./examples/pcr/quote 16 zip.quote
...
TPM2_Quote: success
TPM with signature attests (type 0x8018):
    TPM signed 1 count of PCRs
...
```

The quote is saved to the binary file `zip.quote`. The `TPMS_ATTEST` structure also holds clock and time information. See the next section for time attestation.

12.4.7 Quote with encrypted qualifying data

The qualifying data supplied for a quote can be protected with parameter encryption. See Examples Overview.

12.5 Signed timestamp (GetTime)

The `signed_timestamp` example creates an Attestation Identity Key (AIK) and uses it to generate a TPM signed timestamp. The timestamp can serve as a protected report of the current system uptime.

```
./examples/timestamp/signed_timestamp
```

The example uses an `authSession` (authorization session) and a `policySession` (policy authorization) to enable the Endorsement Hierarchy, which is needed to create the AIK. The AIK then issues a `TPM2_GetTime` command through the native API, which returns a TPM generated and signed timestamp.

The `clock_set` example increments the TPM2 clock:

```
./examples/timestamp/clock_set [time]
```

12.6 Endorsement key certificates

TPM manufacturers provision endorsement certificates based on a TPM key. The TCG EK Credential Profile defines how they are stored in the TCG NV index range (`TPM_20_TCG_NV_SPACE`). The `get_ek_certs` example enumerates and validates the EK certificates stored there, and creates a primary EK handle that can be used for signing. The `verify_ek_cert` example validates a single EK certificate against the trusted CA list. Some root and intermediate CAs are loaded in `trusted_certs.h`.

```
./examples/endorsement/get_ek_certs
./examples/endorsement/verify_ek_cert
```

12.6.1 Example detail

1. Get the handles in the TCG NV range with `wolfTPM2_GetHandles` and `TPM_20_TCG_NV_SPACE`.
2. Get the certificate size by reading the public NV information with `wolfTPM2_NVReadPublic`.
3. Read the NV data (certificate DER/ASN.1) from the NV index with `wolfTPM2_NVReadAuth`.
4. Get the EK public template for the NV index with `wolfTPM2_GetKeyTemplate_EKIndex` or `wolfTPM2_GetKeyTemplate_EK`.
5. Create the primary endorsement key with the public template and the `TPM_RH_ENDORSEMENT` hierarchy using `wolfTPM2_CreatePrimaryKey`.
6. Parse the ASN.1/DER certificate with `wc_ParseCert` to get the issuer, serial number and other fields.
7. The URI for the CA issuer certificate is in `extAuthInfoCaIssuer`.
8. Import the certificate public key and compare it with the primary EK public unique area.
9. Validate the EK certificate with the wolfSSL Certificate Manager. Load trusted certificates with `wolfSSL_CertManagerLoadCABuffer` and verify with `wolfSSL_CertManagerVerifyBuffer`.
10. Optionally convert to PEM and export with `wc_DerToPem`.

12.6.2 Example certificate chains

Infineon SLB9672. Certificates can be downloaded from these URLs (replace xxx with the 3 digit CA number):

- `https://pki.infineon.com/OptigaRsaMfrCAxxx/OptigaRsaMfrCAxxx.crt`
- `https://pki.infineon.com/OptigaEccMfrCAxxx/OptigaEccMfrCAxxx.crt`

Examples:

- Infineon OPTIGA(TM) RSA Root CA 2, then Infineon OPTIGA(TM) TPM 2.0 RSA CA 059
- Infineon OPTIGA(TM) ECC Root CA 2, then Infineon OPTIGA(TM) TPM 2.0 ECC CA 059

STMicro ST33KTPM:

- STSAFE RSA root CA 02 (http://sw-center.st.com/STSAFE/STSAFERsaRootCA02.crt), then STSAFE-TPM RSA intermediate CA 10 (http://sw-center.st.com/STSAFE/stsafetpmrsaint10.crt)
- STSAFE ECC root CA 02 (http://sw-center.st.com/STSAFE/STSAFEEccRootCA02.crt), then STSAFE-TPM ECC intermediate CA 10 (http://sw-center.st.com/STSAFE/stsafetpmeccint10.crt)

Sample output on an ST33KTPM (certificate hex dumps shortened):

```
$ ./examples/endorsement/verify_ek_cert
Endorsement Certificate Verify
TPM2: Caps 0x30000415, Did 0x0004, Vid 0x104a, Rid 0x 1
TPM2_Startup pass
TPM2_NV_ReadPublic: Sz 14, Idx 0x1c00002, nameAlg 11, Attr 0x62076801, authPol
    0, dataSz 1300, name 34
TPM2_NV_Read: Auth 0x1c00002, Idx 0x1c00002, Offset 0, Size 768
TPM2_NV_Read: Auth 0x1c00002, Idx 0x1c00002, Offset 768, Size 532
EK Data: 1300
        30 82 05 10 30 82 02 f8 a0 03 02 01 02 02 14 58 | 0...0.........X
        ...
wolfTPM2_HashStart: Handle 0x80000002
wolfTPM2_HashUpdate: Handle 0x80000002, DataSz 764
wolfTPM2_HashFinish: Handle 0x80000002, DigestSz 48
Cert Hash: 48
        ...
Issuer Public Exponent 0x10001, Modulus 512
        ...
TPM2_LoadExternal: 0x80000002
EK Certificate Signature: 512
        ...
TPM2_RSA_Encrypt: 512
Decrypted Sig: 512
        ...
Expected Hash: 48
        ...
Sig Hash: 48
        ...
Certificate signature is valid
TPM2_FlushContext: Closed handle 0x80000002
TPM2_FlushContext: Closed handle 0x80000000
```

12.7 Device identity

The TCG publishes a specification for TPM manufacturer guidance on setting up keys for device identity and attestation. wolfTPM supports it with WOLFTPM_MFG_IDENTITY, and it has been tested with the ST33KTPM.

ST33KTPM samples are provisioned with a default master password, enabled with TEST_SAMPLE. To use your own master password, define TPM2_IAK_SAMPLE_MASTER_PASSWORD. The master password is hashed together with the device serial number to produce the authentication for accessing these keys.

The default keys are ECDSA SECP384R1 with SHA2-384. They are stored at the NV indexes defined by `TPM2_IAK_KEY_HANDLE`, `TPM2_IAK_CERT_HANDLE`, `TPM2_IDEVID_KEY_HANDLE` and `TPM2_IDEVID_CERT_HANDLE`.

12.8 See Also

- Examples Overview
- Key Management
- Sealing and NVRAM
- Supported Hardware

COPYRIGHT ©2026 wolfSSL Inc.

13 Sealing and NVRAM

A TPM 2.0 can act as a secure vault. This page covers sealing secrets to a key or to PCR values, storing keys and data in TPM non-volatile memory (NVRAM), and the secure boot root of trust example built on both. All examples are built with the rest of the wolfTPM examples and run from the root of the wolfTPM source tree.

13.1 Seal and unseal overview

TPM 2.0 can protect secrets using a standard Seal/Unseal procedure. A seal can be created using a TPM 2.0 key or against a set of PCR values.

!!! note Secret data sealed in a key is limited to a maximum size of 128 bytes.

The simplest pair of examples is `seal/seal` and `seal/unseal`. Demo usage is shown when they are run without parameters.

13.1.1 Sealing data into a TPM 2.0 key

The `seal` example stores data securely in a newly generated TPM 2.0 key. Only when this key is loaded into the TPM can the secret data be read back.

Example output from sealing and unsealing a secret message:

```
$ ./examples/seal/seal keyblob.bin mySecretMessage
TPM2.0 Simple Seal example
    Key Blob: keyblob.bin
    Use Parameter Encryption: NULL
Loading SRK: Storage 0x81000200 (282 bytes)
Sealing the user secret into a new TPM key
Created new TPM seal key (pub 46, priv 141 bytes)
Wrote 193 bytes to keyblob.bin
Key Public Blob 46
Key Private Blob 141

$ ./examples/keygen/keyload -persistent
TPM2.0 Key load example
    Key Blob: keyblob.bin
    Use Parameter Encryption: NULL
Loading SRK: Storage 0x81000200 (282 bytes)
Reading 193 bytes from keyblob.bin
Reading the private part of the key
Loaded key to 0x80000001
Key was made persistent at 0x81000202

$ ./examples/seal/unseal message.raw
Example how to unseal data using TPM2.0
wolfTPM2_Init: success
Unsealing succeeded
Stored unsealed data to file = message.raw

$ cat message.raw
mySecretMessage
```

After a successful unsealing, the data is stored into a new file. If no filename is provided, the `unseal` tool stores the data in `unseal.bin`.

13.1.2 Sealing to PCRs with a signed policy

To seal a secret to PCRs without the brittleness of a fixed PCR value, an external key signs the expected PCR state. See Secure boot root of trust below, and the `seal_policy_auth` example in the next section.

13.2 Seal examples

The `examples/seal/` directory contains examples of TPM 2.0 seal and unseal operations with different authorization policies, listed from simplest to most flexible.

13.2.1 seal and unseal (password policy)

The simplest seal and unseal, using a password-based authorization policy.

```
./examples/seal/seal keyblob.bin mySecretData
./examples/seal/unseal output.bin keyblob.bin
```

13.2.2 seal_pcr (PCR-only policy)

Seals a secret bound to specific PCR values. The secret can only be unsealed when the PCR values match what was measured at seal time. No password or signing key is required.

Use case: Static Root of Trust, binding secrets to a specific boot state.

```
# Seal and unseal in one step
./examples/seal/seal_pcr -both -pcr=16 -secretstr="MySecret"

# Separate seal/unseal (for example, seal on first boot, unseal on later boots)
./examples/seal/seal_pcr -seal -pcr=16 -secretstr="MySecret"
./examples/seal/seal_pcr -unseal -pcr=16

# With parameter encryption
./examples/seal/seal_pcr -both -pcr=16 -xor -secretstr="MySecret"
./examples/seal/seal_pcr -both -pcr=16 -aes -secretstr="MySecret"

# Custom sealed blob filename
./examples/seal/seal_pcr -seal -sealblob=myblob.bin -secretstr="MySecret"
./examples/seal/seal_pcr -unseal -sealblob=myblob.bin
```

13.2.3 seal_policy_auth (PolicyAuthorize and PCR)

Seals a secret using PolicyAuthorize with a TPM-resident signing key and a PCR policy. The signing key can re-authorize the policy for new PCR values, so secrets can survive authorized changes such as OS updates.

Use case: flexible measured boot with authorized policy updates.

!!! note `authkey.bin` and `sealblob.bin` must be kept together. If the signing key is regenerated, the sealed blob can no longer be unsealed.

```
# ECC signing key (default)
./examples/seal/seal_policy_auth -both -ecc -pcr=16 -secretstr="MySecret"

# RSA signing key
./examples/seal/seal_policy_auth -both -rsa -pcr=16 -secretstr="MySecret"
```

```
# Separate seal/unseal
./examples/seal/seal_policy_auth -seal -ecc -pcr=16 -secretstr="MySecret"
./examples/seal/seal_policy_auth -unseal -ecc -pcr=16

# With parameter encryption
./examples/seal/seal_policy_auth -both -ecc -pcr=16 -xor -secretstr="MySecret"
./examples/seal/seal_policy_auth -both -rsa -pcr=16 -aes -secretstr="MySecret"
```

13.2.4 seal_nv (NV storage and PCR policy)

Stores a secret in TPM NV (non-volatile) memory protected by a PCR policy. Unlike file-based sealed blobs, the secret lives entirely inside the TPM. The program is located at `examples/nvram/seal_nv`.

Use case: secrets that must persist in TPM hardware without external files.

```
# Store, read, delete lifecycle
./examples/nvram/seal_nv -store -pcr=16 -secretstr="MySecret"
./examples/nvram/seal_nv -read -pcr=16
./examples/nvram/seal_nv -delete

# Custom NV index
./examples/nvram/seal_nv -store -pcr=16 -nvindex=0x01800204
 ↪  -secretstr="MySecret"
./examples/nvram/seal_nv -read -pcr=16 -nvindex=0x01800204
./examples/nvram/seal_nv -delete -nvindex=0x01800204
```

13.2.5 Testing

`seal_test.sh` runs 28 tests across all three seal example groups:

```
bash examples/seal/seal_test.sh
```

The tests include positive cases (seal and unseal lifecycle, secret verification), negative cases (PCR mismatch, missing auth key), parameter encryption variants (XOR, AES), and custom filenames and NV indices. Output uses colored PASS, FAIL and SKIP markers with a summary. Verbose output is saved to `seal_test.log`.

Variable	Default	Description
`WOLFCRYPT_ENABLE`	1	wolfCrypt support compiled in
`WOLFCRYPT_DEFAULT`	0	Using the default (reduced) wolfCrypt config
`WOLFCRYPT_ECC`	1	ECC support available
`WOLFCRYPT_RSA`	1	RSA support available

The seal examples are also tested as part of `examples/run_examples.sh`, which runs during `make check`.

13.2.6 Policy comparison

Feature	seal (password)	seal_pcr	seal_policy_auth	seal_nv
Authorization	Password	PCR values	Signing key + PCR	PCR values
Complexity	Low	Low	High	Medium
Survives PCR change	N/A	No	Yes (with auth key)	No
Storage	File	File	File (blob + key)	TPM NV

Feature	seal (password)	seal_pcr	seal_policy_auth	seal_nv
Parameter Encryption	Yes	Yes	Yes	Yes

13.3 Storing keys in NVRAM

These examples show how to use the TPM as a secure vault for keys. There are two programs: one stores a TPM key into the TPM's NVRAM, and the other extracts the key from NVRAM. Both can use parameter encryption to protect against MITM attacks. The NV location is protected with a password authorization that is passed in encrypted form when `-aes` is given on the command line.

Before running the examples, make sure a `keyblob.bin` was generated using the keygen tool. The key can be of any type: RSA, ECC or symmetric. The example stores the private and public part. For a symmetric key the public part is metadata from the TPM.

Typical output for storing and then reading an RSA key with parameter encryption enabled:

```
$ ./examples/nvram/store -aes
Parameter Encryption: Enabled (AES CFB).

TPM2_StartAuthSession: sessionHandle 0x2000000
Reading 840 bytes from keyblob.bin
Storing key at TPM NV index 0x1800202 with password protection

Public part = 616 bytes
NV write of public part succeeded

Private part = 222 bytes
Stored 2-byte size marker before the private part
NV write of private part succeeded


$ ./examples/nvram/read -aes
Parameter Encryption: Enabled (AES CFB).

TPM2_StartAuthSession: sessionHandle 0x2000000
Trying to read 616 bytes of public key part from NV
Successfully read public key part from NV

Trying to read size marker of the private key part from NV
Successfully read size marker from NV

Trying to read 222 bytes of private key part from NV
Successfully read private key part from NV

Extraction of key from NVRAM at index 0x1800202 succeeded
Loading SRK: Storage 0x81000200 (282 bytes)
Trying to load the key extracted from NVRAM
Loaded key to 0x80000001
```

The `read` example tries to load the extracted key if both the public and private part were stored in NVRAM. The `-aes` switch turns on parameter encryption.

The examples can work with partial key material, private or public only, using the `-priv` and `-pub` options. Typical output of storing only the private part of an RSA key pair without parameter encryption:

```
$ ./examples/nvram/store -priv
Parameter Encryption: Not enabled (try -aes or -xor).

Reading 506 bytes from keyblob.bin
Reading the private part of the key
Storing key at TPM NV index 0x1800202 with password protection

Private part = 222 bytes
Stored 2-byte size marker before the private part
NV write of private part succeeded

$ ./examples/nvram/read -priv
Parameter Encryption: Not enabled (try -aes or -xor).

Trying to read size marker of the private key part from NV
Successfully read size marker from NV

Trying to read 222 bytes of private key part from NV
Successfully read private key part from NV

Extraction of key from NVRAM at index 0x1800202 succeeded
```

After a successful key extraction with `read`, the NV index is destroyed. To use `read` again, run `store` again first.

13.3.1 NVRAM programs

All programs are in `examples/nvram/`.

Program	Purpose
`store.c`	Stores a TPM key (private part, public part, or both) into an NV index.
`read.c`	Reads a key back from NV, loads it, and can delete the NV index.
`counter.c`	Creates and increments an NV counter.
`extend.c`	NV extend example showing bus protection with a PolicyOR.
`policy_nv.c`	Stores data in NV and tests a TPM2_PolicyNV based authorization.
`seal_nv.c`	Stores a secret in NV protected by a PCR policy (see Seal examples).

13.4 Secure boot root of trust

The `examples/boot/` directory holds a TPM based root of trust design for secure boot, such as wolfBoot.

13.4.1 Secure boot ROT

The design for storing a public key based root of trust in the TPM:

1. Use AES-CFB parameter encryption for all communication (salted and bound).
2. Derive a password from unique device parameters and use it as the “auth” to load the NV (authenticate).

3. The NV contains a hash of the public key (the hash matches the `.config` setting).
4. wolfBoot still has the public key internally and programs the TPM NV if it is not populated.
5. The NV is locked and created under the platform hierarchy.

Example:

```
$ ./examples/boot/secure_rot
 ↪  -write=../wolfBoot/wolfboot_signing_public_key.der -lock
TPM2: Caps 0x00000000, Did 0x0000, Vid 0x0000, Rid 0x 0
TPM2_Startup pass
TPM2_SelfTest pass
NV Auth (32)
    19 3f bf 0c bb 90 ca a1 40 96 a6 ee 8e fc 7c 3f | .?......@.....|?
    c1 c2 7f 1d c3 e0 a2 5e c7 72 5a a1 94 76 63 53 | .......^.rZ..vcS
Parameter Encryption: Enabled. (AES CFB)

TPM2_StartAuthSession: handle 0x2000000, algorithm AES
TPM2_StartAuthSession: sessionHandle 0x2000000
Storing hash of public key file ../wolfBoot/wolfboot_signing_public_key.der to
 ↪  NV index 0x1400200 with password protection

Public Key Hash (32)
    e3 29 f9 9e 56 93 6e 24 02 34 13 81 0f 7c 73 4d | .)..V.n$.4...|sM
    8f 9d 63 b8 8f 43 39 7b e5 46 93 dd 77 58 77 29 | ..c..C9{.F..wXw)
TPM2_NV_ReadPublic: Sz 14, Idx 0x1400200, nameAlg 11, Attr 0x42072005, authPol
 ↪  0, dataSz 32, name 34
TPM2_NV_DefineSpace: Auth 0x4000000c, Idx 0x1400200, Attribs 0x1107763205, Size
 ↪  32
TPM2_NV_Write: Auth 0x1400200, Idx 0x1400200, Offset 0, Size 32
Wrote 32 bytes to NV 0x1400200
Reading NV 0x1400200 public key hash
TPM2_NV_ReadPublic: Sz 14, Idx 0x1400200, nameAlg 11, Attr 0x62072005, authPol
 ↪  0, dataSz 32, name 34
TPM2_NV_Read: Auth 0x1400200, Idx 0x1400200, Offset 0, Size 32
Read Public Key Hash (32)
    e3 29 f9 9e 56 93 6e 24 02 34 13 81 0f 7c 73 4d | .)..V.n$.4...|sM
    8f 9d 63 b8 8f 43 39 7b e5 46 93 dd 77 58 77 29 | ..c..C9{.F..wXw)
Locking NV index 0x1400200
NV 0x1400200 locked
TPM2_FlushContext: Closed handle 0x2000000
```

13.4.2 Secure boot encryption key storage

To seal a secret to PCRs without the brittleness issue, an external key signs the state of the PCRs.

Tool	Purpose
`./examples/pcr/policy_sign`	Signs a digest for a PCR policy. Outputs the signature and, with `-outpolicy`, the policy authorization digest for the public key.
`./examples/boot/secret_seal`	Seals a secret using the authorization policy digest for the public key. If no secret is provided, a random value is generated and sealed.
`./examples/boot/secret_unseal`	Unseals a secret using the signed authorization policy and the public key.

Create a signed PCR policy:

```
# Extend "aaa" to test PCR 16
echo aaa > aaa.bin
./examples/pcr/reset 16
./examples/pcr/extend 16 aaa.bin

# RSA sign this PCR (result to pcrsig.bin), also creates policyauth.bin from
 ↪  the public key
./examples/pcr/policy_sign -pcr=16 -rsa -key=./certs/example-rsa2048-key.der
 ↪  -out=pcrsig.bin -outpolicy=policyauth.bin
# OR
# ECC sign
./examples/pcr/policy_sign -pcr=16 -ecc -key=./certs/example-ecc256-key.der
 ↪  -out=pcrsig.bin -outpolicy=policyauth.bin
```

Create a sealed secret using that signed policy, based on the public key:

```
# Create a keyed hash sealed object using the policy authorization for the
 ↪  public key
./examples/boot/secret_seal -rsa -policy=policyauth.bin -out=sealblob.bin
./examples/boot/secret_seal -ecc -policy=policyauth.bin -out=sealblob.bin
# OR
# Provide the public key for policy authorization (instead of -policy=)
./examples/boot/secret_seal -rsa
 ↪  -publickey=./certs/example-rsa2048-key-pub.der -out=sealblob.bin
./examples/boot/secret_seal -ecc -publickey=./certs/example-ecc256-key-pub.der
 ↪  -out=sealblob.bin
```

Unseal:

```
# Unseal using the public key
./examples/boot/secret_unseal -pcr=16 -pcrsig=pcrsig.bin -rsa
 ↪  -publickey=./certs/example-rsa2048-key-pub.der -seal=sealblob.bin
./examples/boot/secret_unseal -pcr=16 -pcrsig=pcrsig.bin -ecc
 ↪  -publickey=./certs/example-ecc256-key-pub.der -seal=sealblob.bin
```

13.5 See Also

- TLS and certificates
- Management and GPIO
- Firmware update
- Supported hardware

14 TLS and Certificates

This page covers the wolfTPM examples that build on TPM keys for certificates and secure connections: generating a certificate signing request (CSR), signing test certificates, PKCS #7 signing, and TLS client and server programs that keep the private key inside the TPM.

The PKCS #7 and TLS examples create RSA and ECC keys in NV for testing, using handles defined in `./hal/tpm_io.h`. They require generating CSRs and signing them with a test script, as described below.

14.1 CSR

The `csr` example (`examples/csr/csr.c`) generates a Certificate Signing Request for building a certificate based on a TPM key pair.

```
./examples/csr/csr
```

It creates two files:

- `./certs/tpm-rsa-cert.csr`
- `./certs/tpm-ecc-cert.csr`

Options:

Option	Description
`-cert`	Make a self signed certificate instead of a CSR.
`-signcb`	Use the `wc_SignCert_cb` callback based signing.

Example output (the base64 body is shortened here):

```
./examples/csr/csr
TPM2 CSR Example
Generated/Signed Cert (DER 860, PEM 1236)
-----BEGIN CERTIFICATE REQUEST-----
MIIDWDCCAkACAQIwgZsxCzAJBgNVBAYTAlVTMQ8wDQYDVQQIDAZPcmVnb24xETAP
BgNVBAcMCFBvcnRsYW5kMQ0wCwYDVQQEDARUZXN0MRAwDgYDVQQKDAd3b2xmU1NM
...
l/076ekjTI+7PwzBZIG2F3nOIDUmHwe0lAWdU8h9IoAlM6kS22fh6gZZqQg=
-----END CERTIFICATE REQUEST-----

Generated/Signed Cert (DER 467, PEM 704)
-----BEGIN CERTIFICATE REQUEST-----
MIIBzzCCAXUCAQIwgZsxCzAJBgNVBAYTAlVTMQ8wDQYDVQQIDAZPcmVnb24xETAP
...
6AIgBm+EU6m5SDsk7BYmxTQAhgJFrelwymOa7m16kAXnFuU=
-----END CERTIFICATE REQUEST-----
```

The first request is for the RSA key and the second is for the ECC key.

!!! note The CSR example needs wolfSSL built with certificate generation and certificate request support (`--enable-certgen --enable-certreq`) and the crypto callback.

14.2 Certificate signing

An external script generates test certificates from the TPM generated CSRs. Normally the CSR would be provided to a trusted CA for signing.

```
./certs/certreq.sh
```

The script creates the following X.509 files (also in .pem format):

- `./certs/ca-ecc-cert.der`
- `./certs/ca-rsa-cert.der`
- `./certs/client-rsa-cert.der`
- `./certs/client-ecc-cert.der`
- `./certs/server-rsa-cert.der`
- `./certs/server-ecc-cert.der`

14.3 PKCS #7

The `pkcs7` example (`examples/pkcs7/pkcs7.c`) signs and verifies data with PKCS #7 using a TPM based key. Run these in order:

1. `./examples/csr/csr`
2. `./certs/certreq.sh`
3. `./examples/pkcs7/pkcs7`

The result is displayed on the console.

Option	Description
`-ecc` / `-rsa`	Use an ECC or RSA key (default is RSA).
`-incert=file`	Certificate for the key used. Defaults are `./certs/client-rsa-cert.der` and `./certs/client-ecc-cert.der`.
`-out=file`	Write the generated PKCS #7 file containing the signed data and certificate.

Example output:

```
./examples/pkcs7/pkcs7
TPM2 PKCS7 Example
PKCS7 Signed Container 1625
PKCS7 Container Verified (using TPM)
PKCS7 Container Verified (using software)
```

14.4 TLS examples

The TLS examples use TPM based ECDHE (ECC ephemeral key) support. Compile-time toggles:

Define	Effect
`WOLFTPM2_USE_SW_ECDHE`	Disables use of the TPM for ECC ephemeral key generation and the shared secret. Set with `CFLAGS="-DWOLFTPM2_USE_SW_ECDHE"` or a `#define`.
`WOLFTPM_USE_SYMMETRIC`	Enables symmetric AES, hashing and HMAC support through the TPM for the TLS examples.
`TLS_USE_ECC`	Forces ECC use with wolfSSL when RSA is also enabled.

!!! note To run the TLS server and client on the same machine, build wolfTPM with `WOLFTPM_TIS_LOCK` (`--enable-tislock`). It enables concurrent access protection for the TPM device.

The programs are in `examples/tls/`:

Program	Purpose
`tls_client.c`	TLS client that uses a TPM key and certificate for mutual authentication.
`tls_server.c`	TLS server that uses a TPM key and certificate.
`tls_client_notpm.c`	TLS client that does not use the TPM, for comparison and benchmarking.

14.4.1 Generate the certificates

Generating the client and server certificates requires running:

1. `./examples/keygen/keygen rsa_test_blob.raw -rsa -t`
2. `./examples/keygen/keygen ecc_test_blob.raw -ecc -t`
3. `./examples/csr/csr`
4. `./certs/certreq.sh`
5. Copy the CA files from wolfTPM to the wolfSSL certs directory:

```
cp ./certs/ca-ecc-cert.pem ../wolfssl/certs/tpm-ca-ecc-cert.pem
cp ./certs/ca-rsa-cert.pem ../wolfssl/certs/tpm-ca-rsa-cert.pem
```

The `wolf-ca-rsa-cert.pem` and `wolf-ca-ecc-cert.pem` files come from the wolfSSL example certificates:

```
cp ../wolfssl/certs/ca-cert.pem ./certs/wolf-ca-rsa-cert.pem
cp ../wolfssl/certs/ca-ecc-cert.pem ./certs/wolf-ca-ecc-cert.pem
```

14.4.2 TLS client

The client shows a TPM key and certificate used for TLS mutual authentication (client authentication). The wolfSSL TLS client loads a public key to indicate that mutual authentication is used, and the crypto callback uses the TPM for the private key signing.

By default the client connects to localhost on port 11111. Override with `TLS_HOST` and `TLS_PORT`.

Start a wolfSSL example server:

```
./examples/server/server -b -p 11111 -g -d -i -V
```

To validate the client certificate, use one of these instead:

```
./examples/server/server -b -p 11111 -g -A ./certs/tpm-ca-rsa-cert.pem -i -V
./examples/server/server -b -p 11111 -g -A ./certs/tpm-ca-ecc-cert.pem -i -V
```

Then run the wolfTPM TLS client:

```
./examples/tls/tls_client -rsa
./examples/tls/tls_client -ecc
```

Example output:

```
./examples/tls/tls_client
TPM2 TLS Client Example
Write (29): GET /index.html HTTP/1.0


Read (193): HTTP/1.1 200 OK
Content-Type: text/html
```

```
Connection: close

<html>
<head>
<title>Welcome to wolfSSL!</title>
</head>
<body>
<p>wolfSSL has successfully performed handshake!</p>
</body>
</html>
```

14.4.3 TLS server

The server shows a TPM key and certificate used for a TLS server. It loads the TPM public key, and the crypto callback uses the TPM for the private key signing. By default it listens on port 11111, which can be changed at build time with the TLS_PORT macro.

Run the wolfTPM TLS server:

```
./examples/tls/tls_server -rsa
./examples/tls/tls_server -ecc
```

Then connect with the wolfSSL example client:

```
./examples/client/client -h localhost -p 11111 -g -d
```

To validate the server certificate:

```
./examples/client/client -h localhost -p 11111 -g -A
 ↪  ./certs/tpm-ca-rsa-cert.pem
./examples/client/client -h localhost -p 11111 -g -A
 ↪  ./certs/tpm-ca-ecc-cert.pem
```

You can also browse to `https://localhost:11111`. Browsers show certificate warnings until the test CAs `./certs/ca-rsa-cert.pem` and `./certs/ca-ecc-cert.pem` are loaded into the OS key store. For testing, most browsers allow continuing past the warning.

Example output:

```
./examples/tls/tls_server
TPM2 TLS Server Example
Loading RSA certificate and public key
Read (29): GET /index.html HTTP/1.0


Write (193): HTTP/1.1 200 OK
Content-Type: text/html
Connection: close

<html>
<head>
<title>Welcome to wolfSSL!</title>
</head>
<body>
<p>wolfSSL has successfully performed handshake!</p>
</body>
</html>
```

14.5 See Also

- Sealing and NVRAM
- Management and GPIO
- Firmware update
- Supported hardware

COPYRIGHT ©2026 wolfSSL Inc.

15 Firmware Update

wolfTPM can update the firmware of some TPM 2.0 modules. Enable the examples and library support by configuring with `--enable-firmware`. Supported parts:

- Infineon SLB9672 (SPI) and SLB9673 (I2C) TPM 2.0 modules. Infineon has open sourced its firmware update.
- STMicroelectronics ST33KTPM TPM 2.0 modules. Support covers generation 1 firmware (RSA signed manifest), generation 9 firmware below 512 (ECDSA signed manifest) and generation 9 firmware at 512 and above (LMS signature requirement).

Firmware update programs are in `examples/firmware/`: `ifx_fw_extract.c`, `ifx_fw_update.c`, `st33_fw_update.c` and the policy helper `firmware_policy.c`.

15.1 Infineon firmware (extract and update)

15.1.1 Extracting the firmware

Infineon releases firmware as a .bin file (for example `TPM20_15.23.17664.0_R1.BIN`). The .bin contains a 16 byte GUID header, at least one manifest based on key group, and the firmware. A typical manifest is 3KB and the firmware is 920KB.

The host side tool `ifx_fw_extract` extracts the manifest and firmware data file required for a TPM upgrade.

```
# Build host tool
make

# Help
./ifx_fw_extract --help
Usage:
  ifx_fw_extract <fw-file>
  ifx_fw_extract <fw-file> <keygroup_id> <manifest-file> <data-file>

# Find key groups in .bin
./ifx_fw_extract TPM20_26.13.17770.0_R1.BIN
Reading TPM20_26.13.17770.0_R1.BIN
Found group 00000007

# Extract manifest and firmware data files for key group
./ifx_fw_extract TPM20_26.13.17770.0_R1.BIN 7 TPM20_26.13.17770.0_R1.MANIFEST
 ↪  TPM20_26.13.17770.0_R1.DATA
Reading TPM20_26.13.17770.0_R1.BIN
Found group 00000007
Chosen group found: 00000007
Manifest size is 3224
Data size is 934693
Writing TPM20_26.13.17770.0_R1.MANIFEST
Writing TPM20_26.13.17770.0_R1.DATA
```

15.1.2 Updating the firmware

The `ifx_fw_update` tool uses the manifest (header) and the firmware data file.

The TPM has a vendor capability for getting the key group id. It is populated in `WOLFTPM2_CAPS.keyGroupId` when `wolfTPM2_GetCapabilities` is called. This value should match the `keygroup_id` given to

the extract tool.

```
# Help
./ifx_fw_update --help
Infineon Firmware Update Usage:
        ./ifx_fw_update (get info)
        ./ifx_fw_update --abandon (cancel)
        ./ifx_fw_update --policytest (safe policy auth self-test)
        ./ifx_fw_update [policy opts] <manifest_file> <firmware_file>
        ./ifx_fw_update <manifest_file> <firmware_file> (default auth)
Policy options (caller-supplied authorization):
        --policy    provision+satisfy a PolicyCommandCode
        --policyor  provision+satisfy a PolicyOR (multi-branch)
        --sha256|--sha384|--sha512  policy hash (default SHA-256)

# Run without arguments to display the current firmware information,
# including key group id and operational mode
./ifx_fw_update
Infineon Firmware Update Tool
TPM2: Caps 0x1ae00082, Did 0x001c, Vid 0x15d1, Rid 0x16
TPM2_Startup pass
Mfg IFX (1), Vendor SLB9673, Fw 26.13 (0x456a)
Operational mode: Normal TPM operational mode (0x0)
KeyGroupId 0x7, FwCounter 1254 (255 same)

# Run with manifest and firmware files
./ifx_fw_update TPM20_26.13.17770.0_R1.MANIFEST TPM20_26.13.17770.0_R1.DATA
Infineon Firmware Update Tool
    Manifest File: TPM20_26.13.17770.0_R1.MANIFEST
    Firmware File: TPM20_26.13.17770.0_R1.DATA
TPM2: Caps 0x1ae00082, Did 0x001c, Vid 0x15d1, Rid 0x16
TPM2_Startup pass
Mfg IFX (1), Vendor SLB9673, Fw 26.13 (0x456a)
Operational mode: Normal TPM operational mode (0x0)
KeyGroupId 0x7, FwCounter 1254 (255 same)
TPM2_StartAuthSession: handle 0x3000000, algorithm NULL
TPM2_FlushContext: Closed handle 0x3000000
TPM2_StartAuthSession: handle 0x3000000, algorithm NULL
Firmware manifest chunk 1024 offset (0 / 3224), state 1
Firmware manifest chunk 1024 offset (1024 / 3224), state 2
Firmware manifest chunk 1024 offset (2048 / 3224), state 2
Firmware manifest chunk 152 offset (3072 / 3224), state 0
Firmware data chunk offset 0
Firmware data chunk offset 1024
Firmware data chunk offset 2048
Firmware data chunk offset 3072
...
Firmware data chunk offset 932864
Firmware data chunk offset 933888
Firmware data done
Mfg IFX (1), Vendor , Fw 0.0 (0x0)
Operational mode: After finalize or abandon, reboot required (0x4)
KeyGroupId 0x7, FwCounter 1253 (254 same)
TPM2_Shutdown failed 304: Unknown
```

```
# Reset or power cycle TPM
./ifx_fw_update
Infineon Firmware Update Tool
TPM2: Caps 0x1ae00082, Did 0x001c, Vid 0x15d1, Rid 0x16
TPM2_Startup pass
Mfg IFX (1), Vendor SLB9673, Fw 26.13 (0x456a)
Operational mode: Normal TPM operational mode (0x0)
KeyGroupId 0x7, FwCounter 1253 (254 same)
```

15.2 ST33 firmware update

Build with `--enable-st33 --enable-firmware` to enable this example.

15.2.1 Firmware format auto-detection

ST33KTPM firmware update detects the required format from the TPM firmware version. The manifest (blob0) is a 33 byte fixed header followed by the firmware digest and the signature over it, so its size follows the algorithms that generation signs with:

- Generation 1 (major version 1, for example 1.257 or 1.771): non-LMS format.
  - Manifest size: 321 bytes (SHA-256 digest, RSAPSS-2048 signature).
  - Always non-LMS, no matter how high the minor version goes.
- Generation 9 below 512 (for example 9.257): non-LMS format.
  - Manifest size: 177 bytes (SHA-384 digest, ECDSA P-384 signature).
- Generation 9 at 512 and above (for example 9.512): LMS (Leighton-Micali Signature) format.
  - Manifest size: 2697 bytes (includes the embedded LMS signature).

The LMS requirement is a generation 9 rule, so both `fwVerMajor` and `fwVerMinor` from the TPM capabilities are consulted. The example also confirms its choice against the file itself: everything after blob0 is a chain of `[type][length]` records that ends exactly at the end of the file, and only the correct manifest size lands on the final byte. No manual format selection is needed.

15.2.2 Identifying the part

What `TPM_PT_VENDOR_STRING_1..4` reports on an ST33TPHF2X depends on the firmware. Early 1.x firmware reports binary rather than text, so the `Vendor` field prints empty. Later 1.x firmware reports ASCII, for example `ST33TPHF2XSPI`. The change is present by firmware 1.771 and absent at 1.258. Which firmware introduced it is not known. The firmware major version is the reliable identifier either way, and it tracks the part and interface line:

`fwVerMajor`	Part line	Example firmware image
1	ST33TPHF2X, SPI firmware line	`TPM_ST33TPHF2XSPI_00010301.fi`
2	ST33TPHF2X, I2C firmware line	`TPM_ST33TPHF2XI2C_00020200.fi`
9	ST33KTPM2X	`TPM_ST33KTPM2X_00090200_V1.fi`
10 (`0x000a`)	ST33KTPM2A	`TPM_ST33KTPM2A_000a0200.fi`
11 (`0x000b`)	ST33KTPMQ	(none on hand)

The manifest header carries the same version: a zero byte followed by the firmware version the image upgrades to, in the `TPM_PT_FIRMWARE_VERSION_1` layout (`00 | 00 02 02 00` is 2.512). `st33_fw_update` prints both and refuses an image whose major version does not match the running part.

See Supported hardware for how the ST33 firmware line relates to the firmware version.

15.2.3 Field upgrade command codes

ST33 implements the field upgrade with one of two command code pairs. The wrong one is answered with TPM_RC_COMMAND_CODE (0x143) before the manifest is even parsed:

Pair	Start	Data
Standard TCG	TPM_CC_FieldUpgradeStart (0x0000012F)	TPM_CC_FieldUpgradeData (0x00000141)
ST33KTPM vendor	0x2000030C	0x2000030D

wolfTPM asks the TPM which pair it implements by querying TPM_CAP_COMMANDS for the two start codes. That is authoritative and needs no table of parts. ST's own reference tool instead infers the pair from version numbers (standard codes when the running firmware minor version is below 256, or when the image targets firmware generation 2). wolfTPM falls back to that same rule, `wolfTPM2_ST33_FwUpgradeCommands()`, only when the TPM will not answer (which is the case once it has entered firmware upgrade mode) or when it lists both pairs.

The probe matters: an ST33KTPMQ at firmware 11.1 implements only the vendor pair even though its minor version is below 256, so the version rule alone would pick the wrong codes for it.

Nothing needs to be selected by hand. When a caller-supplied policy is used, the PolicyCommandCode is bound to whichever start code will actually be sent, and `st33_fw_update` reports whether the codes came from the TPM or were inferred.

Running `st33_fw_update` with no firmware file also reports which of the four codes the attached part implements, read from TPM_CAP_COMMANDS. This is read-only and is the quickest way to diagnose a TPM_RC_COMMAND_CODE on a new part:

```
./st33_fw_update
...
Field upgrade command set:
    0x0000012f FieldUpgradeStart       (standard): implemented
    0x00000141 FieldUpgradeData        (standard): implemented
    0x2000030c FieldUpgradeStartVendor (ST33KTPM): not implemented
    0x2000030d FieldUpgradeDataVendor  (ST33KTPM): not implemented
```

A firmware major version outside the known families is treated as unknown rather than guessed at. No manifest size is asserted for it, the tool reports that the size is taken from the image, and the block-chain check in `st33_detect_blob0` establishes the real size from the file itself. That way a part from a newer line, such as an ST33KTPMQ, is not refused an image on the strength of a rule that does not apply to it.

15.2.4 Updating the firmware

The `st33_fw_update` tool detects the firmware format automatically.

```
# Help
./st33_fw_update --help
ST33 Firmware Update Usage:
    ./st33_fw_update (get info)
    ./st33_fw_update --abandon (cancel)
    ./st33_fw_update --policytest (safe policy auth self-test)
    ./st33_fw_update [policy opts] <firmware.fi>
    ./st33_fw_update <firmware.fi> (default password auth)
Policy options (caller-supplied authorization):
    --policy    provision+satisfy a PolicyCommandCode
```

```
    --policyor  provision+satisfy a PolicyOR (multi-branch)
    --sha256|--sha384|--sha512  policy hash (default SHA-256)

Firmware format is auto-detected from TPM firmware version and the file:
      - Generation 1 (e.g. 1.771): Non-LMS format (321 byte manifest)
      - Generation 9 below 512: Non-LMS format (177 byte manifest)
      - Generation 9 at 512 and above: LMS format (2697 byte manifest)

# Run without arguments to display the current firmware information.
# This capture is an ST33TPHF2XSPI, which implements only the vendor codes.
./st33_fw_update
ST33 Firmware Update Tool
TPM2: Caps 0x30000415, Did 0x0000, Vid 0x104a, Rid 0x4e
TPM2_Startup pass
Mfg STM  (2), Vendor , Fw 1.258 (0x0)
Firmware version details: Major=1, Minor=258, Vendor=0x0
Part line: ST33TPHF2X (SPI firmware line)
Firmware generation: 1
Firmware update: Non-LMS format required (321 byte manifest)
Field upgrade command set:
    0x0000012f FieldUpgradeStart       (standard): not implemented
    0x00000141 FieldUpgradeData        (standard): not implemented
    0x2000030c FieldUpgradeStartVendor (ST33KTPM): implemented
    0x2000030d FieldUpgradeDataVendor  (ST33KTPM): implemented

# Run with firmware file (format auto-detected from TPM version)
./st33_fw_update TPM_ST33KTPM2X_00090200_V1.fi
ST33 Firmware Update Tool
    Firmware File: TPM_ST33KTPM2X_00090200_V1.fi
TPM2: Caps 0x30000415, Did 0x0003, Vid 0x104a, Rid 0x 1
TPM2_Startup pass
Mfg STM (2), Vendor ST33KTPM2X, Fw 9.257 (0x0)
Firmware version details: Major=9, Minor=257, Vendor=0x0
Part line: ST33KTPM2X
Firmware generation: 9 below 512
Firmware update: Non-LMS format required (177 byte manifest)
    Format: Non-LMS (blob0 177 bytes, verified against the block chain)
Firmware Update:
    Total file size: 364290 bytes
    Manifest (blob0): 177 bytes
    Firmware data: 364113 bytes
    Image targets firmware: 9.512 (ST33KTPM2X)
    Command codes: start 0x2000030c, data 0x2000030d
...
Firmware update completed successfully.
Please reset or power cycle the TPM.
```

!!! note Firmware files cannot be made public and must be obtained separately from STMicroelectronics.

15.2.5 LMS firmware

For a generation 9 TPM with firmware at 512 and above, the LMS format is required:

```
./st33_fw_update ST33KTPM2X_FAC_00090200_V2.fi
ST33 Firmware Update Tool
    Firmware File: ST33KTPM2X_FAC_00090200_V2.fi
TPM2: Caps 0x30000415, Did 0x0003, Vid 0x104a, Rid 0x 3
TPM2_Startup pass
Mfg STM (2), Vendor ST33KTPM2X, Fw 9.512 (0x0)
Firmware version details: Major=9, Minor=512, Vendor=0x0
Firmware generation: 9 at 512 and above
Firmware update: LMS format required (2697 byte manifest)
    Format: LMS (blob0 2697 bytes, verified against the block chain)
Firmware Update:
    Total file size: 360092 bytes
    Manifest (blob0): 2697 bytes
    Firmware data: 357395 bytes
...
Firmware update completed successfully.
Please reset or power cycle the TPM.
```

15.2.6 Cancel an update

```
./st33_fw_update --abandon
ST33 Firmware Update Tool
TPM2: Caps 0x30000415, Did 0x0003, Vid 0x104a, Rid 0x 1
TPM2_Startup pass
Mfg STM (2), Vendor ST33KTPM2X, Fw 9.257 (0x0)
Firmware version details: Major=9, Minor=257, Vendor=0x0
Firmware generation: 9 below 512
Firmware update: Non-LMS format required (177 byte manifest)
Firmware Update Abandon:
Success: Please reset or power cycle TPM
```

The same tool is also described in the main README as a short block: `./examples/firmware/st33_fw_update` displays firmware information, `--abandon` cancels any in-progress update, and passing a `<firmware.fi>` file performs the update with the format auto-detected from the TPM firmware version.

15.3 Policy-based authorization (advanced)

By default wolfTPM manages the platform hierarchy authorization for the firmware update start command internally. On Infineon it installs and satisfies a `PolicyCommandCode(TPM_CC_FieldUpgradeStartVendor)` policy on the platform primary policy. On ST33 it uses password authorization (`TPM_RS_PW`) with an empty platform password. This assumes the platform hierarchy has default (empty) authorization.

Deployments that gate firmware upgrade behind their own platform policy (for example a signed-policy check, a PCR state, or a multi-branch `PolicyOR`) can supply an already-satisfied authorization session using `wolfTPM2_FirmwareUpgradeHash_ex()`. When a session is supplied:

- **Infineon**: the library does not overwrite your platform primary policy. You provision the platform `authPolicy` yourself (through `TPM2_SetPrimaryPolicy` with `authHandle = TPM_RH_PLATFORM`, using SHA2-256 or SHA2-512) and pass a session that satisfies it. This applies to the library. The `--policy` and `--policyor` example modes are themselves such a caller, and their helper (`examples/firmware/firmware_policy.c`) does overwrite the platform `authPolicy` with a digest it generates. Do not run those modes on a system whose platform hierarchy already carries a policy you need.
- **ST33**: the supplied session replaces the default `TPM_RS_PW` password authorization.

Supported session contract: the vendor FieldUpgradeStart command is sent with an authorization area carrying only the session handle, with an empty nonceCaller, zero session attributes and an empty HMAC. The supplied session must therefore be an unsalted, unbound TPM_SE_POLICY session with no auth value and no parameter encryption. Policies satisfied with wolfTPM2_PolicyAuthValue() or wolfTPM2_PolicyPassword() are not supported, because the session HMAC they require is not serialized on this path. Such a session is rejected with BAD_FUNC_ARG before anything is sent to the TPM. PolicyPCR, PolicySigned, PolicySecret, PolicyAuthorize, PolicyCommandCode and PolicyOR branches are all fine.

Both SHA2-256 (non-PQC) and SHA2-512 (PQC) policy digests are supported, because the session hash is chosen with wolfTPM2_StartSession_ex(..., authHash) and wolfTPM2_PolicyOR() carries per-branch digest sizes.

Example: satisfy a multi-branch PolicyOR (up to 8 branches, SHA2-512 shown) and start the upgrade under it:

```
WOLFTPM2_SESSION session;
TPML_DIGEST orList;
uint8_t manifest_hash[TPM_SHA512_DIGEST_SIZE];
int rc;

/* zero both structs: orList must not carry uninitialized branch sizes */
XMEMSET(&session, 0, sizeof(session));
XMEMSET(&orList, 0, sizeof(orList));

/* start a policy session using the desired policy hash (SHA2-512 for PQC) */
rc = wolfTPM2_StartSession_ex(&dev, &session, NULL, NULL,
    TPM_SE_POLICY, TPM_ALG_NULL, TPM_ALG_SHA512);
if (rc != TPM_RC_SUCCESS) goto cleanup;

/* Satisfy one branch (PCR, PolicySigned, PolicyAuthorize, PolicyCommandCode,
 * ...), then OR against the full branch list the platform authPolicy encodes.
 * Set count and each digests[i].size/buffer for every branch you populate.
 * PolicyOR requires at least 2 branches. */
orList.count = 2;
/* orList.digests[0].size = ...; XMEMCPY(orList.digests[0].buffer, ...); */
/* orList.digests[1].size = ...; XMEMCPY(orList.digests[1].buffer, ...); */
rc = wolfTPM2_PolicyOR(&dev, &session, &orList);
if (rc != TPM_RC_SUCCESS) goto cleanup;

/* hash the manifest with the matching algorithm, then start the upgrade under
 * the caller-satisfied session (NULL would use the library-default auth) */
rc = wc_Sha512Hash(manifest, manifest_sz, manifest_hash);
if (rc != 0) goto cleanup;
rc = wolfTPM2_FirmwareUpgradeHash_ex(&dev, TPM_ALG_SHA512,
    manifest_hash, (uint32_t)sizeof(manifest_hash),
    manifest, manifest_sz, fwDataCb, fwCbCtx, &session);

cleanup:
/* On a successful FieldUpgradeStart the TPM consumes the session and the
 * library sets session.handle.hndl to TPM_RH_NULL (0x40000007). It is NOT
 * zeroed, so do not test for == 0 to detect consumption. Calling
 * wolfTPM2_UnloadHandle is always safe: it is a no-op on TPM_RH_NULL, so this
 * only releases a session that is still loaded. */
if (session.handle.hndl != 0)
    wolfTPM2_UnloadHandle(&dev, &session.handle);
```

Passing NULL for the final startSession argument makes wolfTPM2_FirmwareUpgradeHash_ex() behave exactly like wolfTPM2_FirmwareUpgradeHash() (library-managed authorization), so existing code is unaffected.

15.3.1 Destructive: provisioning replaces any existing platform policy

!!! warning --policy and --policyor call TPM2_SetPrimaryPolicy on the platform hierarchy with a digest the example generates. TPM 2.0 provides no way to read a hierarchy's authPolicy back: there is no read command, and TPMA_PERMANENT reports only authValue state. The example therefore cannot detect an existing policy, cannot preserve it, and cannot restore it. Cleanup removes the policy rather than restoring whatever was there before.

If your platform hierarchy is gated by a policy you need to keep, do not run these modes. The example prints this warning at provisioning time. --policytest is unaffected: it is non-destructive and never calls TPM2_SetPrimaryPolicy.

The modes also require the normal operational mode. In recovery and finalize modes the library skips FieldUpgradeStart entirely, so a caller-supplied session would never be used, and the example refuses rather than installing a policy nothing will exercise. On ST33, if the TPM is already in firmware-upgrade mode the policy flags are likewise rejected, since the start command has already run.

15.3.2 Rollback of the example-provisioned policy

The example --policy and --policyor modes provision the platform hierarchy authPolicy with TPM2_SetPrimaryPolicy before the upgrade. On failure the example clears it again so a later default-auth run is not locked out. On success the required TPM reset clears it.

- Rollback normally uses platform password authorization. Per TPM 2.0 Part 1 Sec. 19.7 a hierarchy is authorized by either its authValue or its authPolicy, so installing an authPolicy does not disable the password path. With the default empty platformAuth the clear always succeeds.
- --policyor also provisions a PolicyCommandCode(TPM_CC_SetPrimaryPolicy) branch alongside the firmware-start branch, so the policy can authorize its own removal. If the password path fails (a deployment that set a non-default platformAuth), the example retries the clear under that branch.
- --policy provisions a single PolicyCommandCode(FieldUpgradeStart) branch and therefore has no policy-based rollback path. It relies entirely on platformAuth still being usable.
- Rollback is attempted only when the example actually installed the policy, so an early failure (a missing firmware file, for example) never clears a policy the deployment provisioned itself.
- A failed rollback is reported explicitly and becomes the exit status. If a run is interrupted before cleanup, or the clear fails, the platform hierarchy still requires the policy until the TPM is reset or power cycled.

15.4 See Also

- Supported hardware
- Sealing and NVRAM
- TLS and certificates
- Management and GPIO

16 Management and GPIO

This page covers the small TPM management utilities in `examples/management/` and the GPIO control examples in `examples/gpio/`.

16.1 Management utilities

Program	Purpose
`da_check.c`	Dictionary attack (DA) lockout check. Exercises a DA-protected key and a noDA key, enters lockout with repeated bad authorization, and recovers with a lockout reset.
`flush.c`	Flushes transient and persistent handles. Run with a handle (for example `0x80000000`) to free that object; with no parameters it flushes common transient objects (transient keys, policy sessions and HMAC sessions).
`tpmclear.c`	Runs `TPM2_Clear` to clear a hierarchy.

```
./examples/management/da_check
./examples/management/flush [handle]
./examples/management/tpmclear
```

!!! warning `tpmclear` clears the TPM. Keys and data held under the cleared hierarchy are lost.

16.2 GPIO control

Some TPM 2.0 modules have extra I/O functions and additional GPIO that a developer can use. The extra GPIO can signal other subsystems about security events or system states.

!!! note The GPIO control examples support only ST33 and NPCT75x TPM 2.0 modules.

There are three programs in `examples/gpio/`:

Program	Purpose
`gpio_config.c`	Configures a GPIO.
`gpio_set.c`	Sets a configured GPIO high or low.
`gpio_read.c`	Reads the level of a configured GPIO.

Every example has a help option `-h`. Run `gpio_config -h` to see the GPIO modes. Demo usage runs when no parameters are supplied. Choose options carefully, because GPIO interact with the physical world.

16.2.1 GPIO config (ST33)

ST33 supports 6 modes. Help output from `gpio_config`:

```
$ ./examples/gpio/gpio_config -h
Expected usage:
./examples/gpio/gpio_config [num] [mode]
* num is a GPIO number between 0-3 (default 0)
* mode is a number selecting the GPIO mode between 0-6 (default 3):
```

```
    0. standard - reset to the GPIO's default mode
    1. floating - input in floating configuration.
    2. pullup   - input with pull up enabled
    3. pulldown - input with pull down enabled
    4. opendrain - output in open drain configuration
    5. pushpull  - output in push pull configuration
    6. unconfigure - delete the NV index for the selected GPIO
Example usage, without parameters, configures GPIO0 as input with a pull down.
```

Configure a GPIO as an output:

```
$ ./examples/gpio/gpio_config 0 5
GPIO num is: 0
GPIO mode is: 5
Example how to use extra GPIO on a TPM 2.0 modules
Trying to configure GPIO0...
TPM2_GPIO_Config success
NV Index for GPIO access created
```

Configure a GPIO as an input with a pull down (mode 3):

```
$ ./examples/gpio/gpio_config 0 3
GPIO num is: 0
GPIO mode is: 3
Demo how to use extra GPIO on a TPM 2.0 modules
Trying to configure GPIO0...
TPM2_GPIO_Config success
NV Index for GPIO access created
```

16.2.2 GPIO config (NPCT75xx)

NPCT75x supports 3 output modes and no input modes. Help output from gpio_config:

```
$ ./examples/gpio/gpio_config -h
Expected usage:
./examples/gpio/gpio_config [num] [mode]
* num is a GPIO number between 3 and 4 (default 3)
* mode is either push-pull, open-drain or open-drain with pull-up
    1. pushpull  - output in push pull configuration
    2. opendrain - output in open drain configuration
    3. pullup - output in open drain with pull-up enabled
    4. unconfig - delete NV index for GPIO access
Example usage, without parameters, configures GPIO3 as push-pull output.
```

NPCT75x GPIO numbering starts from GPIO3, while ST33 starts from GPIO0.

```
$ ./examples/gpio/gpio_config 4 1
Example for GPIO configuration of a NPTC7xx TPM 2.0 module
GPIO number: 4
GPIO mode: 1
Successfully read the current configuration
Successfully wrote new configuration
NV Index for GPIO access created
```

16.2.3 GPIO usage

Switching a GPIO configuration works as follows:

- For ST33, `gpio_config` deletes the existing NV index, so a new GPIO configuration can be chosen.
- For NPCT75xx, `gpio_config` can reconfigure any GPIO without deleting the created NV index.

Once configured, set and read the GPIO:

```
$ ./examples/gpio/gpio_set 0 -high
GPIO0 set to high level

$ ./examples/gpio/gpio_set 0 -low
GPIO0 set to low level

$ ./examples/gpio/gpio_read 0
GPIO0 is Low
```

16.3 See Also

- Sealing and NVRAM
- TLS and certificates
- Firmware update
- Supported hardware

17 Post-Quantum Cryptography

wolfTPM adds support for the post-quantum algorithms introduced in TCG TPM 2.0 Library Specification v1.85. The client library marshals the new v1.85 commands to a TPM. When the target is the in-tree firmware TPM, that server performs the algorithms with wolfCrypt's FIPS 203 (ML-KEM) and FIPS 204 (ML-DSA) modules; when the target is a hardware TPM such as SEALSQ QVault, the device performs them on-chip. This page covers the supported algorithms, how to build with them, and the PQC examples shipped in `examples/pqc`.

17.1 Overview

Supported algorithms:

Algorithm	Standard	Parameter sets
ML-DSA (signing)	FIPS 204	ML-DSA-44 / 65 / 87
HashML-DSA (pre-hash signing)	FIPS 204	ML-DSA-44 / 65 / 87 with caller hash
ML-KEM (key encapsulation)	FIPS 203	ML-KEM-512 / 768 / 1024

wolfTPM supports the SEALSQ QVault TPM, which SEALSQ positions as the first TPM 2.0 device with these v1.85 PQC algorithms in silicon. SEALSQ describes QVault TPM-185 engineering samples as available, so check with SEALSQ for current production and certification status. The same PQC API also runs against the in-tree fwTPM server, which is useful for CI or when no hardware is present. For measured ML-DSA and ML-KEM performance on QVault TPM silicon, see the Benchmarks section at the end of this page.

17.2 Building

17.2.1 wolfSSL

wolfSSL provides ML-DSA and ML-KEM in wolfCrypt:

```
./configure --enable-wolftpm --enable-pkcallbacks --enable-keygen \
            --enable-mldsa --enable-mlkem \
            --enable-harden CFLAGS="-DWC_RSA_NO_PADDING"
make
sudo make install
```

For the PQC TLS 1.3 demo described below, also add `--enable-tls-mlkem-standalone`. The standalone option is required for the standalone `ML_KEM_*` TLS groups; without it wolfSSL only offers the hybrid groups and `wolfSSL_UseKeyShare` rejects the client default. The `gen_pqc_certs` tool needs certificate generation, which `--enable-wolftpm` already enables, and `--enable-wolftpm` also provides the crypto callback and private-key-id support the TLS server uses. Keep `--enable-pkcallbacks`, which is a separate wolfSSL option. The demo needs wolfSSL 5.9.4-stable or later.

17.2.2 fwTPM (software TPM)

```
./configure --enable-fwtpm --enable-swtpm --enable-pqc
make
```

`--enable-swtpm` builds `fwtpm_server` with the mssim socket transport on `127.0.0.1:2321` that the examples below connect to. On Linux x86_64 and AArch64 the socket transport is already the default, so the flag is what makes the socket server reliable on every platform. The fwTPM server

uses the full v1.85 command set, so configure promotes `--enable-pqc` to `--enable-v185`. If you omit both PQC flags and wolfCrypt has ML-DSA and ML-KEM, configure auto-enables v1.85. Pass `--disable-pqc` to opt out explicitly.

17.2.3 Hardware TPM: SEALSQ QVault

SEALSQ QVault is the hardware TPM currently supported for v1.85 PQC:

```
./configure --enable-sealsq --enable-pqc
make
```

On Linux, add `--enable-devtpm` to use the kernel TPM driver. For examples that pass transient handles between processes, also add `CFLAGS='-DTPM2_LINUX_DEV="/dev/tpm0"'`. The default `/dev/tpmrm0` virtualizes and discards those handles when a process closes the device. Keep the inner double quotes literal inside the single-quoted `CFLAGS` value.

`--enable-pqc` builds the lean ML-DSA and ML-KEM subset (`WOLFTPM_PQC`) for hardware. Use `--enable-v185` (`WOLFTPM_V185`) for the full v1.85 command set.

Non-SHA-1 TPM examples include:

```
./examples/pqc/pqc_ctrl --caps --algs
./examples/pqc/pqc_ctrl --mldsa=65 --mlkem=768
./examples/wrap/hash "wolfTPM" -sha256
```

17.2.4 Trimming the PQC footprint

To compile only the operations you call (a smaller binary, with the unused code paths removed), mirror the wolfSSL flags:

```
# ML-DSA verify-only + ML-KEM encapsulate-only (no sign, no decapsulate)
./configure --enable-pqc --enable-mldsa=verify-only --enable-mlkem=enc
```

Flag	Values	Drops
`--enable-mldsa`	`all` (default) / `sign-only` / `verify-only` / `no`	the unselected ML-DSA operation
`--enable-mlkem`	`all` (default) / `enc` / `dec` / `no`	the unselected ML-KEM operation
`--disable-hash-mldsa`	none	pre-hash ML-DSA key support

These map to `WOLFTPM_NO_MLDSA_SIGN`, `WOLFTPM_NO_MLKEM_DECAP`, and similar defines, which embedded integrators can also pass directly via `CFLAGS` without autotools. Existing `--enable-v185` builds are unaffected (every operation defaults on). Disabling both algorithms (`--enable-mldsa=no --enable-mlkem=no`) is a configure error. Use `--disable-pqc` to build without any post-quantum support. Trimming removes code paths and shrinks the binary; it does not reduce the public TPM2B buffer maximums, which stay sized for the largest v1.85 parameter set.

The same flags also trim the fwTPM server: `--enable-fwtpm --enable-mldsa=verify-only` compiles out the ML-DSA sign command handlers, dispatch entries, and crypto. ML-KEM is controlled separately and stays at its default of `all`, so to build a server whose PQC surface is only ML-DSA verify, also pass `--enable-mlkem=no`. fwTPM always builds the full v1.85 spec surface, so the trims apply on top of `WOLFTPM_V185`.

!!! note Trimming does not make the examples allocation-free; they still allocate signature and ciphertext buffers with `XMALLOC`. fwTPM has a separate `WOLFTPM2_NO_HEAP` option that moves all buffers to the stack, at the cost of much larger stack use. See fwTPM Building for details.

17.2.5 CMake

The CMake build takes the same choices as cache variables, and also accepts ON and OFF for yes and no:

```
cmake .. -DWITH_WOLFSSL=/prefix/to/wolfssl -DWOLFTPM_FWTPM=yes \
         -DWOLFTPM_INTERFACE=SWTPM -DWOLFTPM_V185=yes
```

Variable	Values	Configure flag
WOLFTPM_V185	detect (default) / yes / no	--enable-v185
WOLFTPM_PQC	detect (default) / yes / no	--enable-pqc
WOLFTPM_MLDSA	all (default) / sign-only / verify-only / no	--enable-mldsa
WOLFTPM_MLKEM	all (default) / enc / dec / no	--enable-mlkem
WOLFTPM_HASH_MLDSA	yes (default) / no	--disable-hash-mldsa

As with configure, a fwTPM build enables v1.85 when wolfCrypt has ML-DSA and ML-KEM, and promotes WOLFTPM_PQC=yes to v1.85. With -DWITH_WOLFSSL_TREE the check reads the tree's WOLFSSL_MLDSA and WOLFSSL_MLKEM options instead of compiling against installed headers. wolfSSL's own CMake build leaves ML-DSA off unless it gets -DWOLFSSL_MLDSA=yes.

17.3 Running the examples

```
make check
```

With the fwTPM build above, make check runs the software-TPM suite, including PQC coverage:

- tests/fwtpm_unit.test: 30+ in-process PQC handler tests
- tests/fwtpm_check.sh: starts the server, then runs tests/unit.test (PQC wrapper tests over the mssim socket such as ML-DSA Sign/Verify Sequence, ML-KEM Encap/Decap, and EncryptSecret ML-KEM) and examples/run_examples.sh, and runs the tpm2-tools suite when the socket transport is enabled and tpm2-tools is installed
- tests/fwtpm_da_retry.sh: dictionary attack retry check (needs a build with -DFWTPM_DA_USED_RETRY)

make check does not run tests/pqc_mssim_e2e.sh. Run that script directly for a fast, PQC-focused end-to-end check:

```
./tests/pqc_mssim_e2e.sh
```

With an fwTPM build, start fwtpm_server once in a separate terminal and leave it running while you run the individual examples below (including the TLS demo). It listens on 127.0.0.1:2321. A SEALSQ build uses its configured hardware transport instead. The --clear flag below deletes the fwTPM NV state file before starting, which gives the examples a clean TPM; use it only on a disposable test instance, and omit it to keep an existing fwTPM's persistent state.

```
# separate terminal; leave this running
./src/fwtpm/fwtpm_server --clear
```

For the fwTPM server's PQC internals (the eight v1.85 commands, primary-key derivation, buffer constants, and spec-interpretation decisions), see docs/FWTPM.md.

17.4 Examples

17.4.1 pqc_ctrl

pqc_ctrl is a single CLI to drive and validate a PQC TPM (SEALSQ QVault TPM, or the fwTPM). Each command runs an operation against the TPM. Every key operation flushes the transient object

table first, so a TPM with a small object memory (such as the SEALSQ QVault TPM) does not hit TPM_RC_OBJECT_MEMORY when commands are chained.

```
./examples/pqc/pqc_ctrl                 # --all (default)
./examples/pqc/pqc_ctrl --caps --algs   # identify + list supported algorithms
./examples/pqc/pqc_ctrl --mldsa=87      # ML-DSA-87 sign/verify
./examples/pqc/pqc_ctrl --mlkem=1024    # ML-KEM-1024 encap/decap
./examples/pqc/pqc_ctrl --selftest --getrandom=32 --pcrread=0
```

Command	Description
`--caps`	Manufacturer, vendor string, firmware, FIPS mode
`--algs`	List the algorithms the TPM reports as supported
`--selftest`	TPM2_SelfTest
`--getrandom[=N]`	N random bytes (default 16)
`--pcrread[=idx]`	Read a PCR (SHA-256 bank, falling back to SHA-384)
`--pcrextend=idx`	Extend a PCR with a test digest (explicit index required)
`--flush`	Flush all loaded transient objects between operations
`--clear`	TPM2_Clear: installs a new Storage primary seed, deletes Storage and Endorsement objects and non-platform NV, and resets owner, endorsement, and lockout auth
`--mldsa[=44/65/87]`	Pure ML-DSA sign/verify (default 65)
`--hash-mldsa[=44/65/87]`	HashML-DSA sign/verify (SHA-256/384/512 pre-hash for 44/65/87)
`--mlkem[=512/768/1024]`	ML-KEM encapsulate/decapsulate
`--all`	caps + algs + selftest + getrandom + pcrread + every PQC set

Commands run left to right, so they can be chained. The functional `pqc_ctrl` builds only from an untrimmed PQC configuration: the wrapper plus both ML-DSA operations, both ML-KEM operations, and HashML-DSA. Build with `--enable-v185` (or `--enable-pqc`); a trimmed build such as `--enable-mldsa=verify-only` compiles out the CLI. Point it at the SEALSQ part with `--enable-sealsq`, or at the fwTPM with `--enable-fwtpm --enable-swtpm`.

Pre-production TPM firmware that supports only SHA-256 with HashML-DSA-65 or HashML-DSA-87 can be used by defining `WOLFSSL_MLDSA_ALLOW_WEAK_PREHASH` when building both wolfSSL and wolfTPM. The examples then select SHA-256 for every parameter set. This compatibility mode is disabled by default and does not comply with the FIPS 204 section 5.4 pre-hash strength requirement.

`pqc_ctrl.sh` runs the whole command set as a pass/fail suite (it mirrors `examples/spdm/spdm_test.sh`) when the device supports all parameter sets. The persistent state changes (the PCR 16 extend and TPM2_Clear) are opt-in via `PQC_CTRL_CLEAR=1`. The default run still flushes loaded transient objects, and every key operation flushes the transient object table, so do not run it alongside a workload whose transient TPM handles must stay valid:

```
./examples/pqc/pqc_ctrl.sh
PQC_CTRL_CLEAR=1 ./examples/pqc/pqc_ctrl.sh   # also extend PCR 16 and run
 ↪  TPM2_Clear
```

!!! warning PQC_CTRL_CLEAR=1 does two things, not one. It first extends PCR 16, which cannot be undone without resetting the PCR, and then runs TPM2_Clear. TPM2_Clear installs a new Storage primary seed, so every key blob and sealed object protected by the old seed is permanently unusable afterward, and a file backup does not restore them. Use it only on a disposable test TPM.

17.4.2 pqc_mssim_e2e

End-to-end client test over the mssim socket. It runs four checks in sequence:

1. ML-KEM-768 CreatePrimary, Encapsulate, and Decapsulate. It asserts the ciphertext is 1088 bytes and the two shared secrets are byte-identical.
2. HashML-DSA-44 (SHA-256) CreatePrimary, SignDigest, and VerifyDigestSignature. It asserts the signature is 2420 bytes and the validation ticket tag is TPM_ST_DIGEST_VERIFIED.
3. An ML-KEM MakeCredential and ActivateCredential round-trip.
4. An ML-DSA Quote.

```
./examples/pqc/pqc_mssim_e2e
```

17.4.3 mlkem_encap

ML-KEM encapsulation round-trip. It creates a primary ML-KEM key, runs Encapsulate, then runs Decapsulate on the produced ciphertext and confirms the shared secrets match.

```
./examples/pqc/mlkem_encap                # default: ML-KEM-768
./examples/pqc/mlkem_encap -mlkem=512
./examples/pqc/mlkem_encap -mlkem=1024
```

17.4.4 mldsa_sign

Pure ML-DSA sign and verify round-trip. It creates a primary ML-DSA key and signs a fixed message via SignSequenceStart and SignSequenceComplete. Pure ML-DSA sequences are streamable, so the message could also be fed through SequenceUpdate; this example passes it whole on the Complete buffer for signing. It then verifies via VerifySequenceStart, VerifySequenceUpdate, and VerifySequenceComplete. It asserts the returned validation ticket tag is TPM_ST_MESSAGE_VERIFIED.

```
./examples/pqc/mldsa_sign                 # default: ML-DSA-65
./examples/pqc/mldsa_sign -mldsa=44
./examples/pqc/mldsa_sign -mldsa=87
```

17.4.5 PQC keys via keygen and keyload

examples/keygen/keygen accepts v1.85 PQC options alongside -rsa, -ecc, -sym, and -keyedhash:

```
./examples/keygen/keygen keyblob.bin -mldsa=65          # Pure ML-DSA
./examples/keygen/keygen keyblob.bin -hash_mldsa=65     # SHA-384 pre-hash
./examples/keygen/keygen keyblob.bin -mlkem=768         # ML-KEM
```

Parameter sets:

- -mldsa=44|65|87 (default 65)
- -hash_mldsa=44|65|87 (default 65; SHA-256/384/512 pre-hash for 44/65/87)
- -mlkem=512|768|1024 (default 768)

Verify that the produced blob round-trips through TPM2_Create and TPM2_Load by loading it back:

```
./examples/keygen/keyload keyblob.bin
```

A successful load prints a transient key handle. The full matrix (three variants times three parameter sets, nine key configurations, each run through keygen and keyload) is exercised by `examples/run_examples.sh` when v1.85 is detected in `config.h`. That generic suite also covers non-PQC operations with their own TPM requirements.

17.4.6 PQC keys for parameter encryption

A post-quantum primary can key a TPM 2.0 parameter-encryption session. ML-KEM (decrypt capable) is used as the session salt key and ML-DSA (sign only) as the session bind key. The session protects the command's first sized parameter the same way an RSA or ECC salted session does. Any RSA or ECC storage key the example needs (for example the parent of a created child) is unchanged.

!!! note Parameter-encryption confidentiality comes from the session key, which a bound session derives from the bind entity's authValue (TPM 2.0 Library Part 1, Salted Session). A sign-only ML-DSA key cannot exchange a salt, and the example's bind authValue is a public constant, so an ML-DSA bind alone provides session binding but no confidentiality against a bus observer. To keep the advertised encryption real, the helper also creates a transient SRK and uses it as the asymmetric salt for the ML-DSA session: confidentiality comes from the encrypted salt while the ML-DSA key supplies the binding. A real deployment that relies on a bare bound session for confidentiality must use a bind entity whose authValue is secret and was not sent in cleartext.

`wrap_test`, `pcr/quote`, `nvram/store`, and `nvram/counter` take `-mlkem[=512|768|1024]` and `-mldsa[=44|65|87]`. `keygen` uses `-paramkey=mlkem[=...]` and `-paramkey=mldsa[=...]` because its `-mlkem` and `-mldsa` options already select the child key algorithm.

```
./examples/wrap/wrap_test -aes -mlkem=768
./examples/pcr/quote 16 quote.blob -ecc -xor -mldsa=65
./examples/nvram/counter -aes -mldsa=65
./examples/keygen/keygen keyblob.bin -ecc -aes -paramkey=mlkem=768
```

ML-KEM is a restricted decryption (salt) key, which requires a symmetric definition. The example helper sets AES-128-CFB on it, because a TPM rejects a restricted key with no symmetric algorithm via `TPM_RC_SYMMETRIC`.

17.4.7 create_primary with an ML-DSA primary

`examples/keygen/create_primary` can create an ML-DSA primary key:

```
./examples/keygen/create_primary -mldsa            # default ML-DSA-65
./examples/keygen/create_primary -mldsa=87 -oh
```

17.4.8 Post-Quantum TLS 1.3 (ML-KEM and TPM ML-DSA)

This is a full TLS 1.3 handshake where the server's ML-DSA identity key lives in the TPM. The server signs the CertificateVerify inside the TPM via the wolfTPM crypto callback. On a hardware TPM such as QVault this signing happens on-chip; the commands below drive the software fwTPM. The client performs an ML-KEM key exchange and validates the server against a software CA.

It requires a wolfSSL that routes `wc_MlDsaKey_SignCtx` to the crypto callback for device keys (private key in the TPM). That change is in wolfSSL 5.9.4-stable and later. A development snapshot must contain wolfSSL commit `6b0c832284286dbaec8e5ab35581ff470e90826b`. The commands below reuse the `fwtpm_server` you started earlier.

!!! warning This is a demo. The identity key is an unauthenticated deterministic TPM primary (empty auth), reproducible by both `gen_pqc_certs` and the server from the owner hierarchy. A primary's key material is derived from the hierarchy seed and the creation inputs, and an object auth value does not

change that derivation. Adding a non-empty auth value or policy alone therefore does not stop another caller who can authorize `CreatePrimary` under the owner hierarchy from recreating the same key. For production, prefer a provisioned child or persistent identity object together with controlled hierarchy authorization. The client validates the server chain against the demo CA but does not bind the certificate to the host name, so the demo connects to the default localhost and does not pass `-h=`. Supplying `-h=` turns on strict verification including `wolfSSL_check_domain_name`, which this leaf cannot satisfy. A production deployment should issue the leaf with a matching subjectAltName.

Three programs are involved:

- `examples/pqc/gen_pqc_certs` makes a software ML-DSA CA and a device leaf cert whose subject key is the TPM ML-DSA key.
- `examples/tls/tls_server -mldsa` recreates that TPM key and serves TLS 1.3.
- `examples/tls/tls_client -mldsa` connects, does the ML-KEM key exchange, and verifies the CA.

```
# the fwtpm_server from "Running the examples" is already listening on
 ↪  127.0.0.1:2321

# 1. certificate chain bound to the TPM key (-mldsa must match the server)
./examples/pqc/gen_pqc_certs -mldsa=65

# 2. server (same -mldsa as gen_pqc_certs)
./examples/tls/tls_server -p=11111 -mldsa=65 &

# 3. client (choose the ML-KEM group)
./examples/tls/tls_client -p=11111 -mldsa -group=ML_KEM_768
```

Options:

- `gen_pqc_certs -mldsa=44/65/87`: ML-DSA parameter set.
- `tls_server -p=<port> -mldsa=44/65/87`.
- `tls_client -h=<host> -p=<port> -group=<name>`, where `<name>` is `ML_KEM_512/768/1024` or a hybrid `SECP256R1MLKEM768` / `X25519MLKEM768` (hybrids need the matching classical curve enabled in wolfSSL).

The one-shot end-to-end test drives all three and asserts the ML-KEM group, TPM-signed ML-DSA authentication, and CA verification:

```
ENABLE_PQC_TLS=1 ./examples/run_examples.sh   # includes the PQC TLS matrix
```

17.5 Benchmarks

Measured ML-DSA and ML-KEM latencies on SEALSQ QVault TPM silicon (key generation, sign, verify, encapsulate, decapsulate), taken with `examples/bench/bench`, are in benchmarks.md. Key generation is a one-off provisioning cost. See that page for how the ML-DSA and ECDSA figures were captured before comparing them.

17.6 See Also

- benchmarks.md
- FWTPM.md
- DEVTPM.md
- spdm.md

18 SPDM Attestation and Secure Sessions

wolfTPM includes built-in SPDM (Security Protocol and Data Model, DMTF DSP0274) support for Nuvoton NPCT75x and Nations NS350 TPMs, using wolfSSL/wolfCrypt. SPDM negotiates protocol version 1.3 over the TCG SPDM-over-TPM binding. Both vendors support identity key mode (ECDHE P-384) for session establishment. The Nations NS350 additionally supports PSK (pre-shared key) mode. Once a session is established, all TPM commands and responses are encrypted with AES-256-GCM over the existing SPI or I2C bus. Identity key mode requires the responder's P-384 public key from a trusted provisioning source. The TCG exchange is a raw public key exchange (GET_PUBK and GIVE_PUB); no certificates are exchanged.

The SPDM code lives in the wolfSPDM library, included as the `lib/wolfSPDM` submodule and compiled into libwolftpm in its TPM profile. A checkout that will use SPDM needs the submodule present, either from a recursive clone or from initializing it afterward. Without it, `./configure --enable-spdm` stops with: `--enable-spdm needs the wolfSPDM submodule: run git submodule update --init lib/wolfSPDM`.

18.1 Quick start

Clone recursively, because SPDM lives in the `lib/wolfSPDM` submodule:

```
git clone --recursive https://github.com/wolfSSL/wolfTPM.git
git clone https://github.com/wolfSSL/wolfssl.git   # sibling checkout
cd wolfTPM
```

Already cloned without `--recursive`? Run `git submodule update --init lib/wolfSPDM` once inside the checkout.

18.1.1 Nuvoton NPCT75x

```
# Build wolfSSL (in the sibling checkout, then return here)
cd ../wolfssl && ./autogen.sh && \
./configure --enable-wolftpm --enable-ecc --enable-sha384 --enable-aesgcm
 ↪  --enable-hkdf --enable-sp && \
make && sudo make install && sudo ldconfig && cd -

# Build wolfTPM (submodule already present from the recursive clone)
./autogen.sh && ./configure --enable-spdm --enable-nuvoton && make

# Enable SPDM (one-time), then reset the TPM, then connect
./examples/spdm/spdm_ctrl --enable
# reset the TPM now (see "TPM reset pin control" for the gpioset and reset-HAL
 ↪  commands)
RESPONDER_PUBKEY="$(cat responder_pubkey.hex)"
./examples/spdm/spdm_ctrl --responder-pubkey "$RESPONDER_PUBKEY" --connect
```

See the TPM reset pin control section for the reset commands and their caveats. `responder_pubkey.hex` holds the trusted raw P-384 X||Y point (192 hex characters) from your provisioning records.

18.1.2 Nations NS350

```
# Build wolfSSL (in the sibling checkout, then return here)
cd ../wolfssl && ./autogen.sh && \
```

```
./configure --enable-wolftpm --enable-ecc --enable-sha384 --enable-aesgcm
 ↪  --enable-hkdf --enable-sp && \
make && sudo make install && sudo ldconfig && cd -

# Build wolfTPM (submodule already present from the recursive clone)
./autogen.sh && ./configure --enable-spdm --enable-nations && make

# Connect (identity key is factory default)
RESPONDER_PUBKEY="$(cat responder_pubkey.hex)"
./examples/spdm/spdm_ctrl --responder-pubkey "$RESPONDER_PUBKEY" --connect
```

18.2 Overview and how it works

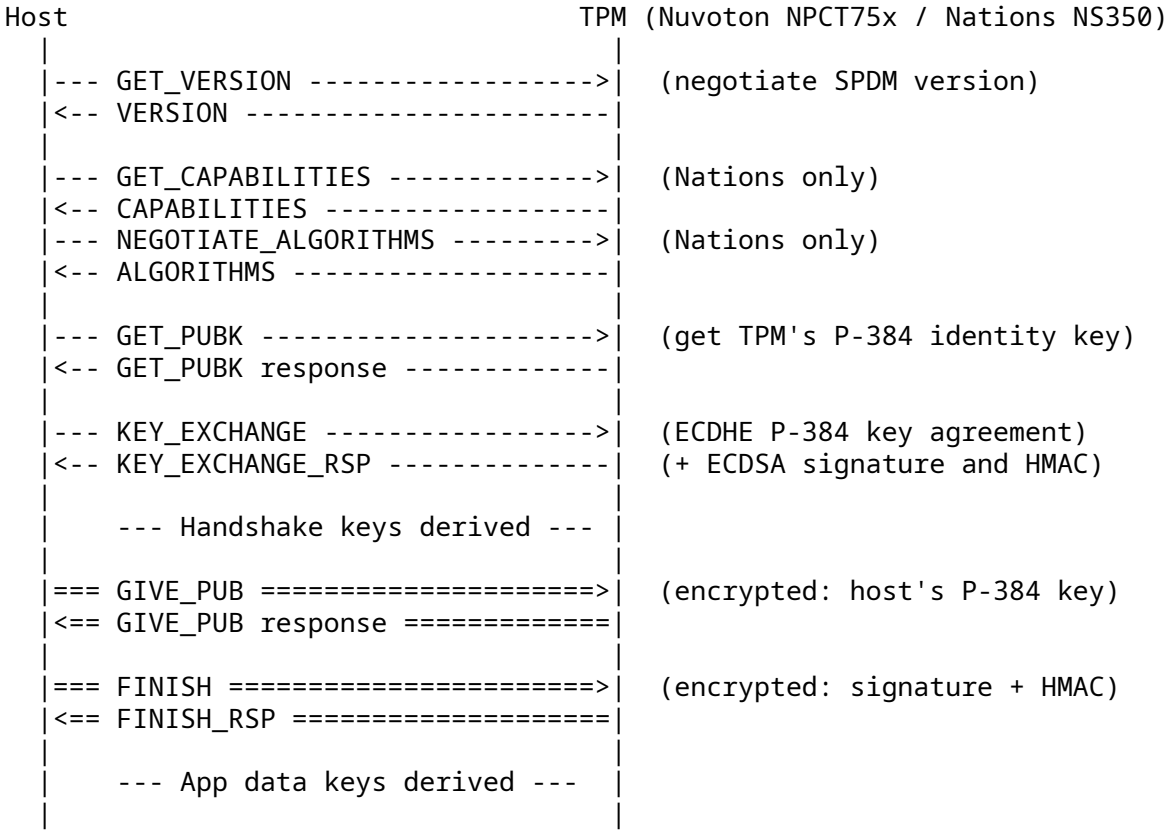
The `spdm_ctrl` tool establishes SPDM secure sessions between the host and a TPM over SPI or I2C, enabling AES-256-GCM encrypted bus communication. The implementation uses Algorithm Set B: SHA-384 and AES-256-GCM, with ECDH P-384, ECDSA P-384, and HKDF-SHA384 added in identity key mode. Two session establishment modes are supported.

`spdm_ctrl` and `nv_bind` are the examples that accept SPDM credentials. Other wolfTPM examples use uncredentialed `wolfTPM2_Init()` and intentionally return `WOLFSPDM_E_BAD_STATE` while a TPM is locked in SPDM-only mode. Unlock it with `spdm_ctrl` before running those examples.

Supported hardware:

- Nuvoton NPCT75x: identity key mode (ECDHE P-384)
- Nations NS350: identity key mode and PSK mode

18.2.1 Identity key mode (Nuvoton and Nations)



```
  |=== TPM2_CMD (AES-256-GCM) =======>|  (every command encrypted)
  |<== TPM2_RSP (AES-256-GCM) ========|
```

The Nuvoton adapter skips GET_CAPABILITIES and NEGOTIATE_ALGORITHMS; the Nations adapter sends them after VERSION. The handshake uses ECDH P-384 for key agreement, ECDSA P-384 signatures, HKDF-SHA384 for key derivation, and HMAC-SHA384 for authentication. P-384 is not part of the PSK key agreement. After the handshake, all TPM commands are wrapped in SPDM VENDOR_DEFINED_REQUEST("TPM2_CMD") messages and encrypted with AES-256-GCM. TPM responses arrive in VENDOR_DEFINED_RESPONSE messages. Secured records keep independent 64-bit request and response sequence numbers that increment with each message to prevent replay.

18.2.2 PSK mode (Nations only)

PSK mode replaces the ECDHE key exchange with a symmetric pre-shared key. The same AES-256-GCM encryption is used for data transport. The requester goes straight from GET_VERSION to PSK_EXCHANGE; capability and algorithm negotiation are not required for this flow.

```
Host                                TPM (Nations NS350)
  |                                   |
  |--- GET_VERSION ------------------>| (negotiate SPDM version)
  |<-- VERSION -----------------------|
  |                                   |
  |--- PSK_EXCHANGE ----------------->| (session key from PSK)
  |<-- PSK_EXCHANGE_RSP -------------| (+ HMAC proof)
  |                                   |
  |    --- Handshake keys derived --- | (Salt_0 = 0xFF * H for PSK mode)
  |                                   |
  |=== PSK_FINISH ==================>| (encrypted: requester HMAC)
  |<== PSK_FINISH_RSP ================|
  |                                   |
  |    --- App data keys derived ---  |
  |                                   |
  |=== TPM2_CMD (AES-256-GCM) =======>| (every command encrypted)
  |<== TPM2_RSP (AES-256-GCM) ========|
```

PSK and identity key modes are mutually exclusive on the NS350. The identity key is provisioned by factory default and must be unset before PSK can be used. See the PSK lifecycle section.

18.2.3 SPDM-only mode (encrypted bus enforcement)

SPDM-only mode forces TPM commands through the encrypted SPDM channel. The one exception is plaintext TPM2_GetCapability, which the fwTPM responder deliberately allows while locked to match the supported silicon. Both vendors support SPDM-only mode. The typical lifecycle:

```
1. Enable SPDM        (one-time, persists across resets)
2. Connect            (handshake, derives session keys)
3. Lock SPDM-only     (TPM rejects cleartext commands except GetCapability)
4. Reset              (TPM enters SPDM-only enforcement)
5. Initialize with the trusted key or PSK and run commands (all encrypted)
6. Unlock             (connect + unlock in one session)
7. Reset              (TPM back to normal cleartext mode)
```

After an application supplies the responder key through wolfTPM2_InitWithSpdmKey(), wolfTPM authenticates the responder and establishes the encrypted session. It continues after a successful or already-initialized startup probe and after the expected TPM_RC_DISABLED result. Other startup

failures, such as a firmware-upgrade state, are returned before any SPDM connection is attempted. See the Auto-SPDM section for details.

The reset method differs by vendor:

- Nuvoton: GPIO 4 reset (see TPM reset pin control)
- Nations: GPIO 4 is wired to TPM_RST on the NS350 daughter board used by the test harness, so the same GPIO reset applies there. If your board does not wire it, use a full power cycle.

18.3 Building

18.3.1 1. Clone with the wolfSPDM submodule

See the Quick start section. SPDM is built from the `lib/wolfSPDM` submodule, so clone wolfTPM recursively, or run `git submodule update --init lib/wolfSPDM` in an existing checkout.

18.3.2 2. wolfSSL

Both Nuvoton and Nations use the same wolfSSL flags, which provide the crypto for SPDM Algorithm Set B. wolfSSL 5.8.0 or later is required; the wolfSPDM configure check in the `lib/wolfSPDM` submodule enforces this.

```
cd ../wolfssl
./autogen.sh
./configure --enable-wolftpm --enable-ecc --enable-sha384 \
    --enable-aesgcm --enable-hkdf --enable-sp
make
sudo make install && sudo ldconfig
cd -   # back to the wolfTPM checkout
```

18.3.3 3. wolfTPM

```
./autogen.sh
./configure --enable-spdm --enable-nuvoton   # Nuvoton
# or
./configure --enable-spdm --enable-nations    # Nations
make
```

Build with `--enable-spdm` plus at least one handshake mode: `--enable-tcg` for the TCG raw public key handshake, `--enable-psk` for the PSK handshake. Vendor wire-format adapters (`--enable-nuvoton`, `--enable-nations`) are optional for PSK mode and the fwTPM responder, but an identity-key session needs the matching vendor adapter; without one, `wolfTPM2_InitWithSpdmKey_ex()` returns `WOLFSPDM_E_NOT_AVAILABLE`.

18.3.4 The wolfTPM SPDM profile

`--enable-spdm` defines `WOLFTPM_SPDM`, which auto-selects wolfSPDM's `WOLFSPDM_PROFILE_TPM`. The TPM binding is the TCG SPDM Binding, so this profile is a lean, TCG-focused build. The generic DSP0274 requester functions below are compiled out automatically. The Nations adapter still uses its own TCG-specific GET_CAPABILITIES and NEGOTIATE_ALGORITHMS implementations in `spdm_tcg.c`.

- the DMTF standard requester functions: `GET_CAPABILITIES`, `NEGOTIATE_ALGORITHMS`, `GET_DIGESTS`, `GET_CERTIFICATE`, and certificate-chain validation
- measurements, challenge, and chunking (they ride the cert flow)
- heartbeat and key update
- the MCTP application-data API (secured messages use the TCG 16-byte pad)

There is no `--disable-mctp` option on wolfTPM's `configure`, and you should not try to add one. The lean profile is automatic with `--enable-spdm`, and wolfSPDM's downstream CI asserts the standard-requester symbols above are absent from `libwolftpm`.

The profile does not define `WOLFSPDM_NO_MCTP`, so the MCTP secured-message framing stays compiled, though a TCG-only TPM never exercises it. Building wolfSPDM standalone with `--disable-mctp` (which requires `--enable-tcg`) additionally strips that path for a pure-TCG requester. That flag belongs to wolfSPDM's own `configure`, never wolfTPM's.

18.3.5 Configure options

Option	Description
`--enable-spdm`	Enable SPDM support (required)
`--enable-tcg`	TCG SPDM Binding spec handshake (auto when fwtpm/nuvoton/nations on)
`--enable-psk`	DSP0274 PSK handshake (auto with `--enable-nations`; requires `--enable-tcg`)
`--enable-fwtpm`	Build fwtpm_server with the SPDM responder (needs `--enable-spdm` and a handshake mode; no silicon needed)
`--enable-nuvoton`	Enable Nuvoton TPM hardware support (auto-enables `--enable-tcg`)
`--enable-nations`	Enable Nations NS350 hardware support (auto-enables `--enable-tcg --enable-psk`)
`--enable-debug`	Debug output with verbose SPDM tracing
`--enable-smallstack`	Heap-allocate the SPDM context and the SPDM request and response buffers, and lower the public message-size limits (default: caller-owned inline context, about 32 KB)

`configure` rejects these incompatible combinations:

- `--enable-nuvoton --disable-tcg` (Nuvoton uses the TCG SPDM Binding)
- `--enable-nations --disable-tcg` or `--enable-nations --disable-psk`
- `--enable-psk --disable-tcg` (PSK rides on TCG framing)

18.3.6 fwTPM SPDM responder (no silicon required)

`fwtpm_server` ships an SPDM 1.3 responder that drives the same handshake the real Nuvoton and Nations parts use, so the full TCG and PSK stack can be exercised in CI without real hardware:

Build it with the socket responder enabled:

```
./configure --enable-fwtpm --enable-swtpm --enable-spdm --enable-tcg
 ↪  --enable-psk --enable-nuvoton --enable-nations
make
```

Then start it in one of the SPDM modes. The fwTPM responder takes a non-empty PSK of at most 64 bytes (128 hex characters); the exact 64-byte requirement applies to Nations hardware provisioning, not to this responder. The value below is the test PSK used by `spdm_test.sh`:

```
SPDM_PSK=dbc2192291d807742441b963f6712841f7697e2e39c45931f3abc53658c8b9338bd3561cab5d90cf9e
./src/fwtpm/fwtpm_server --spdm-tcg                              # TCG raw
 ↪  public key handshake
./src/fwtpm/fwtpm_server --spdm-psk --spdm-psk-hex "$SPDM_PSK"   # PSK
 ↪  handshake
```

For a manual PSK test, supply the same SPDM_PSK value to the requester, for example `spdm_ctrl --psk "$SPDM_PSK"`.

Test it end-to-end:

```
./examples/spdm/spdm_test.sh ./examples/spdm/spdm_ctrl fwtpm-tcg
./examples/spdm/spdm_test.sh ./examples/spdm/spdm_ctrl fwtpm-psk
```

See fwtpm/spdm.md for the responder modes and the end-to-end test scripts.

18.3.7 Vendor selection in dual-vendor builds

When both `--enable-nuvoton` and `--enable-nations` are compiled in, `spdm_ctrl` selects the vendor adapter with an optional runtime flag:

```
./examples/spdm/spdm_ctrl --vendor=nuvoton \
    --responder-pubkey "$RESPONDER_PUBKEY" --connect
./examples/spdm/spdm_ctrl --vendor=nations \
    --responder-pubkey "$RESPONDER_PUBKEY" --connect
```

Single-vendor builds accept only the adapter compiled into the binary and reject an unavailable `--vendor=` value.

18.4 Usage and control commands

18.4.1 One-time setup

The administrative commands (enable, disable, identity key set and unset) use an empty platform authorization, and `--tpm-clear` uses the default empty lockout authorization. `spdm_ctrl` has no option to supply different hierarchy secrets, so these commands do not work on a TPM provisioned with non-empty authorizations.

Nuvoton:

```
# Enable SPDM on the TPM (persists across resets)
./examples/spdm/spdm_ctrl --enable

# Reset the TPM (see TPM reset pin control)

# Verify SPDM is enabled
./examples/spdm/spdm_ctrl --status
```

Nations: identity key mode is the factory default, so no setup is required. If it was previously unset, restore it with:

```
./examples/spdm/spdm_ctrl --identity-key-set
```

18.4.2 Establishing a session

Identity key mode (both vendors):

```
# Establish SPDM session (VERSION, GET_PUBK, KEY_EXCHANGE, GIVE_PUB, FINISH;
# Nations also sends GET_CAPABILITIES and NEGOTIATE_ALGORITHMS)
./examples/spdm/spdm_ctrl \
    --responder-pubkey "$RESPONDER_PUBKEY" --connect

# Query SPDM status
./examples/spdm/spdm_ctrl --status
```

`--responder-pubkey` takes the trusted raw P-384 X||Y point as 192 hex characters. Obtain it from device provisioning records or another authenticated manufacturer channel. The examples here read these values into shell variables, for instance `RESPONDER_PUBKEY="$(cat responder_pubkey.hex)"`. The responder public key is not secret, but it is a trust anchor, so protect it from tampering. The PSK and ClearAuth are secrets: keep those files readable only by their owner (`chmod 600`). Passing a secret as a command-line argument (for example `--psk`) exposes it to other users on the host through `/proc/<pid>/cmdline`; these example CLIs are demonstration tools, so handle real secrets accordingly on a shared machine.

!!! warning `--get-pubkey` is unauthenticated discovery and must not be used by itself to establish trust.

PSK mode (Nations) requires the PSK to be provisioned first. See the PSK lifecycle section. In the commands below, `PSK_HEX` holds the 64-byte PSK as 128 hex characters and `CLEARAUTH_HEX` holds the 32-byte ClearAuth as 64 hex characters, each read from a file only you can read.

```
# Establish PSK session (VERSION, PSK_EXCHANGE, PSK_FINISH)
./examples/spdm/spdm_ctrl --psk "$PSK_HEX"
```

18.4.3 Lock and unlock SPDM-only mode

Lock requires an active SPDM session. After locking, a reset is required for enforcement to take effect.

Nuvoton (identity key):

```
./examples/spdm/spdm_ctrl \
    --responder-pubkey "$RESPONDER_PUBKEY" --connect --lock
# Reset the TPM (see TPM reset pin control)

# Unlock
./examples/spdm/spdm_ctrl \
    --responder-pubkey "$RESPONDER_PUBKEY" --connect --unlock
# Reset the TPM again
```

Nations (identity key):

```
./examples/spdm/spdm_ctrl \
    --responder-pubkey "$RESPONDER_PUBKEY" --connect --lock
# Reset the TPM (GPIO 4 on the tested board, otherwise a power cycle)

./examples/spdm/spdm_ctrl \
    --responder-pubkey "$RESPONDER_PUBKEY" --connect --unlock
# Reset the TPM again
```

Nations (PSK mode):

```
./examples/spdm/spdm_ctrl --psk "$PSK_HEX" --lock
# Reset the TPM

./examples/spdm/spdm_ctrl --psk "$PSK_HEX" --unlock
# Reset the TPM again
```

18.4.4 PSK lifecycle (Nations)

PSK and identity key modes are mutually exclusive on the NS350. The identity key is provisioned by default and must be unset before PSK can be used.

```
# 1. Unset identity key (enables PSK mode)
./examples/spdm/spdm_ctrl --identity-key-unset
```

```
# 2. Provision PSK (64-byte PSK + 32-byte ClearAuth)
#    The demo computes SHA-384(ClearAuth) and sends PSK(64)+Digest(48) = 112
 ↪  bytes
./examples/spdm/spdm_ctrl --psk-set "$PSK_HEX" "$CLEARAUTH_HEX"

# 3. Establish PSK session
./examples/spdm/spdm_ctrl --psk "$PSK_HEX"

# 4. Clear PSK (sends raw 32-byte ClearAuth; TPM verifies SHA-384 internally)
./examples/spdm/spdm_ctrl --psk-clear "$CLEARAUTH_HEX"

# 5. Restore identity key (factory default)
./examples/spdm/spdm_ctrl --identity-key-set
```

!!! warning The ClearAuth must be exactly 32 bytes. PSK_SET stores its SHA-384 digest (48 bytes). PSK_CLEAR sends the raw 32 bytes and the TPM computes SHA-384 to verify. `spdm_ctrl` rejects a ClearAuth of the wrong length.

18.4.5 Command reference

All `spdm_ctrl` options (the set accepted depends on the vendor adapters compiled in):

Option	Vendor	Description
`--enable`	Nuvoton	Enable SPDM via NTC2_PreConfig (one-time, persists, requires reset)
`--disable`	Nuvoton	Disable SPDM via NTC2_PreConfig (requires reset)
`--identity-key-set`	Nations	Provision SPDM identity key (factory default)
`--identity-key-unset`	Nations	Remove the provisioned identity key (required before PSK)
`--vendor=nuvoton|nations`	Both	Select the identity/vendor adapter explicitly
`--get-pubkey`	Both	Discover the TPM identity key without authenticating it
`--responder-pubkey` *hex*	Both	Pin a trusted raw P-384 X||Y responder key (192 hex characters)
`--connect`	Both	Establish identity key SPDM session (ECDH P-384 handshake)
`--caps`	Both	Read TPM capabilities over the current transport
`--status`	Both	Query SPDM status
`--session-info`	Both	Show the TPM's view of the SPDM session (`TPM_CAP_SPDM_SESSION_INFO`)
`--policy-nv`	Both	Define an NV index guarded by `TPM2_PolicyTransportSPDM`, then write and read it over the session
`--lock`	Both	Lock SPDM-only mode (needs an active session: `--connect`, or `--psk` for Nations PSK)
`--unlock`	Both	Unlock SPDM-only mode (needs an active session: `--connect`, or `--psk` for Nations PSK)

Option	Vendor	Description
`--psk` *hex*	Nations	Establish PSK session (64-byte PSK)
`--psk-set` *psk clearauth*	Nations	Provision PSK (64-byte PSK, 32-byte ClearAuth)
`--psk-clear` *clearauth*	Nations	Clear PSK (32-byte ClearAuth)
`--caps184`	Nations	Query TPM 184 vendor properties and SPDM session info
`--tpm-clear`	Both	Send `TPM2_Clear` over the current transport (authorized with `TPM_RH_LOCKOUT`, default empty lockout auth)

18.4.6 Usage example

```
# One-time setup: enable SPDM + reset TPM
./examples/spdm/spdm_ctrl --enable
# Reset the TPM (see "TPM reset pin control" below)

# Query SPDM status
./examples/spdm/spdm_ctrl --status

# Discover TPM identity key (unauthenticated; do not use as its own trust
 ↪  source)
./examples/spdm/spdm_ctrl --get-pubkey

# Establish SPDM session with a key from trusted provisioning records
./examples/spdm/spdm_ctrl \
    --vendor=nuvoton --responder-pubkey "$RESPONDER_PUBKEY" --connect

# Lock SPDM-only mode (connect + lock in one session)
./examples/spdm/spdm_ctrl \
    --responder-pubkey "$RESPONDER_PUBKEY" --connect --lock
# Reset the TPM

# Unlock SPDM-only mode
./examples/spdm/spdm_ctrl \
    --responder-pubkey "$RESPONDER_PUBKEY" --connect --unlock
# Reset the TPM
```

18.4.7 nv_bind

The nv_bind example is a focused, self-contained version of the `--policy-nv` idea. It provisions an NV index whose authPolicy is `TPM2_PolicyTransportSPDM`, stores a secret over an SPDM-PSK session, then shows that the identical read over a plain (non-SPDM) connection is refused with `TPM_RC_CHANNEL`.

Start the responder in a separate terminal and leave it listening, then run nv_bind from another terminal. `--clear` deletes the fwTPM NV state file, so use a disposable instance.

```
# terminal 1: responder
./src/fwtpm/fwtpm_server --spdm-psk --spdm-psk-hex "$SPDM_PSK" --clear

# terminal 2: once the responder is listening
./examples/spdm/nv_bind --psk "$SPDM_PSK"
```

The fwTPM generates a fresh SPDM identity key each time it starts, so on the fwTPM a policy bound to `tpmKeyName` is only valid for that server lifetime. A hardware TPM holds a persistent identity key, where such a binding is durable. PSK sessions report empty key names, since no asymmetric key authenticated them.

18.4.8 TPM reset pin control

SPDM enable/disable and SPDM-only mode changes require a TPM reset to take effect. A host-controllable reset pin is the easiest way to do that, but cycling the TPM power rail also works.

!!! warning For custom hardware designs, route the TPM reset pin to a host-controllable GPIO or make the TPM power rail switchable. Without a way to reset or power cycle the TPM, SPDM mode changes cannot be applied and recovery from SPDM-only mode is not possible.

The reset line is board specific. On a Raspberry Pi, Nuvoton uses GPIO4 and the ST33KTPM uses GPIO24 (pin 18). The tested NS350 daughter board also wires GPIO4 to TPM_RST. Confirm your wiring before toggling.

With libgpiod 1.x, `gpioset` drives the line and then, in its default mode, releases the request when it exits (the chip is a positional argument). The pulse below holds each level only while the board's pull resistor does, which is what `spdm_test.sh` relies on on the tested boards:

```
gpioset gpiochip0 4=0 && sleep 0.1 && gpioset gpiochip0 4=1 && sleep 2
```

With libgpiod 2.x the chip is given with `--chip`, and `gpioset` holds the line only while the process runs. A plain && chain releases the line as each `gpioset` exits, and `--daemonize` does the opposite: it keeps the request and the line level held until the daemon is killed, which leaves the reset pin claimed and can block a later reset. Neither is a clean finite pulse. For a timed pulse, use `gpioset --toggle` (see the gpioset manual) so one process drives the line low, waits, drives it high, waits, and then exits; verify the sequence on your board first, because the released level depends on the pull resistor. For ST33 use line 24 instead of 4. For repeatable automation, prefer the wolfTPM reset HAL below.

wolfTPM can also drive the reset from code: build with `--enable-hal-reset` and call `TPM2_IoCb_Reset(ctx, userCtx)`, which takes a `TPM2_CTX*` and a `void*`. The default line is ST33 GPIO24 and Nuvoton GPIO4. A Nations build also defaults to GPIO24 unless line 4 is supplied explicitly. See `hal/README.md` in the source tree.

18.5 TCG SPDM vendor commands

Both Nuvoton and Nations TPMs implement the TCG "TPM Communication over SPDM Secure Session" binding, though the Nuvoton flow omits the GET_CAPABILITIES and NEGOTIATE_ALGORITHMS negotiation that the Nations flow performs (see Identity key mode). It carries each message as an SPDM `VENDOR_DEFINED_REQUEST` (request code `0xFE`) answered by a `VENDOR_DEFINED_RESPONSE` (response code `0x7E`), with `StandardID=0x0001` (TCG). The vendor code (VdCode) inside the message is an 8-byte ASCII string.

The published TCG table defines `GET_PUBK`, `GIVE_PUB`, `TPM2_CMD`, and optional locality-specific `TPM2CMD0` through `TPM2CMD4` values. `GET_STS_`, `SPDMONLY`, `PSK_SET_`, and `PSK_CLR_` are implementation or vendor extensions, not TCG-defined commands. The exact vendor-extension wire details for the two vendor adapters live in the `lib/wolfSPDM` submodule.

VdCode	Command	Defined by	Vendor	Description
GET_PUBK	Get Public Key	TCG	Both	Get TPM's SPDM-Identity P-384 public key

VdCode	Command	Defined by	Vendor	Description
GIVE_PUB	Give Public Key	TCG	Both	Send host's P-384 public key to TPM
TPM2_CMD	TPM Command	TCG	Both	Wrap TPM command in SPDM secured message
GET_STS_	Get Status	Vendor extension	Both	Query SPDM status
SPDMONLY	SPDM-Only Mode	Vendor extension	Both	Lock/unlock SPDM-only enforcement
PSK_SET_	PSK Set	Vendor extension	Nations	Provision pre-shared key (64-byte PSK + SHA-384 digest)
PSK_CLR_	PSK Clear	Vendor extension	Nations	Clear provisioned PSK (requires ClearAuth)

18.6 Vendor specifics

18.6.1 Nuvoton NPCT75x

- Enable/disable: SPDM is enabled via the `NTC2_PreConfig` vendor command (`--enable` / `--disable`). This persists across resets.
- GPIO reset: GPIO 4 is wired to TPM_RST on the Nuvoton daughter board. A GPIO reset clears stale SPDM state. See TPM reset pin control for the commands.

18.6.2 Nations NS350

- Mode switching: identity key and PSK modes are mutually exclusive. The identity key is provisioned by factory default. Use `--identity-key-unset` before provisioning PSK, and `--identity-key-set` to restore.
- Reset: the hardware-tested harness (`spdm_test.sh`) treats GPIO 4 as wired to TPM_RST on the NS350 daughter board and uses it when normalizing Nations state. The identity key and PSK are stored in NV and survive a reset. If your board does not wire the line, a full power cycle is needed, because `sudo reboot` leaves the 3.3V rail powered.
- Capabilities query: use `--caps184` to query TPM 184 vendor properties including SPDM session info.
- ClearAuth: must be exactly 32 bytes. PSK_SET stores its SHA-384 digest (48 bytes). PSK_CLEAR sends the raw 32 bytes and the TPM computes SHA-384 to verify.

!!! note On some NS350 firmware versions, `--status` may report "Identity Key: not provisioned" even when the key is present. The `--connect` command is the definitive test: if the ECDHE handshake succeeds, the identity key is provisioned.

Nations PSK operations can return vendor-specific error codes, for example PSK already provisioned, no PSK provisioned, an internal SPDM session error, or a ClearAuth that does not match the stored digest. The exact numeric values are firmware specific; they are not defined in public Nations material or in the source tree, so take them from the Nations integration guide for your firmware revision rather than from a fixed table here.

18.6.3 Auto-SPDM

Call `wolfTPM2_InitWithSpdmKey()` with the trusted responder key for identity mode, or `wolfTPM2_InitWithSpdmPsk()` with the provisioned PSK for PSK mode. Both entry points re-

cover a TPM that is already locked in SPDM-only mode. The identity-mode initialization sequence is:

1. TPM2_Startup probes whether the TPM is already in SPDM-only mode.
2. The caller-provided responder key is installed as the trust anchor.
3. The discovered responder key is compared with that trusted key.
4. An SPDM session is always established (P-384 keygen and handshake).
5. If the probe returned TPM_RC_DISABLED, TPM2_Startup is retried securely.
6. All subsequent commands go through the SPDM encrypted channel.

In a dual-vendor build, wolfTPM2_InitWithSpdmKey() selects the identity adapter from the TPM DID/VID. A transport that does not expose DID/VID must call wolfTPM2_InitWithSpdmKey_ex() with WOLFSPDM_MODE_NUVOTON or WOLFSPDM_MODE_NATIONS. Automatic mode fails closed rather than guessing.

wolfTPM2_Init() without a credential fails closed if it detects any SPDM-only mode. For a normal-mode TPM that does not require an immediate secure channel, identity-mode applications may instead call wolfTPM2_SpdmInit(), wolfTPM2_SpdmSetResponderPubKey(), and then the vendor-specific connect function.

Both TPM2_SendCommand (non-auth commands) and TPM2_SendCommandAuth (auth-session commands such as PCR operations, key creation, and signing) are intercepted and routed through SPDM when a session is active.

18.6.4 Memory modes

- Default: zero heap allocation. The SPDM context is a caller-owned inline context of about 32 KB. It is not necessarily static-duration storage; it lives wherever the caller places it.
- Small stack (--enable-smallstack): the wolfSPDM context and the SPDM request and response buffers are allocated with XMALLOC; some per-command buffers, such as the TPM response buffer and the TIS I/O buffer, stay on the stack. It also lowers three public message-size limits, so an oversized command or response can return BUFFER_E. Useful on platforms with small stacks; size payloads to the reduced limits.

wolfSPDM_New() exists only when wolfSPDM is built with WOLFSPDM_DYNAMIC_MEMORY. Without it, use wolfSPDM_InitStatic() or wolfSPDM_Init() on caller-provided storage.

18.7 wolfSPDM API

Function	Description
wolfSPDM_InitStatic()	Initialize context in caller-provided buffer (static mode)
wolfSPDM_New()	Allocate and initialize context on heap (only with WOLFSPDM_DYNAMIC_MEMORY)
wolfSPDM_Init()	Initialize a pre-allocated context
wolfSPDM_Free()	Free context (releases resources; frees heap only if dynamic)
wolfSPDM_GetCtxSize()	Return sizeof(WOLFSPDM_CTX) at runtime
wolfSPDM_SetIO()	Set transport I/O callback
wolfSPDM_SetResponderPubKey()	Pin the trusted responder P-384 key (identity mode)
wolfSPDM_SetPSK()	Set the pre-shared key (PSK mode)
wolfSPDM_SetMode()	Select the vendor/handshake mode
wolfSPDM_SetRequesterKeyPair()	Set the host's P-384 key pair used for GIVE_PUB and FINISH
wolfSPDM_SetDebug()	Enable/disable debug output

Function	Description
`wolfSPDM_Connect()`	Full SPDM handshake
`wolfSPDM_IsConnected()`	Check session status
`wolfSPDM_Disconnect()`	End session
`wolfSPDM_SecuredExchange()`	Encrypt/send/receive/decrypt in one call

18.8 Troubleshooting

18.8.1 Handshake fails after an interrupted session

Stale SPDM state on the TPM can make the next handshake fail. Reset the TPM.

- Nuvoton: GPIO 4 is wired to TPM_RST on the Nuvoton daughter board, so a GPIO reset clears the state. See TPM reset pin control for the commands.
- Nations NS350: the tested daughter board also wires GPIO 4 to TPM_RST, so the same GPIO reset applies. If your board does not wire it, do a full power cycle. `sudo reboot` is not sufficient because the 3.3V rail stays powered.

18.8.2 SPDM error codes

Code	Name	Description
0x01	InvalidRequest	Message format incorrect
0x04	UnexpectedRequest	Message out of sequence
0x05	Unspecified	Unspecified error
0x06	DecryptError	Decryption or MAC verification failed
0x07	UnsupportedRequest	Request not supported or format rejected
0x41	MajorVersionMismatch	SPDM major version mismatch

18.9 Standard SPDM support

The in-tree TPM profile covers only the TCG SPDM binding. For standard SPDM protocol support, including sessions with the DMTF spdm-emu emulator, measurements, challenge authentication, heartbeat, and key update, use the standalone wolfSPDM library. Those features are out of scope in wolfTPM.

18.10 Automated tests

`spdm_test.sh` runs the full SPDM setup lifecycle:

```
# Nuvoton (identity key, includes GPIO resets between tests)
SPDM_RESPONDER_PUBKEY="$(cat responder_pubkey.hex)"
export SPDM_RESPONDER_PUBKEY
./examples/spdm/spdm_test.sh ./examples/spdm/spdm_ctrl nuvoton

# Nations (identity key; the harness also uses GPIO 4 to normalize state)
./examples/spdm/spdm_test.sh ./examples/spdm/spdm_ctrl nations

# Nations (PSK, full lifecycle: provision, connect, clear, restore)
./examples/spdm/spdm_test.sh ./examples/spdm/spdm_ctrl nations-psk
```

The identity-mode hardware runs require `SPDM_RESPONDER_PUBKEY` from a trusted provisioning source (device provisioning records). The PSK run does not use it. The `fwtpm-tcg` test instead reads the freshly generated public key from the owner-only server log created by the test harness and

passes it through the same pinning interface. This local bootstrap is not a hardware provisioning mechanism.

For production use with hardware TPMs and SPDM support, contact support@wolfssl.com.

18.11 See Also

- fwtpm/spdm.md
- post-quantum.md
- FWTPM.md
- DEVTPM.md

19 fwTPM Overview

The wolfTPM fwTPM (also called fTPM) is a firmware TPM 2.0 built on wolfCrypt cryptographic primitives. It is not only an emulator for tests. It is a portable TPM 2.0 command processor that runs as a standalone server process (`fwtpm_server`) or is linked into an embedded firmware image. A default build with RSA, ECC, AES, and every feature group enabled has 103 of the 112 TPM 2.0 Revision 1.38 command codes in its dispatch table. The eight Version 185 post-quantum commands raise that to 111. See Command Coverage below for the counting convention and the documented limitations, such as the minimal self-test and the missing SU_STATE resume support.

The fwTPM can be used for testing, and it can also be used in production, security-critical, and isolated deployments where a discrete TPM chip is not available or not wanted. In those deployments the security of the TPM depends on the platform that hosts it. A firmware TPM does not provide the physical isolation of a discrete TPM chip on its own, so the integrator must supply the isolation (a separate core, a TrustZone secure world, or similar) and must protect the NV storage. The default file-based NV store keeps hierarchy seeds, authorization values, and private keys in plaintext, and `fwtpm_server` itself is a development and test tool. The fwTPM also brings post-quantum cryptography and SPDM to platforms without TPM silicon.

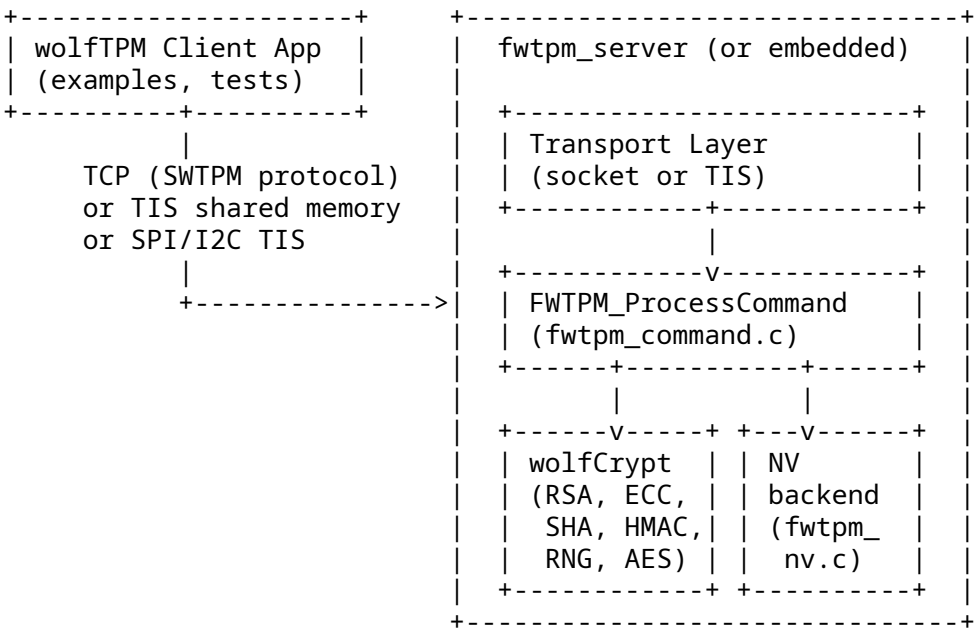
Example integration models:

- **Embedded and IoT platforms** without a discrete TPM chip (bare-metal via an SPI or I2C TIS HAL)
- **Isolated deployments**, such as a TPM running on a separate core, a TrustZone secure world, or a lock-step real-time core next to a Linux application processor. The isolation comes from the platform, not from the fwTPM.
- **Development and testing** of TPM-dependent applications (drop-in for swtpm or the Microsoft TPM simulator)
- **CI/CD pipelines** that need TPM functionality (socket transport compatible with tpm2-tools)
- **Prototyping** TPM workflows before hardware is available
- **Post-quantum and SPDM work** without TPM silicon (see Post-Quantum Support and SPDM Responder)

19.1 Features

- TPM 2.0 command processor covering 103 of the 112 Revision 1.38 command codes in a default build, with the limitations listed in Command Coverage.
- TCP socket transport using the Microsoft TPM simulator protocol, compatible with wolfTPM examples and tpm2-tools. Both the mssim and swtpm TCTI protocols are auto-detected on the command port.
- TIS register-level transport over POSIX shared memory, or over SPI or I2C for bare-metal integration.
- HAL abstractions for I/O and NV storage, so the core logic does not change when porting. See HAL and Porting.
- Post-quantum cryptography with `--enable-pqc`: ML-DSA (FIPS 204) signing and ML-KEM (FIPS 203) key encapsulation per TCG TPM 2.0 Library Specification Version 185. `--enable-pqc` and `--enable-v185` are distinct project-wide modes, but configure promotes `--enable-pqc` to the full Version 185 mode when building the fwTPM, so the two behave the same for the fwTPM. Configure auto-detects PQC when `--enable-fwtpm` is built against a wolfCrypt that has both. See Post-Quantum Support.
- SPDM 1.3 responder for testing the SPDM stack without silicon. See SPDM Responder.
- Compile-time feature gates (`FWTPM_NO_*`) to shrink the build for constrained targets. See Building.

19.2 Architecture



Both the socket and TIS transports pass each command to `FWTPM_ProcessCommand`. The command handlers then use wolfCrypt and the NV backend. The flow is the same for the standalone server and for an embedded integration.

Components:

File	Role
`fwtpm_command.c`	TPM 2.0 command processor and dispatch table
`fwtpm_io.c`	Transport layer: SWTPM TCP socket protocol (enabled by the swtpm configuration; builds without it use the TIS path)
`fwtpm_nv.c`	NV storage: file-based (default); HAL-abstracted, with a built-in append-only mode for write-once flash
`fwtpm_tis.c`	TIS register state machine (transport-agnostic)
`fwtpm_tis_shm.c`	POSIX shared memory and semaphore TIS transport
`fwtpm_main.c`	Server entry point, CLI argument parsing
`tpm2_util.c`	Shared utilities (hash helpers, ForceZero, PrintBin)
`tpm2_packet.c`	TPM packet marshaling and unmarshaling
`tpm2_param_enc.c`	Parameter encryption (XOR and AES session encryption)

19.3 Supported TPM 2.0 Commands

This section lists representative commands and is not exhaustive. It omits supported commands such as `PCR_Event`, `PCR_Allocate`, `ClockRateAdjust`, several policy commands, and some algorithm-dependent commands. The Command Coverage section below has the dispatch-table breakdown. A default build (no `FWTPM_NO_*` macro set) includes every group below. Setting a gate macro removes that group's commands from the dispatch table, from `TPM2_GetCapability(TPM_CAP_COMMANDS)`, and from the `TPM_PT_TOTAL_COMMANDS` count. See Building for the gates.

19.3.1 Startup and Self-Test

Command	Description
TPM2_Startup	Initialize TPM (SU_CLEAR or SU_STATE)
TPM2_Shutdown	Save state and prepare for power-off
TPM2_SelfTest	Minimal self-test: a SHA-256 known-answer test and an RNG check. The source calls this nonconformant.
TPM2_IncrementalSelfTest	No-op stub that returns an empty to-do list
TPM2_GetTestResult	Return self-test result

19.3.2 Random Number Generation

Command	Description
TPM2_GetRandom	Generate random bytes (max 48 per call)
TPM2_StirRandom	Add entropy to RNG state

19.3.3 Capability

Command	Description
TPM2_GetCapability	Query TPM properties, algorithms, handles

19.3.4 Key Management

Command	Description
TPM2_CreatePrimary	Create primary key under a hierarchy
TPM2_Create	Create child key under a parent
TPM2_CreateLoaded	Create and load key in one command
TPM2_Load	Load key from private and public parts
TPM2_LoadExternal	Load external (software) key
TPM2_Import	Import externally wrapped key
TPM2_Duplicate	Export key for transfer (inner and outer wrapping)
TPM2_Rewrap	Re-wrap a duplicated object from the old parent to a new parent
TPM2_FlushContext	Unload a transient object or session
TPM2_ContextSave	Save object or session context
TPM2_ContextLoad	Restore saved context
TPM2_ReadPublic	Read public area of a loaded key
TPM2_ObjectChangeAuth	Change authorization of a key
TPM2_EvictControl	Make transient key persistent (or remove)
TPM2_HierarchyControl	Enable or disable a hierarchy
TPM2_HierarchyChangeAuth	Change hierarchy authorization value
TPM2_Clear	Regenerate the storage primary seed, reset owner and endorsement authorization and policy state, and schedule eligible objects for removal. Does not regenerate the platform seed or the endorsement seed.
TPM2_ChangePPS	Replace platform primary seed

Command	Description
TPM2_ChangeEPS	Replace endorsement primary seed

19.3.5 Cryptographic Operations

Command	Description
TPM2_Sign	Sign digest with loaded key
TPM2_VerifySignature	Verify signature against loaded key
TPM2_RSA_Encrypt	RSA encryption (OAEP, PKCS1)
TPM2_RSA_Decrypt	RSA decryption
TPM2_EncryptDecrypt	Symmetric encrypt and decrypt
TPM2_EncryptDecrypt2	Symmetric encrypt and decrypt (alternate)
TPM2_Hash	Single-shot hash computation
TPM2_HMAC	Single-shot HMAC computation
TPM2_ECDH_KeyGen	Generate ephemeral ECC key pair
TPM2_ECDH_ZGen	Compute ECDH shared secret
TPM2_ECC_Parameters	Get ECC curve parameters
TPM2_TestParms	Validate algorithm parameter support

19.3.6 Hash Sequences

Command	Description
TPM2_HashSequenceStart	Start a hash sequence
TPM2_HMAC_Start	Start an HMAC sequence
TPM2_SequenceUpdate	Add data to a hash or HMAC sequence
TPM2_SequenceComplete	Finalize hash or HMAC sequence and get result
TPM2_EventSequenceComplete	Finalize hash sequence and extend PCR

19.3.7 Sealing

Command	Description
TPM2_Unseal	Unseal data from a sealed object

19.3.8 PCR (Platform Configuration Registers)

Command	Description
TPM2_PCR_Read	Read PCR values
TPM2_PCR_Extend	Extend a PCR with a digest
TPM2_PCR_Reset	Reset a resettable PCR

19.3.9 Clock

Command	Description
TPM2_ReadClock	Read TPM clock values
TPM2_ClockSet	Set TPM clock

19.3.10 Sessions and Authorization

Command	Description
TPM2_StartAuthSession	Create HMAC, policy, or trial session

19.3.11 Policy

Command	Description
TPM2_PolicyGetDigest	Get current policy session digest
TPM2_PolicyRestart	Reset policy session digest
TPM2_PolicyPCR	Bind policy to PCR values
TPM2_PolicyPassword	Include password in policy
TPM2_PolicyAuthValue	Include auth value in policy
TPM2_PolicyCommandCode	Restrict policy to specific command
TPM2_PolicyOR	Logical OR of policy branches
TPM2_PolicySecret	Authorization with secret
TPM2_PolicyAuthorize	Approve policy with signing key
TPM2_PolicyNV	Policy based on NV index comparison
TPM2_PolicyLocality	Restrict policy to specific locality
TPM2_PolicySigned	Authorize policy with external signing key

19.3.12 Dictionary Attack (DA) Protection

Command	Description
TPM2_DictionaryAttackParameters	Set maxTries, recoveryTime, lockoutRecovery
TPM2_DictionaryAttackLockReset	Reset the failed-tries counter (lockoutAuth)

The fwTPM implements dictionary-attack protection modeled on the TPM 2.0 specification (Part 1, Section 19.8). A failed authorization of a DA-protected entity increments failedTries. Once it reaches maxTries the TPM returns TPM_RC_LOCKOUT. failedTries is persisted in NV on every failure, so a power cycle cannot reset it.

When a clock HAL is registered (FWTPM_Clock_SetHAL), the counter self-heals one try per recovery-Time seconds, and a non-orderly shutdown adds a one-try penalty. On clockless builds neither applies: the persisted failedTries counter does not self-heal, and recovery is only through DictionaryAttackLockReset or Clear, so routine unclean power-off cannot accumulate into lockout.

A failed lockoutAuth locks the lockout hierarchy. On a build with a clock HAL, that lock persists across reboot and clears after lockoutRecovery seconds, except when lockoutRecovery is 0 (reboot-only recovery). On a clockless build, the failed-lockoutAuth lock is cleared on every startup. With a clock HAL, the clock HAL reports milliseconds since boot, so this timer measures continuous post-boot uptime, not wall-clock time across reboots. A device that reboots more often than lockoutRecovery extends its effective recovery window.

There are two separate gates. A failed lockoutAuth blocks later uses of lockoutAuth (DictionaryAttackLockRese DictionaryAttackParameters, and lockout-authorized Clear). Reaching maxTries blocks authorization of DA-protected objects, NV indices, and bound entities, and the TPM returns TPM_RC_LOCKOUT. The platform hierarchy is always an escape hatch: TPM2_ClearControl(platformAuth, clearDisable=NO) followed by TPM2_Clear(platformAuth) recovers even when disableClear was set. Startup and Shutdown are never DA-gated, so a reboot in lockout can always recover. Entities marked noDA (TPMA_OBJECT_noDA on objects, TPMA_NV_NO_DA on NV indices) never feed the counter and stay usable during lockout.

TPM2_GetCapability(TPM_CAP_TPM_PROPERTIES) reports TPM_PT_MAX_AUTH_FAIL, TPM_PT_LOCKOUT_INTERVA TPM_PT_LOCKOUT_RECOVERY, TPM_PT_LOCKOUT_COUNTER, and the inLockout bit of TPM_PT_PERMANENT.

Durable accounting writes the NV FLAGS entry on each DA-protected failure (and on the first DA-protected auth use per boot). The counter stops increasing while it is at maxTries, but clock-based self-healing can lower it, which allows further failed attempts and NV writes in the same boot, so the number of writes per boot is not strictly bounded. On flash-backed targets it still adds wear and makes failed-auth latency NV-bound, so size the NV backend accordingly.

The first use of a DA-protected (non-noDA) authorization after startup can make a TPM persist a daUsed flag to NV and return TPM_RC_RETRY ("resubmit the identical command") while it writes. The TCG architecture describes this as one possible implementation approach, not a requirement for every TPM. Build with FWTPM_DA_USED_RETRY to emulate it so clients exercise their resubmit and retry handling. It is off by default, and DA accounting and persistence are active regardless. Compile out all DA logic with FWTPM_NO_DA.

Coverage: DA, noDA, lockout, self-heal, and persistence unit tests in tests/fwtpm_unit_tests.c, the examples/management/da_check end-to-end example (add -lockout for the destructive lockout and recovery path), and the tests/fwtpm_da_retry.sh harness that exercises the TPM_RC_RETRY path against a FWTPM_DA_USED_RETRY build.

19.3.13 Non-Volatile Storage (NV)

Command	Description
TPM2_NV_DefineSpace	Create an NV index
TPM2_NV_UndefineSpace	Delete an NV index
TPM2_NV_ReadPublic	Read NV index public metadata
TPM2_NV_Write	Write data to NV index
TPM2_NV_Read	Read data from NV index
TPM2_NV_Extend	Extend NV index (hash-extend)
TPM2_NV_Increment	Increment NV counter
TPM2_NV_WriteLock	Lock NV index for writes
TPM2_NV_ReadLock	Lock NV index for reads
TPM2_NV_SetBits	OR bits into NV bit field index
TPM2_NV_ChangeAuth	Change NV index authorization value

19.3.14 Attestation and Credentials

Command	Description
TPM2_Quote	Generate signed PCR quote
TPM2_Certify	Certify a loaded key
TPM2_CertifyCreation	Prove key was created by this TPM
TPM2_GetTime	Signed attestation of TPM clock

Command	Description
TPM2_NV_Certify	Certify NV index contents
TPM2_MakeCredential	Create credential blob for a key
TPM2_ActivateCredential	Unwrap credential blob

19.4 Command Coverage

19.4.1 Implemented (103 Revision 1.38 command codes)

This page counts command codes, not command names, and excludes TPM_CC_Vendor_TCG_Test from standard totals. Revision 1.38 defines 112 standard command codes. With RSA, ECC, AES, and all feature groups enabled, the dispatch table holds 103 of them, which is about 92% of the Revision 1.38 set. The Version 185 post-quantum commands add 8. Reaching 113 entries requires WOLFTPM_SPDM (one more command, PolicyTransportSPDM) and the test-only vendor command as well.

Core set, never gated (36 commands): Startup, Shutdown, SelfTest, IncrementalSelfTest, GetTestResult, GetRandom, StirRandom, GetCapability, TestParms, PCR_Read, PCR_Extend, PCR_Reset, PCR_Event, PCR_Allocate, PCR_SetAuthPolicy, PCR_SetAuthValue, CreatePrimary, FlushContext, ReadPublic, Clear, ClearControl, ChangeEPS, ChangePPS, HierarchyControl, HierarchyChangeAuth, SetPrimaryPolicy, EvictControl, Create, ObjectChangeAuth, Load, Sign, VerifySignature, StartAuthSession, Unseal, LoadExternal, CreateLoaded

These are present in every build.

Optional vendor command (off by default, WOLFTPM_FWTPM_TCG_TEST): Vendor_TCG_Test

Conditional on algorithm (NO_RSA, HAVE_ECC, NO_AES): RSA_Encrypt, RSA_Decrypt, ECDH_KeyGen, ECDH_ZGen, ECC_Parameters, EC_Ephemeral, ZGen_2Phase, EncryptDecrypt, EncryptDecrypt2

Conditional on feature macros:

- FWTPM_NO_POLICY: PolicyGetDigest, PolicyRestart, PolicyPCR, PolicyPassword, PolicyAuthValue, PolicyCommandCode, PolicyOR, PolicySecret, PolicyAuthorize, PolicyLocality, PolicySigned, PolicyNV, PolicyPhysicalPresence, PolicyCpHash, PolicyNameHash, PolicyDuplicationSelect, PolicyNvWritten, PolicyTemplate, PolicyCounterTimer, PolicyTicket, PolicyAuthorizeNV (21 commands). Also gates the conditional PolicyTransportSPDM command.
- FWTPM_NO_NV: NV_DefineSpace, NV_UndefineSpace, NV_UndefineSpaceSpecial, NV_ReadPublic, NV_Write, NV_Read, NV_Extend, NV_Increment, NV_WriteLock, NV_ReadLock, NV_SetBits, NV_ChangeAuth, NV_GlobalWriteLock (13 commands). Also gates PolicyNV and PolicyAuthorizeNV when policy is enabled and removes the in-memory NV index slots from FWTPM_CTX. NV_Certify is gated by FWTPM_NO_NV as well as FWTPM_NO_ATTESTATION.
- FWTPM_NO_ATTESTATION: Quote, Certify, CertifyCreation, GetTime, NV_Certify (NV_Certify is also removed by FWTPM_NO_NV)
- FWTPM_NO_CREDENTIAL: MakeCredential, ActivateCredential
- FWTPM_NO_DA: DictionaryAttackLockReset, DictionaryAttackParameters (2 commands)
- FWTPM_NO_PARAM_ENC: Disables parameter encryption and decryption for command and response parameters. Sessions still work for HMAC auth, but encrypted transport is disabled. Reduces code size by removing AES-CFB and XOR parameter encryption.
- FWTPM_NO_KEY_MIGRATION: Import, Duplicate, Rewrap (3 commands). Shared key helpers used by Create and Load are retained.
- FWTPM_NO_ECDH: ECDH_KeyGen, ECDH_ZGen, EC_Ephemeral, ZGen_2Phase, ECC_Parameters (5 commands). ECDSA sign and verify are retained. Also drops the ecEphemeral* commit state from FWTPM_CTX.
- FWTPM_NO_HASH_CMDS: Hash, HMAC, HMAC_Start, HashSequenceStart, SequenceUpdate, SequenceComplete, EventSequenceComplete (7 commands). When WOLFTPM_MLDSA is built,

only SequenceUpdate is retained, because the ML-DSA verify sequences stream their message through it. SequenceComplete is not shared: ML-DSA sequences finalize through SignSequenceComplete and VerifySequenceComplete, so advertising it in a gated build would expose a command that can never succeed. Also drops the per-instance hash-sequence slots (`hashSeq[FWTPM_MAX_HASH_SEQ]`) from `FWTPM_CTX`.

- `FWTPM_NO_CONTEXT`: ContextSave, ContextLoad (2 commands). FlushContext is retained. Also drops the per-boot context protection key and the saved-context replay list from `FWTPM_CTX`.
- `FWTPM_NO_SYM_ENCRYPT`: EncryptDecrypt, EncryptDecrypt2 (2 commands). Nests inside `NO_AES`. AES itself is retained for session parameter encryption, AES-GCM, and (unless `FWTPM_NO_CONTEXT` is also set) context protection.
- `FWTPM_NO_CLOCK`: ReadClock, ClockSet, ClockRateAdjust (3 commands). GetTime is under `FWTPM_NO_ATTESTATION`, not this flag.
- `FWTPM_NO_PP`: PolicyPhysicalPresence and physical-presence enforcement. Drops the physical-presence HAL, the platform latch, and `FWTPM_PP_SetHAL`.

These gates are independent and there is intentionally no umbrella macro: pick exactly the groups your fTPM does not need. Applying every gate leaves the core set of 36 always-present commands, plus whatever the algorithm configuration keeps (keeping NV, or adding `FWTPM_NO_NV` to drop it). See the MicroBlaze V example in the `wolftpm-examples` repository (listed in Usage) for a worked selection.

19.4.2 Missing Commands

19.4.2.1 Revision 1.38 baseline (9 missing command codes) Medium (moderate logic, builds on existing infrastructure):

Command	Spec Section	Difficulty	Notes
`TPM2_SetCommandCodeAuditStatus`	21.2	Medium	Manage list of commands that are audited. Needs audit bitmap in context
`TPM2_PP_Commands`	26.2	Medium	Manage physical presence command list. Needs PP command bitmap

Hard (complex crypto or new subsystems):

Command	Spec Section	Difficulty	Notes
`TPM2_GetSessionAuditDigest`	18.5	Hard	Sign session audit digest. Requires session audit tracking (running hash of all commands in session). New subsystem
`TPM2_GetCommandAuditDigest`	18.6	Hard	Sign command audit digest. Requires command audit log with running hash. New subsystem
`TPM2_Commit`	19.2	Hard	DAA and anonymous attestation ephemeral key. Complex ECC point math (K, L, E generation). Needs DAA support in wolfCrypt

Command	Spec Section	Difficulty	Notes
TPM2_SetAlgorithmSet	26.3	Hard	Vendor-specific algorithm configuration. Rarely implemented, can return TPM_RC_COMMAND_CODE
TPM2_FieldUpgradeStart	27.2	Hard	Firmware upgrade initiation. Vendor-specific, requires secure update infrastructure
TPM2_FieldUpgradeData	27.3	Hard	Firmware upgrade data blocks. Vendor-specific
TPM2_FirmwareRead	27.4	Hard	Read firmware for backup. Vendor-specific

19.4.2.2 Revision 1.59 additions (5 new command codes) TPM2_MAC shares command code 0x155 with TPM2_HMAC, and TPM2_MAC_Start shares 0x15B with TPM2_HMAC_Start, so they add no new codes. The source shows only the HMAC form (no CMAC handling was found), so the symmetric-key MAC form is not confirmed as supported.

Command	Spec Section	Difficulty	Notes
TPM2_MAC	15.6	Medium	Block cipher MAC (CMAC). Like HMAC but uses symmetric key. Needs wolfCrypt CMAC
TPM2_MAC_Start	17.3	Medium	Start MAC sequence. Mirrors HMAC_Start for CMAC
TPM2_CertifyX509	18.8	Hard	Generate partial X.509 certificate. Complex ASN.1 construction, caller provides tbsCert template. Deprecated in Version 184
TPM2_AC_GetCapability	32.2	Hard	Deprecated in Version 184. Attached component capability query. Hardware-specific, rarely needed for software TPM
TPM2_AC_Send	32.3	Hard	Deprecated in Version 184. Send data to attached component. Hardware-specific
TPM2_Policy_AC_SendSelect	32.4	Medium	Deprecated in Version 184. Policy for AC_Send. Like other policy commands
TPM2_ACT_SetTimeout	33.2	Medium	Set authenticated countdown timer. Needs ACT state and timer infrastructure

19.4.2.3 Version 184 additions (9 command codes, 8 missing) Version 184 marks CreateLoaded, AC_GetCapability, AC_Send, Policy_AC_SendSelect, and CertifyX509 as deprecated. CreateLoaded is still implemented here.

Command	Spec Section	Difficulty	Notes
TPM2_ECC_Encrypt	14.8	Medium	ECC-based encryption using the TPM-defined construction from Part 1 Annex C (ephemeral ECDH point, KDF-derived masking, and integrity data). The command does not expose a choice of scheme.
TPM2_ECC_Decrypt	14.9	Medium	ECC-based decryption. Paired with ECC_Encrypt
TPM2_PolicyCapability	23.x	Easy	Assert TPM capability value in policy session
TPM2_PolicyParameters	23.x	Easy	Assert command parameters in policy session
TPM2_SetCapability	30.x	Medium	Modify TPM capability settings. Platform auth required
TPM2_NV_DefineSpace2	31.x	Medium	Extended NV space definition (larger attribute field). Extends existing NV_DefineSpace
TPM2_NV_ReadPublic2	31.x	Easy	Extended NV public read. Extends existing NV_ReadPublic
TPM2_ReadOnlyControl	24.x	Easy	Toggle TPM read-only mode. Simple flag

19.4.2.4 Conditionally implemented TPM2_PolicyTransportSPDM has a handler and a dispatch-table entry whenever WOLFTPM_SPDM is enabled, and it is removed by FWTPM_NO_POLICY. It is not missing. See SPDM Responder.

19.4.3 Coverage Summary

The eight Version 185 PQC commands (TPM2_Encapsulate, TPM2_Decapsulate, TPM2_SignDigest, TPM2_VerifyDigestSignature, TPM2_SignSequenceStart, TPM2_SignSequenceComplete, TPM2_VerifySequenceStart, TPM2_VerifySequenceComplete) are implemented under --enable-pqc. See Post-Quantum Support for the PQC-only restriction on these commands.

Spec Version	Total Command Codes	Implemented (default build)	Missing	Coverage
Revision 1.38	112	103	9	92%
Revision 1.59	117	103	14	88%
Version 184	126	103	23	82%
Version 185	134	111	23	83%

Implemented counts assume RSA, ECC, AES, and all feature groups, with SPDM, PQC (except in the Version 185 row), and the vendor test command disabled. Enabling WOLFTPM_SPDM adds PolicyTransportSPDM (one more implemented code). The Version 185 row includes the eight PQC commands.

Known limitations: TPM2_SelfTest is a minimal smoke test and TPM2_IncrementalSelfTest is a stub, as noted above, SU_STATE resume is not supported (see the lifecycle section), TPM2_PolicySecret rejects nonzero expiration because expiring policy tickets are not supported, and TPM2_DictionaryAttackParamete limits newMaxTries to 65535. The command coverage above should not be read as full conformance with the TCG specification.

COPYRIGHT ©2026 wolfSSL Inc.

19.5 Startup and Shutdown Lifecycle

1. **First boot:** `FWTPM_NV_Init` finds no NV file, generates random hierarchy seeds, and saves the initial state.
2. **`TPM2_Startup(SU_CLEAR)`:** Flushes transient objects and sessions and resets PCRs. Required before most other TPM commands. `TPM2_GetCapability` is accepted before `Startup`.
3. **Normal operation:** Commands are processed through `FWTPM_ProcessCommand`.
4. **`TPM2_Shutdown`:** Saves NV state but does not clear the "started" flag. The TPM remains logically powered on.
5. **Server restart** (process exit and relaunch) constitutes a power cycle. Only after a power cycle can `TPM2_Startup` be called again.

Calling `TPM2_Startup` on an already-started TPM returns `TPM_RC_INITIALIZE`.

Known limitation: `Startup(SU_STATE)` is accepted without checking that a matching `Shutdown(SU_STATE)` came before it, and a process restart zeroes `FWTPM_CTX` and reloads only NV-backed state. Transient objects and sessions are therefore not preserved across a restart, which a conforming TPM Resume requires. This stays a limitation until volatile state serialization and the shutdown and startup sequence checks are implemented.

19.6 Primary Key Derivation

Primary keys are deterministically derived from the hierarchy seed from the hierarchy seed using KDFa-based formulas that are specific to this implementation. The same seed, the same public template, and the same `sensitiveCreate.data` always produce the same key. Revision 1.38 discusses primary-object creation in Clause 27.

- **RSA:** Primes p and q are derived by iterative KDFa with the labels `"RSA p"` and `"RSA q"`, then primality testing, then CRT computation.
- **ECC:** The private scalar d is derived with `KDFa(nameAlg, seed, "ECC", hashUnique, counter)`, and the public point is Q = d*G.
- **KEYEDHASH and SYMCIPHER:** Key bytes are derived with `KDFa(nameAlg, seed, label, hashUnique)`. For KEYEDHASH, nonempty `sensitiveCreate.data` is used directly and is not derived.
- **hashUnique:** `H(sensitiveCreate.data || inPublic.unique)`. Because `sensitiveCreate.data` feeds `hashUnique` and the cache digest, a different value gives a different key.

A primary key cache (SHA-256 of the template, `FWTPM_MAX_PRIMARY_CACHE` slots) avoids re-deriving expensive RSA keys on repeated `CreatePrimary` calls.

Hierarchy seeds are managed by ChangePPS (platform) and ChangeEPS (endorsement). `Clear` regenerates the owner (storage primary) seed and resets endorsement authorization and policy state, but it leaves the endorsement seed and the platform seed unchanged; use ChangeEPS to replace the endorsement seed. The null seed is re-randomized on every `Startup(CLEAR)`. For post-quantum primary keys, see Post-Quantum Support.

19.7 See Also

- Building
- Usage
- HAL and Porting
- Post-Quantum Support
- SPDM Responder
- Post-Quantum (library-wide)
- SPDM (library-wide)

20 Building the fwTPM

This page covers how to build the fwTPM server, the configure options and compile defines that control it, and the compile-time macros that tune size and features. For what the fwTPM is, see the Overview.

20.1 Prerequisites

wolfSSL must be built with TPM support, keygen, and `WC_RSA_NO_PADDING`:

```
cd wolfssl
./configure --enable-wolftpm --enable-pkcallbacks --enable-keygen
 ↪  CFLAGS="-DWC_RSA_NO_PADDING"
make
sudo make install
```

20.2 Build the fwTPM Server

Socket transport (SWTPM protocol, default for development):

```
cd wolftpm
./configure --enable-fwtpm --enable-swtpm
make
```

This produces `src/fwtpm/fwtpm_server` and builds the wolfTPM client library with `WOLFTPM_SWTPM` for socket-based communication.

TIS and shared-memory transport (for fwTPM HAL integration):

```
./configure --enable-fwtpm --disable-swtpm
make
```

When `--enable-swtpm` is omitted, the build uses TIS shared-memory transport (`WOLFTPM_FWTPM_HAL`, `WOLFTPM_ADV_IO`) and compiles `fwtpm_tis.c` into the server.

fwTPM server only (no client library or examples):

```
./configure --enable-fwtpm-only --enable-swtpm
make
```

This builds only the `fwtpm_server` binary and skips `libwolftpm`, examples, and tests. It is useful for embedded targets that only need the TPM server.

Debug build:

```
./configure --enable-fwtpm --enable-swtpm --enable-debug
make
```

20.3 Build Output

Artifact	Description
`src/fwtpm/fwtpm_server`	Standalone fwTPM server binary
`src/.libs/libwolftpm.*`	wolfTPM client library

20.4 Key Build Flags

Configure Option	Effect
`--enable-fwtpm`	Build `fwtpm_server` binary (alongside client library)
`--enable-fwtpm-only`	Build only `fwtpm_server` (no client library, examples, or tests)
`--enable-swtpm`	Use SWTPM TCP socket transport (ports 2321 and 2322)
`--enable-fwtpm-nv-appendonly`	Append-only NV journal for write-once flash ports (off by default)
`--enable-pqc` (the fwTPM build promotes it to `--enable-v185`)	TPM 2.0 v1.85 post-quantum support (see Post-Quantum Support)
`--enable-spdm`	SPDM responder (with `--enable-tcg` or `--enable-psk`; see SPDM Responder)
`--enable-fuzz`	Fuzzing build
`--enable-debug`	Enable debug logging

Compile Define	Set By
`WOLFTPM_FWTPM`	Automatically set for the `fwtpm_server` target only
`WOLFTPM_SWTPM`	`--enable-swtpm`
`WOLFTPM_FWTPM_HAL`	`--enable-fwtpm --disable-swtpm`
`WOLFTPM_FWTPM_TIS`	`--enable-fwtpm --disable-swtpm`
`WOLFTPM_ADV_IO`	Set with `WOLFTPM_FWTPM_HAL`
`WOLFTPM_FWTPM_NV_APPEND_ONLY`	`--enable-fwtpm-nv-appendonly` (CMake `WOLFTPM_FWTPM_NV_APPEND_ONLY=yes`)
`WOLFTPM_FWTPM_TCG_TEST`	Manually (`CFLAGS=-DWOLFTPM_FWTPM_TCG_TEST`); off by default

No vendor commands are registered by default. Define `WOLFTPM_FWTPM_TCG_TEST` to compile in the optional `TPM2_Vendor_TCG_Test` (`0x20000000`) echo command.

20.5 Command-Code Enforcement

A valid command code carries only the 16-bit index plus, for vendor commands, the V bit (`CC_VEND`, bit 29). Codes with any other reserved bit set, or codes not in the dispatch table, are rejected with `TPM_RC_COMMAND_CODE`. `TPM2_GetCapability(TPM_CAP_COMMANDS)` returns proper `TPMA_CC` values (index, handle attributes, and V bit, ordered by command code).

20.6 Configuration Macros

All macros are compile-time overridable (for example `-DFWTPM_MAX_OBJECTS=8`).

Macro	Default	Description
FWTPM_MAX_COMMAND_SIZE	4096	Maximum command and response buffer size (bytes)
FWTPM_MAX_RANDOM_BYTES	48	Maximum bytes per GetRandom call
FWTPM_MAX_OBJECTS	3 (autoconf fwtpm_server: 7)	Maximum concurrently loaded transient objects
FWTPM_MAX_PERSISTENT	8	Maximum persistent objects (via EvictControl)
FWTPM_MAX_PRIVKEY_DER	1280 (256 with NO_RSA)	Maximum DER-encoded private key size (bytes)
FWTPM_MAX_HASH_SEQ	4	Maximum concurrent hash and HMAC sequences
FWTPM_MAX_PRIMARY_CACHE	4	Cached primary keys per hierarchy and template
FWTPM_MAX_SESSIONS	4	Maximum concurrent auth sessions
FWTPM_MAX_NV_INDICES	16	Maximum NV RAM index slots; omitted from FWTPM_CTX with FWTPM_NO_NV
FWTPM_MAX_NV_DATA	2048	Maximum data per NV index (bytes)
FWTPM_DA_DEFAULT_MAX_TRIES	32	DA failed-auth count before lockout
FWTPM_DA_DEFAULT_RECOVERY	600	DA self-heal interval (seconds per try)
FWTPM_DA_DEFAULT_LOCKOUT_RECOVERY	86400	lockoutAuth recovery time (seconds)
FWTPM_DA_MAX_TRIES_LIMIT	0xFFFF	Upper clamp for a replayed maxTries or failedTries
FWTPM_MAX_DATA_BUF	1024	Internal buffer for HMAC, hash, general data
FWTPM_MAX_PUB_BUF	512	Internal buffer for public area, signatures
FWTPM_MAX_DER_SIG_BUF	256	Internal buffer for DER signatures, ECC points
FWTPM_MAX_ATTEST_BUF	1024	Internal buffer for attestation marshaling
FWTPM_MAX_CMD_AUTHS	3	Maximum authorization sessions per command (TPM spec hard cap)
FWTPM_MAX_SENSITIVE_SIZE	FWTPM_MAX_PRIVKEY_DER + 128	Maximum marshaled sensitive area (private key, auth, and nonce headroom)
FWTPM_MAX_SIGN_SEQ	4	Maximum concurrent v1.85 PQC sign and verify sequences
FWTPM_MAX_SYM_KEY_SIZE	32	Symmetric key buffer (sized for AES-256)
FWTPM_MAX_HMAC_KEY_SIZE	64	HMAC key buffer (sized for SHA-512 block)
FWTPM_MAX_HMAC_DIGEST_SIZE	64	HMAC output buffer (sized for SHA-512)
FWTPM_CMD_PORT	2321	Default TCP command port
FWTPM_PLAT_PORT	2322	Default TCP platform port
FWTPM_NV_FILE	"fwtpm_nv.bin"	Default NV storage file path
FWTPM_NV_MAX_WRITE_ALIGN	64	Max append-only program granule in bytes (upper bound on a HAL's writeAlign); sizes the pending-granule buffer when WOLFTPM_FWTPM_NV_APPEND_ONLY is set
FWTPM_PCR_BANKS	2	Number of PCR banks (SHA-256 and SHA-384)
FWTPM_TIS_BURST_COUNT	64	TIS FIFO burst count (bytes per transfer)
FWTPM_TIS_FIFO_SIZE	4096	TIS command and response FIFO size

20.6.1 Stack and Heap Control

Macro	Effect
WOLFTPM_SMALL_STACK	Use heap allocation for large stack objects
WOLFTPM2_NO_HEAP	Forbid heap allocation (all stack)

!!! note WOLFTPM_SMALL_STACK and WOLFTPM2_NO_HEAP are mutually exclusive. Defining both is a compile error.

20.6.2 v1.85 Embedded RAM Impact

Enabling --enable-pqc (or --enable-v185) lifts several internal buffers to accommodate PQC key and signature sizes. The defaults shrink automatically at compile time based on which ML-DSA and ML-KEM parameter sets wolfCrypt was built with (WOLFSSL_NO_ML_DSA_44/65/87, WOLFSSL_NO_KYBER512/768/1024). Boards that only enable the smaller parameter sets get smaller buffers with no per-board override.

Buffer sizes by enabled parameter set:

Macro	Classical	MLDSA-44 + MLKEM-512	MLDSA-65 + MLKEM-768	MLDSA-87 + MLKEM-1024
FWTPM_TIS_FIFO_SIZE	4096	4096	8192	8192
FWTPM_MAX_COMMAND_SIZE	4096	4096	8192	8192
FWTPM_MAX_PUB_BUF	512	1440	2080	2720
FWTPM_MAX_DER_SIG_BUF	256	2548	3437	4755
FWTPM_MAX_KEM_CT_BUF	n/a	832	1152	1632

Sizing logic lives in wolftpm/fwtpm/fwtpm.h (constants FWTPM_MAX_MLDSA_SIG_SIZE, FWTPM_MAX_MLDSA_PUB_SI FWTPM_MAX_MLKEM_CT_SIZE, FWTPM_MAX_MLKEM_PUB_SIZE) and wolftpm/fwtpm/fwtpm_tis.h (FIFO size). The ML-DSA constants come from wolfCrypt's WC_MLDSA_{44,65,87}_*_SIZE macros. The ML-KEM constants are FIPS 203 spec values, because wolfCrypt's WC_ML_KEM_*_SIZE macros cannot be evaluated by the preprocessor.

The 8192 lifts on the FIFO and command buffers only apply when MLDSA-65 or MLDSA-87 is enabled, because their signatures do not fit a 4096-byte response with TPM headers. MLDSA-44-only and MLKEM-only v1.85 builds stay at 4096.

Per-deployment override: every macro above is still #ifndef-guarded, so a board can override individually on the compile line if the automatic default is wrong for its workload (for example -DFWTPM_TIS_FIFO_SIZE=2048).

Heap versus stack: building with WOLFTPM_SMALL_STACK moves the large per-call buffers off the stack into XMALLOC and XFREE regions. The PQC paths already use FWTPM_DECLARE_BUF and FWTPM_ALLOC_BUF, which respect this flag, so no source changes are required. WOLFTPM2_NO_HEAP is supported but pays the full stack cost, so pair it with the smallest PQC parameter set you can.

20.6.3 Algorithm Feature Macros

These macros use wolfCrypt's existing compile-time options to control which cryptographic algorithms are available in fwtpm_server. If an algorithm is disabled, the corresponding TPM commands are excluded from the build.

Macro	Default	Effect
NO_RSA	not defined	Excludes RSA keygen, sign, verify, RSA_Encrypt, RSA_Decrypt
HAVE_ECC	defined	Enables ECC keygen, sign, verify, ECDH_KeyGen, ECDH_ZGen, ECC_Parameters
HAVE_ECC384	defined	Enables P-384 curve support
HAVE_ECC521 or HAVE_ALL_CURVES	build-dependent	Enables P-521 when MAX_ECC_KEY_BITS >= 521 provides 66-byte TPM ECC fields
ECC_MIN_KEY_SZ	wolfCrypt-defined	Excludes smaller curves from ECC_Parameters and TPM_CAP_ECC_CURVES
NO_AES	not defined	Excludes EncryptDecrypt, EncryptDecrypt2, AES parameter encryption
WOLFSSL_SHA384	defined	Enables SHA-384 PCR bank

When an algorithm is disabled, commands that exclusively use that algorithm are removed from the dispatch table at compile time. Commands that support multiple algorithms (for example CreatePrimary and Sign) remain available but return TPM_RC_ASYMMETRIC for the disabled algorithm type.

20.6.4 TPM Feature Group Macros

These fwTPM-specific macros disable entire groups of TPM 2.0 functionality to reduce code size on constrained targets.

Macro	Default	Commands Excluded
FWTPM_NO_ATTESTATION	not defined	Quote, Certify, CertifyCreation, GetTime, NV_Certify
FWTPM_NO_NV	not defined	NV_DefineSpace, NV_UndefineSpace, NV_ReadPublic, NV_Write, NV_Read, NV_Extend, NV_Increment, NV_WriteLock, NV_ReadLock, NV_Certify; also removes the in-memory NV index slots from FWTPM_CTX
FWTPM_NO_POLICY	not defined	PolicyGetDigest, PolicyRestart, PolicyPCR, PolicyPassword, PolicyAuthValue, PolicyCommandCode, PolicyOR, PolicySecret, PolicyAuthorize, PolicyNV
FWTPM_NO_CREDENTIAL	not defined	MakeCredential, ActivateCredential
FWTPM_NO_DA	not defined	DictionaryAttackParameters, DictionaryAttackLockReset, and all lockout accounting
FWTPM_NO_PARAM_ENC	not defined	Command and response parameter (XOR and AES-CFB) encryption support in sessions
FWTPM_NO_KEY_MIGRATION	not defined	Import, Duplicate, Rewrap

Macro	Default	Commands Excluded
FWTPM_NO_ECDH	not defined	ECDH_KeyGen, ECDH_ZGen, EC_Ephemeral, ZGen_2Phase, ECC_Parameters (ECDSA sign and verify retained), plus the ecEphemeral* commit state in FWTPM_CTX
FWTPM_NO_HASH_CMDS	not defined	Hash, HMAC, HMAC_Start, HashSequenceStart, SequenceUpdate, SequenceComplete, EventSequenceComplete, and the FWTPM_CTX hash-sequence slots
FWTPM_NO_CONTEXT	not defined	ContextSave, ContextLoad (FlushContext retained), plus the per-boot context protection key and saved-context replay list in FWTPM_CTX
FWTPM_NO_SYM_ENCRYPT	not defined	EncryptDecrypt, EncryptDecrypt2
FWTPM_NO_CLOCK	not defined	ReadClock, ClockSet, ClockRateAdjust

Removing a command group also removes it from the TPM2_GetCapability(TPM_CAP_COMMANDS) advertisement and the TPM_PT_TOTAL_COMMANDS count, since both are derived from the dispatch table. When WOLFTPM_MLDSA is built, SequenceUpdate alone is retained under FWTPM_NO_HASH_CMDS, because ML-DSA verify sequences stream their message through it. SequenceComplete is not shared (ML-DSA sequences finalize through TPM2_SignSequenceComplete and TPM2_VerifySequenceComplete), so it is gated out with the rest of the hash commands rather than advertised as a command that can never succeed.

The FWTPM_DA_USED_RETRY macro (off by default) does not remove commands. It makes the server return TPM_RC_RETRY on the first DA-protected auth use after startup, emulating a real TPM persisting daUsed. See Dictionary Attack Protection in the Overview.

Minimal build example. There is no umbrella macro. Select the command groups to drop explicitly, so each is a deliberate choice. For example, to build a small ECC-only signing and NV fTPM (this set drops attestation and keeps Sign, VerifySignature, PCR, and NV):

```
./configure --enable-fwtpm --enable-swtpm \
    CFLAGS="-DNO_RSA \
        -DFWTPM_NO_POLICY -DFWTPM_NO_ATTESTATION -DFWTPM_NO_CREDENTIAL \
        -DFWTPM_NO_DA -DFWTPM_NO_PARAM_ENC -DFWTPM_NO_KEY_MIGRATION \
        -DFWTPM_NO_ECDH -DFWTPM_NO_HASH_CMDS -DFWTPM_NO_CONTEXT \
        -DFWTPM_NO_SYM_ENCRYPT -DFWTPM_NO_CLOCK"
```

That set retains a core fTPM: Startup, Shutdown, SelfTest, GetRandom, GetCapability, the PCR_* commands, Create, CreatePrimary, Load, ReadPublic, FlushContext, Sign, VerifySignature, the NV_* commands, and session support (StartAuthSession and Unseal). Add -DFWTPM_NO_NV to also drop NV, or drop any -DFWTPM_NO_* above to keep that group. This ECC-only build is small enough to run as a soft-core fTPM on a constrained FPGA (see the MicroBlaze V example in the wolftpm-examples repository, which fits an ECC-only fTPM into about 192 KB of on-chip memory).

Dependencies:

- FWTPM_NO_NV also removes NV_Certify, even if FWTPM_NO_ATTESTATION is not set.
- NO_RSA implies no RSA attestation signatures. ECC-only attestation still works with HAVE_ECC.

20.7 See Also

- Overview
- Usage

- HAL and Porting
- Post-Quantum Support
- SPDM Responder

COPYRIGHT ©2026 wolfSSL Inc.

21 Using the fwTPM

This page covers running the fwTPM server, connecting clients, transport modes, NV persistence, testing, the C API, and real-world board examples. To build the server first, see Building.

21.1 Starting the Server

```
./src/fwtpm/fwtpm_server [options]
```

Options:

Option	Description
`--help`, `-h`	Show usage information
`--version`, `-v`	Print version string
`--port <port>`	Command port (default: 2321)
`--platform-port <port>`	Platform port (default: 2322)
`--clear`	Start with cleared NV
`--spdm-tcg`, `--spdm-psk`, `--no-spdm`	SPDM responder mode (see SPDM Responder)

The `--port` and `--platform-port` options are socket-mode only and are not available in TIS builds (`--enable-fwtpm --disable-swtpm`).

Example:

```
# Start with default ports (localhost:2321 command, :2322 platform)
./src/fwtpm/fwtpm_server

# Start on custom ports
./src/fwtpm/fwtpm_server --port 2331 --platform-port 2332

# Start with clear NV
./src/fwtpm/fwtpm_server --clear
```

The server prints its configuration on startup:

```
wolfTPM fwTPM Server v0.1.0
  Command port:  2321
  Platform port: 2322
  Manufacturer:  WOLF
  Model:         fwTPM
```

In `--spdm-tcg` test mode the server also prints its generated responder public key. This is a local test-harness convenience, not a provisioning or trust-anchor channel for hardware responders.

21.2 Connecting wolfTPM Clients

Any wolfTPM application built with `--enable-swtpm` connects to the fwTPM server automatically over TCP. The built-in swtpm client speaks the mssim protocol.

```
# In one terminal: start the server
./src/fwtpm/fwtpm_server

# In another terminal: run wolfTPM examples
```

```
./examples/wrap/wrap_test
./examples/wrap/caps
./examples/keygen/keygen keyblob.bin -rsa -t
./examples/attestation/make_credential
```

21.2.1 Using tpm2-tools

In socket mode (`--enable-swtpm`), the server supports both the mssim (Microsoft TPM simulator) and swtpm (Stefan Berger) TCTI protocols, which it auto-detects on the command port. Either TCTI works:

```
# mssim TCTI (default for wolfTPM test scripts)
export TPM2TOOLS_TCTI="mssim:host=localhost,port=2321"
tpm2_startup -c

# swtpm TCTI (also works, auto-detected)
export TPM2TOOLS_TCTI="swtpm:host=localhost,port=2321"
tpm2_getrandom 8
```

21.3 NV Persistence

The server stores persistent state (hierarchy seeds, auth values, PCR state, NV indices) in `fwtpm_nv.bin` (configurable with `FWTPM_NV_FILE`). On first start, seeds are randomly generated and saved. Later starts reload the existing state.

Embedded targets replace the file backend with a flash, EEPROM, or other NV HAL, including an append-only mode for write-once flash. See HAL and Porting.

21.4 Transport Modes

21.4.1 Socket / SWTPM (Default)

Built with `--enable-fwtpm --enable-swtpm`. The server listens on two TCP ports using the SWTPM wire protocol:

- **Command port** (default 2321): TPM command and response traffic
- **Platform port** (default 2322): Platform signals (power on and off, NV on, cancel, reset, session end, stop)

SWTPM TCP protocol commands (platform port):

Signal	Value	Description
SIGNAL_POWER_ON	1	Power on the TPM
SIGNAL_POWER_OFF	2	Power off the TPM
SIGNAL_PHYS_PRES_ON	3	Assert physical presence
SIGNAL_PHYS_PRES_OFF	4	Deassert physical presence
SIGNAL_HASH_START	5	Start measured boot hash
SIGNAL_HASH_DATA	6	Provide measured boot data
SIGNAL_HASH_END	9	End measured boot hash
SEND_COMMAND	8	Send TPM command (command port)
SIGNAL_NV_ON	11	NV storage available
SIGNAL_CANCEL_ON	13	Cancel current command
SIGNAL_CANCEL_OFF	14	Clear cancel
SIGNAL_RESET	17	Reset TPM
SESSION_END	20	End TCP session

Signal	Value	Description
STOP	21	Stop server

wolfTPM clients connect through the standard SWTPM interface, which is compatible with `tpm2-tools` and other SWTPM-aware software.

21.4.2 TIS / Shared Memory

Built with `--enable-fwtpm --disable-swtpm`. This mode uses POSIX shared memory and named semaphores to emulate TIS (TPM Interface Specification) register-level access. It simulates an SPI-attached TPM.

Shared memory layout (`FWTPM_TIS_SHM`):

Field	Description
`magic` / `version`	Validation header (`0x57544953` / "WTIS", protocol version 2)
`reg_addr`, `reg_len`, `reg_is_write`, `reg_data`	Register access request
TIS register shadow: `access`, `sts`, `int_enable`, `int_status`, `intf_caps`, `did_vid`, `rid`	Emulated TIS registers
`cmd_buf[4096]`, `cmd_len`, `fifo_write_pos`	Command FIFO
`rsp_buf[4096]`, `rsp_len`, `fifo_read_pos`	Response FIFO

Paths (compile-time configurable):

Define	Default	Description
`FWTPM_TIS_SHM_PATH`	`/tmp/fwtpm.shm`	Shared memory file; clients require a regular, single-link, same-UID, exact-size `0600` endpoint
`FWTPM_TIS_SEM_CMD`	`/fwtpm_cmd`	Command semaphore name
`FWTPM_TIS_SEM_RSP`	`/fwtpm_rsp`	Response semaphore name

Clients require an exact protocol version and shared-region-size match, so rebuild the client library and `fwtpm_server` together when changing options that affect `FWTPM_TIS_FIFO_SIZE`. The default paths are global, so run one server per host.

Server-side API:

- `FWTPM_TIS_Init()`: create shared memory and semaphores
- `FWTPM_TIS_Cleanup()`: remove shared memory and semaphores
- `FWTPM_TIS_ServerLoop()`: process TIS register accesses and dispatch commands

Client-side API (enabled by `WOLFTPM_FWTPM_HAL`):

- `FWTPM_TIS_ClientConnect()`: attach to existing shared memory
- `FWTPM_TIS_ClientDisconnect()`: detach from shared memory

21.5 Testing

```
make check                  # Build + unit.test + run_examples.sh + tpm2-tools
scripts/tpm2_tools_test.sh  # tpm2-tools only (311 tests)
```

`make check` runs `tests/fwtpm_check.sh`, which starts and stops `fwtpm_server` automatically. Do not start the server manually for it.

21.5.1 CI Tests (fwtpm-test.yml)

All tests below run in GitHub Actions CI. Run them manually before PR submission. ASan, UBSan, and LeakSan coverage lives in `sanitizer.yml`, not this workflow.

Runtime tests (build, run_examples.sh, make check):

Name	wolfTPM Config	Extra	Notes
fwtpm-socket	`--enable-fwtpm --enable-swtpm --enable-debug`		Primary test
fwtpm-tis	`--enable-fwtpm --disable-swtpm --enable-debug`		TIS/SHM transport
fwtpm-v185	`--enable-fwtpm --enable-v185`		PQC: wrapper and handler unit tests
fwtpm-macos-socket	`--enable-fwtpm --enable-swtpm --enable-debug`		macOS runner

Runtime tests, gated builds (`fwtpm-gated-runtime` job):

These configurations remove commands, so `make check` (which drives the examples and tpm2-tools) does not apply. The job builds and runs only `tests/fwtpm_unit.test`. Its `test_fwtpm_command_gates`, `test_fwtpm_total_commands`, and `test_fwtpm_pcr_bounds` cases assert that a gated command is rejected with `TPM_RC_COMMAND_CODE`, is absent from `TPM_CAP_COMMANDS`, and is not counted in `TPM_PT_TOTAL_COMMANDS`.

Name	wolfTPM Config	wolfSSL Config	Extra CFLAGS
all-gates-ecc-only	`--enable-fwtpm --enable-swtpm`	`--disable-rsa`	the eleven command-group `-DFWTPM_NO_*` gates together (NV retained)
all-gates-mldsa	`--enable-fwtpm --enable-swtpm --enable-v185 --enable-mldsa`	`--enable-dilithium --enable-mlkem`	the same eleven gates; proves SequenceUpdate survives for ML-DSA while SequenceComplete does not
reduced-pcr	`--enable-fwtpm --enable-swtpm`		`-DIMPLEMENTATION_PCR=8 -DPLATFORM_PCR=8`

Build-only tests:

Name	wolfTPM Config	wolfSSL Config	Extra CFLAGS
fwtpm-no-rsa	--enable-fwtpm --enable-swtpm	--disable-rsa	
fwtpm-no-ecc	--enable-fwtpm --enable-swtpm	--disable-ecc	
fwtpm-no-sha384	--enable-fwtpm --enable-swtpm	--disable-sha384	
fwtpm-no-sha1	--enable-fwtpm --enable-swtpm	--disable-sha	-DNO_SHA
fwtpm-v185-build-only	--enable-fwtpm --enable-v185		-DDEBUG_WOLFTPM
fwtpm-only	--enable-fwtpm-only --enable-swtpm		No client library
fwtpm-minimal	--enable-fwtpm --enable-swtpm		-DFWTPM_NO_ATTESTATION -DFWTPM_NO_NV -DFWTPM_NO_POLICY -DFWTPM_NO_CREDENTIAL -DFWTPM_NO_DA -DFWTPM_NO_PARAM_ENC
fwtpm-no-policy	--enable-fwtpm --enable-swtpm		-DFWTPM_NO_POLICY
fwtpm-no-nv	--enable-fwtpm --enable-swtpm		-DFWTPM_NO_NV
fwtpm-no-attestation	--enable-fwtpm --enable-swtpm		-DFWTPM_NO_ATTESTATION
fwtpm-no-credential	--enable-fwtpm --enable-swtpm		-DFWTPM_NO_CREDENTIAL
fwtpm-no-da	--enable-fwtpm --enable-swtpm		-DFWTPM_NO_DA
fwtpm-no-param-enc	--enable-fwtpm --enable-swtpm		-DFWTPM_NO_PARAM_ENC
fwtpm-no-key-migration	--enable-fwtpm --enable-swtpm		-DFWTPM_NO_KEY_MIGRATION
fwtpm-no-ecdh	--enable-fwtpm --enable-swtpm		-DFWTPM_NO_ECDH
fwtpm-no-hash-cmds	--enable-fwtpm --enable-swtpm		-DFWTPM_NO_HASH_CMDS
fwtpm-no-context	--enable-fwtpm --enable-swtpm		-DFWTPM_NO_CONTEXT
fwtpm-no-sym-encrypt	--enable-fwtpm --enable-swtpm		-DFWTPM_NO_SYM_ENCRYPT
fwtpm-no-clock	--enable-fwtpm --enable-swtpm		-DFWTPM_NO_CLOCK
fwtpm-reduced-pcr	--enable-fwtpm --enable-swtpm		-DIMPLEMENTATION_PCR=8 -DPLATFORM_PCR=8

Name	wolfTPM Config	wolfSSL Config	Extra CFLAGS
fwtpm-no-rsa-no-policy	`--enable-fwtpm --enable-swtpm`	`--disable-rsa`	`-DFWTPM_NO_POLICY`
fwtpm-no-ecc-no-nv	`--enable-fwtpm --enable-swtpm`	`--disable-ecc`	`-DFWTPM_NO_NV`
fwtpm-small-stack	`--enable-fwtpm --enable-swtpm`		`-DWOLFTPM_SMALL_STACK`

Pedantic builds (build-only, -Werror):

Name	Compiler	Config
fwtpm-pedantic-gcc	gcc	`--enable-fwtpm --enable-swtpm`
fwtpm-pedantic-clang	clang	`--enable-fwtpm --enable-swtpm`
fwtpm-pedantic-only	gcc	`--enable-fwtpm-only`

Separate job: tpm2-tools (311 tests):

`scripts/tpm2_tools_test.sh`

21.6 API Reference

21.6.1 Core (`fwtpm.h`)

Function	Description
`int FWTPM_Init(FWTPM_CTX* ctx)`	Initialize fwTPM context, RNG, load NV state
`int FWTPM_Cleanup(FWTPM_CTX* ctx)`	Save NV, free resources, zero sensitive data
`const char* FWTPM_GetVersionString(void)`	Return version string (for example `"0.1.0"`)

21.6.2 Command Processor (`fwtpm_command.h`)

Function	Description
`int FWTPM_ProcessCommand(FWTPM_CTX* ctx, const byte* cmdBuf, int cmdSize, byte* rspBuf, int* rspSize, int locality)`	Process a raw TPM command packet and produce a response. Returns TPM_RC_SUCCESS on successful processing; the response buffer may contain a TPM error RC.

21.6.3 IO Transport (`fwtpm_io.h`)

Function	Description
`int FWTPM_IO_SetHAL(FWTPM_CTX* ctx, FWTPM_IO_HAL* hal)`	Register custom IO transport callbacks
`int FWTPM_IO_Init(FWTPM_CTX* ctx)`	Initialize the custom HAL, TCP socket, or TIS transport
`void FWTPM_IO_Cleanup(FWTPM_CTX* ctx)`	Close transport and release resources
`int FWTPM_IO_ServerLoop(FWTPM_CTX* ctx)`	Run the selected transport loop; blocks until `ctx->running` is cleared

21.6.4 NV Storage (`fwtpm_nv.h`)

Function	Description
`int FWTPM_NV_Init(FWTPM_CTX* ctx)`	Load NV state from storage or create new (generates seeds)
`int FWTPM_NV_Save(FWTPM_CTX* ctx)`	Save current TPM state to NV storage
`int FWTPM_NV_SetHAL(FWTPM_CTX* ctx, FWTPM_NV_HAL* hal)`	Register custom NV storage callbacks

21.6.5 TIS Server (`fwtpm_tis.h`)

Function	Description
`int FWTPM_TIS_Init(FWTPM_CTX* ctx)`	Create shared memory region and semaphores
`void FWTPM_TIS_Cleanup(FWTPM_CTX* ctx)`	Unlink shared memory and semaphores
`int FWTPM_TIS_ServerLoop(FWTPM_CTX* ctx)`	Process TIS register accesses (blocks)

21.6.6 TIS Client (`fwtpm_tis.h`, requires `WOLFTPM_FWTPM_HAL`)

Function	Description
`int FWTPM_TIS_ClientConnect(FWTPM_TIS_CLIENT_CTX* client)`	Attach to fwTPM shared memory
`void FWTPM_TIS_ClientDisconnect(FWTPM_TIS_CLIENT_CTX* client)`	Detach from shared memory

21.7 Real-world examples

The wolftpm-examples repository holds complete fwTPM projects for real boards. Each one shows a different isolation or storage choice.

Board	Project	What it shows
STM32H5 NUCLEO-H563ZI	STM32/fwtpm-stm32h5	fwTPM in the Cortex-M33 TrustZone secure world, internal-flash NV, UART with the mssim protocol
Xilinx ZCU102 (R5, lock-step)	Xilinx/fwtpm-zcu102-r5	AMP: fwTPM bare-metal on the Cortex-R5 pair in lock-step, PetaLinux client on the A53 over OpenAMP RPMsg; volatile DDR NV or persistent QSPI
Xilinx ZC702 (A9)	Xilinx/fwtpm-zc702-a9	SRAM-PUF derived device-unique NV key, so no root key is stored in flash
SCU35 (MicroBlaze-V soft core)	Xilinx/fwtpm-scu35-microblazev	ECC-only fwTPM that fits in about 190 KB of block RAM
PolarFire SoC MPFS250T	Microchip/fwtpm-polarfire-miv	AMP: fwTPM bare-metal on a U54 hart, isolated from Linux, TIS over shared L2-LIM memory
PolarFire MPF300 Splash (soft MIV)	Microchip/miv-mpf300-splash	Soft Mi-V core with persistent on-die sNVM

The STM32H5, PolarFire SoC, and ZCU102 projects are also listed as ports in HAL and Porting.

21.8 See Also

- Overview
- Building
- HAL and Porting
- Post-Quantum Support
- SPDM Responder

22 HAL and Porting

The fwTPM provides hardware abstraction layers (HALs) for the server transport, persistent storage, and the platform clock. Full reference ports for several boards live in the wolftpm-examples repository.

22.1 IO HAL (Transport)

The IO HAL abstracts the transport between the fwTPM server and its clients. Socket builds use TCP sockets (SWTPM protocol) by default. Registering an FWTPM_IO_HAL replaces the socket transport with callbacks for an embedded or in-process transport. TIS builds use their separate TIS HAL.

Callback structure (defined as FWTPM_IO_HAL in fwtpm.h):

Callback	Signature	Description
send	int (*)(void* ctx, int type, const void* buf, int sz)	Send data to the command or platform client
recv	int (*)(void* ctx, int type, void* buf, int sz)	Receive data from the command or platform client
wait	int (*)(void* ctx, int timeoutMs)	Wait up to timeoutMs for data or connections. Returns 0 on timeout or a bitmask: 0x01=command data, 0x02=platform data, 0x04=new command connection, 0x08=new platform connection
accept	int (*)(void* ctx, int type)	Accept new connection (type: 0=command, 1=platform)
close_conn	void (*)(void* ctx, int type)	Close connection (type: 0=command, 1=platform)
ctx	void*	User context pointer

The type argument is FWTPM_IO_TYPE_COMMAND or FWTPM_IO_TYPE_PLATFORM. The send, recv, and wait callbacks are required. wait must return by its requested timeout so stop requests are observed; override FWTPM_IO_WAIT_TIMEOUT_MS to change the default 1000 ms interval. Connectionless transports may omit accept and close_conn; a transport that provides accept must also provide close_conn and clean up any partial connection state before returning an accept error.

Registration:

```
FWTPM_IO_HAL myHal;
XMEMSET(&myHal, 0, sizeof(myHal));
myHal.send = my_send;
myHal.recv = my_recv;
myHal.wait = my_wait;
myHal.accept = my_accept;
myHal.close_conn = my_close;
myHal.ctx = &myTransportCtx;

FWTPM_IO_SetHAL(&ctx, &myHal);
```

Register the IO HAL before FWTPM_Init(), or after FWTPM_Init() but before FWTPM_IO_Init().

22.2 NV HAL (Persistent Storage)

The NV HAL abstracts persistent storage. The default implementation uses a local file (`fwtpm_nv.bin`). For embedded targets, replace it with flash, EEPROM, or other non-volatile storage callbacks.

Callback structure (defined as `FWTPM_NV_HAL` in `fwtpm.h`):

Callback	Signature	Description
`read`	`int (*)(void* ctx, word32 offset, byte* buf, word32 size)`	Read from NV at offset
`write`	`int (*)(void* ctx, word32 offset, const byte* buf, word32 size)`	Write to NV at offset
`erase`	see `fwtpm.h`	Erase the NV region (used by flash ports and by compaction)
`ctx`	`void*`	User context pointer
`maxSize`	`word32`	Size of the NV region in bytes
`appendOnly`, `writeAlign`	fields	Append-only mode and program granule size (see below)
`get_integrity_key`	callback	Supplies a device secret used to authenticate the journal

Registration:

```
FWTPM_NV_HAL myNvHal;
myNvHal.read = my_flash_read;
myNvHal.write = my_flash_write;
myNvHal.ctx = &myFlashCtx;

FWTPM_NV_SetHAL(&ctx, &myNvHal);
```

Register the HAL before `FWTPM_Init()`.

22.2.1 NV Storage HAL for Embedded Ports

NV access goes through `FWTPM_NV_HAL` (`read`, `write`, `erase`, `ctx`, `maxSize`, `get_integrity_key`). The default backend is a file. An embedded port supplies its own HAL and registers it before `FWTPM_Init()`.

The journal is log-structured. On a byte-addressable backend (the file default) it writes TLV entries at byte-granular offsets, rewrites the header in place, and rewrites a trailing integrity MAC after every append. Internal flash and NOR are write-once and aligned to the program granularity, so they cannot service in-place rewrites.

For those parts, build with `--enable-fwtpm-nv-appendonly` (`-DWOLFTPM_FWTPM_NV_APPEND_ONLY`, CMake `WOLFTPM_FWTPM_NV_APPEND_ONLY=yes`) and set `appendOnly` and `writeAlign` on the HAL. The journal then runs in append-only mode:

- The header is written only at compaction.
- `writePos` is derived by scanning on load.
- Each commit is sealed with an appended MAC-checkpoint entry, padded up to `writeAlign`.

The existing `read`, `write`, and `erase` HAL is the integration point. There is no separate adapter.

```
/* Native flash HAL. In append-only mode the journal only ever calls write()
 * with writeAlign-aligned, forward, into-erased bytes, so write() is a simple
```

```
 * flash program; erase() erases the region (sector loop); read() reads raw. */
FWTPM_NV_HAL hal;
XMEMSET(&hal, 0, sizeof(hal));
hal.read = myRead; hal.write = myProgram; hal.erase = myErase;
hal.ctx = myCtx; hal.maxSize = NV_SIZE;
hal.appendOnly = 1;
hal.writeAlign = PROG_SIZE;           /* flash word size, e.g. 16 (STM32H5) */
hal.get_integrity_key = myDeviceSecret;/* recommended on flash */
FWTPM_NV_SetHAL(&ctx, &hal);          /* before FWTPM_Init() */
```

In append-only mode the journal buffers a pending program granule internally and flushes full, aligned granules through `write()`. A programmed cell is never rewritten, and a whole sector is erased only on compaction. The header sector is therefore not erased on every append, and a torn final commit (for example from power loss) is ignored on the next load while all previously committed state survives.

A `get_integrity_key` callback is strongly recommended on flash, so the MAC checkpoints authenticate the journal and reject a torn or tampered tail. Setting `writeAlign <= 1` selects no buffering, so byte-writable NV (EEPROM or FRAM) works with a plain `write()`.

!!! warning Compaction still erases the whole region before rewriting, so a power loss during compaction itself remains a vulnerable window. A future two-region ping-pong layout would close it.

22.3 Clock HAL

The clock HAL is optional. It supplies `get_ms()`, which returns milliseconds since boot. Register it with `FWTPM_Clock_SetHAL()` before `FWTPM_Init()`. With a clock HAL registered, dictionary attack protection self-heals over time (see Overview).

22.4 Porting Example

A bare-metal target can register SPI transport and SPI flash NV callbacks before initialization:

```
FWTPM_CTX ctx;
int cleanupRc;
int rc;
XMEMSET(&ctx, 0, sizeof(ctx));

/* Set custom NV storage before FWTPM_Init, which loads NV state through it */
FWTPM_NV_HAL nvHal = {
    .read = spi_flash_read,
    .write = spi_flash_write,
    .ctx = &flashHandle
};
FWTPM_NV_SetHAL(&ctx, &nvHal);

/* Register a connectionless IO transport before initialization. */
FWTPM_IO_HAL ioHal = {
    .send = spi_slave_send,
    .recv = spi_slave_recv,
    .wait = spi_slave_poll,
    .accept = NULL,
    .close_conn = NULL,
    .ctx = &spiHandle
};
FWTPM_IO_SetHAL(&ctx, &ioHal);
```

```
rc = FWTPM_Init(&ctx);
if (rc == 0) {
    rc = FWTPM_IO_Init(&ctx);
    if (rc == 0) {
        rc = FWTPM_IO_ServerLoop(&ctx);  /* blocks */
        FWTPM_IO_Cleanup(&ctx);
    }
    cleanupRc = FWTPM_Cleanup(&ctx);
    if (rc == 0)
        rc = cleanupRc;
}
return rc;
```

22.5 Available Ports

Port	Repository	Description
STM32H5	STM32/fwtpm-stm32h5	STM32H5 Cortex-M33 with TrustZone (CMSE); internal-flash NV
PolarFire SoC	Microchip/fwtpm-polarfire-miv	MPFS250T; fwTPM bare-metal in M-mode on a U54 RISC-V hart (HSS AMP alongside Linux), TIS over shared L2-LIM memory
Zynq UltraScale+ ZCU102	Xilinx/fwtpm-zcu102-r5	ZynqMP MPSoC; fwTPM bare-metal on the Cortex-R5 RPU pair in lock-step, OpenAMP RPMsg client on the A53 (PetaLinux); volatile DDR NV or persistent QSPI

22.6 Porting Guide

To add a new platform, implement these HAL callbacks:

1. **NV Storage HAL** (`FWTPM_NV_HAL`): `read()`, `write()`, and `erase()` for persistent flash storage. Register it with `FWTPM_NV_SetHAL()` before `FWTPM_Init()`.

   The NV journal is log-structured. On a byte-addressable backend it writes at byte-granular offsets and rewrites the header and a trailing integrity MAC in place after every append. Internal flash and NOR are write-once and aligned to the program granularity, so they cannot service in-place rewrites. For those, build with `--enable-fwtpm-nv-appendonly` and set `hal.appendOnly = 1` and `hal.writeAlign = <program size>` (for example 16 on STM32H5) before `FWTPM_NV_SetHAL()`.

   The journal then writes the header only at compaction, derives `writePos` by scanning on load, seals each commit with an appended, program-granule-aligned MAC checkpoint, and buffers a pending program granule internally. It only ever calls `write()` with `writeAlign`-aligned, forward, into-erased bytes. Your `write()` is therefore a simple flash program with no buffering or read-modify-write in the port, `erase()` erases the region (sector loop), and `read()` reads raw bytes. A programmed cell is never rewritten and a whole sector is erased only on compaction, so the header sector is not worn on every append and a torn final commit is ignored on the next load. Provide a `get_integrity_key` on `FWTPM_NV_HAL` so the checkpoints authenticate the journal. Setting `writeAlign <= 1` disables buffering for byte-writable NV such as EEPROM or FRAM.

2. **Clock HAL** (optional): `get_ms()` returning milliseconds since boot. Register it with `FWTPM_Clock_SetHAL()` before `FWTPM_Init()`.
3. **Entry point**: zero `FWTPM_CTX`, register the HALs, call `FWTPM_Init()`, then process TPM commands with `FWTPM_ProcessCommand()`.

See the STM32 port in wolftpm-examples for a complete reference implementation.

22.7 See Also

- Overview
- Building
- Usage
- Post-Quantum Support
- SPDM Responder

23 fwTPM Post-Quantum Support (TPM 2.0 v1.85)

The fwTPM implements the post-quantum additions from TCG TPM 2.0 Library Specification v1.85, using wolfCrypt's FIPS 203 (ML-KEM) and FIPS 204 (ML-DSA) modules. This brings post-quantum keys, signing, and key encapsulation to platforms that have no TPM silicon. These v1.85 commands raise the implemented count from 103 to 111 commands. For the library-wide post-quantum view, see Post-Quantum.

Enable it with `--enable-pqc`, which the fwTPM build promotes to the full `--enable-v185`, at configure time. It is also auto-detected when `--enable-fwtpm` is built against a wolfCrypt that has both ML-DSA and ML-KEM available. Both flags set the internal `WOLFTPM_V185` macro that gates the implementation. Pass `--disable-pqc` to opt out when auto-detect would otherwise enable it.

23.1 Algorithms

Alg	Parameter Sets	Use
`TPM_ALG_MLKEM` (0x00A0)	ML-KEM-512 / 768 / 1024	Key encapsulation (decrypt-only keys)
`TPM_ALG_MLDSA` (0x00A1)	ML-DSA-44 / 65 / 87	Pure ML-DSA message signing
`TPM_ALG_HASH_MLDSA` (0x00A2)	HashML-DSA-44 / 65 / 87	Pre-hashed ML-DSA signing

23.2 Commands

The eight v1.85 PQC commands live in `src/fwtpm/fwtpm_command.c`:

Command	CC	Purpose
`TPM2_Encapsulate`	`0x000001A7`	ML-KEM encapsulation, returns sharedSecret and ciphertext
`TPM2_Decapsulate`	`0x000001A8`	ML-KEM decapsulation from ciphertext (requires USER auth)
`TPM2_SignSequenceStart`	`0x000001AA`	Begin ML-DSA sign sequence
`TPM2_SignSequenceComplete`	`0x000001A4`	Finalize sign sequence with message buffer
`TPM2_VerifySequenceStart`	`0x000001A9`	Begin ML-DSA verify sequence
`TPM2_VerifySequenceComplete`	`0x000001A3`	Finalize verify sequence, returns TPMT_TK_VERIFIED
`TPM2_SignDigest`	`0x000001A6`	One-shot digest sign (HashML-DSA or ext-mu ML-DSA)
`TPM2_VerifyDigestSignature`	`0x000001A5`	Verify digest signature

23.3 Primary Key Derivation

PQC primary keys follow the same deterministic derivation model as RSA and ECC: hierarchy seed and template, then a KDFa-derived seed, then FIPS 203 or FIPS 204 key expansion.

- **ML-DSA:** `KDFa(nameAlg, seed, "MLDSA", hashUnique)` gives a 32-byte Xi. `wc_MlDsaKey_MakeKeyFromSee` turns that into the public key and the expanded private key. The wire format stores only the 32-byte Xi per TCG Part 2 Table 210.
- **HashML-DSA:** the label is `"HASH_MLDSA"`, with the same seed size and expansion.
- **ML-KEM:** `KDFa(nameAlg, seed, "MLKEM", hashUnique)` gives a 64-byte value (d followed by z). `wc_MlKemKey_MakeKeyWithRandom` turns that into the encapsulation and decapsulation keys. The wire format stores only the 64-byte seed per TCG Part 2 Table 206.

!!! note These label strings are an interpretation. TCG Part 4 v185, which would specify them normatively, is unpublished. They are subject to change if a later release candidate or Part 4 v185 prescribes different labels.

23.4 Sign and Verify Sequences

Pure ML-DSA sequences are streamable on both sign and verify, so `TPM2_SequenceUpdate` is accepted. `TPM_RC_ONE_SHOT_SIGNATURE` applies to multi-pass schemes such as EdDSA, not to pure ML-DSA. A caller can also pass the whole message through the `buffer` parameter of `TPM2_SignSequenceComplete`. Verify sequences accumulate the message through `TPM2_SequenceUpdate`, because `TPM2_VerifySequenceComplete` has no buffer parameter.

HashML-DSA sequences (both sign and verify) use wolfCrypt's `wc_HashAlg` context to stream the message into the key's hash algorithm. `TPM2_SignSequenceComplete` finalizes the hash and calls `wc_MlDsaKey_SignCtxHash`.

Signature wire formats differ per spec Part 2 Table 217:

- **Pure ML-DSA:** `TPM2B_SIGNATURE_MLDSA`, laid out as `sigAlg + size + bytes`
- **HashML-DSA:** `TPMS_SIGNATURE_HASH_MLDSA`, laid out as `sigAlg + hashAlg + size + bytes`

23.5 Buffer Constants

Under `WOLFTPM_V185`, buffers are lifted to fit ML-DSA-87 signatures (4627 bytes) and public keys (2592 bytes):

Symbol	v1.38	v1.85
`FWTPM_MAX_COMMAND_SIZE`	4096	8192
`FWTPM_MAX_PUB_BUF`	512	2720
`FWTPM_MAX_DER_SIG_BUF`	256	4736
`FWTPM_MAX_KEM_CT_BUF`	n/a	1600
`FWTPM_TIS_FIFO_SIZE`	4096	8192
`FWTPM_NV_PUBAREA_EST`	600	2720

These are the worst-case values. The defaults shrink at compile time to match the parameter sets wolfCrypt was built with. See "v1.85 Embedded RAM Impact" in Building for the per-parameter-set table.

23.6 Limitations and Scope

The v1.85 commands are implemented for post-quantum keys only. Non-PQC key types are rejected with `TPM_RC_KEY` or `TPM_RC_SCHEME`, even when the v1.85 spec defines the commands generically:

- `TPM2_Encapsulate` and `TPM2_Decapsulate`: ML-KEM only. ECC DHKEM (the Table 100 `ecdh` arm with a non-NULL KDF) is not implemented.

- `TPM2_SignSequenceStart`, `TPM2_VerifySequenceStart`, `TPM2_SignSequenceComplete`, and `TPM2_VerifySequenceComplete`: ML-DSA and HashML-DSA only. Classical schemes (RSASSA, RSAPSS, ECDSA, SM2, ECSCHNORR, HMAC) that the spec also permits through these commands are not supported.
- `TPM2_SignDigest` and `TPM2_VerifyDigestSignature`: ML-DSA and HashML-DSA only. Classical digest signing (RSASSA, RSAPSS, ECDSA) over these new commands is not supported. Use the existing `TPM2_Sign` and `TPM2_VerifySignature` commands for those schemes.

23.7 Deferred and Out of Scope

Three v1.85 features are deferred, each for a documented reason:

1. **ML-KEM-salted sessions.** Part 3 Sec.11.1 (`TPM2_StartAuthSession`) does not describe an ML-KEM bullet alongside the RSA-OAEP and ECDH paths, even though Part 2 Sec.11.4.2 Table 222 defines the `mlkem` arm of `TPMU_ENCRYPTED_SECRET`. Part 4 v185, which would specify this normatively, is not yet published. Current behavior: `TPM2_StartAuthSession` returns `TPM_RC_KEY` for an ML-KEM tpmKey. Revisit when Part 4 v185 is published.
2. **External-mu ML-DSA signing.** wolfCrypt has no mu-direct sign API. Part 2 Sec.12.2.3.7 says "512-byte external Mu", but FIPS 204 Algorithm 7 Line 6 produces 64 bytes (SHAKE256 output). This is pending a wolfCrypt API addition and TCG errata confirmation. Current behavior: `TPM_RC_SCHEME` for ext-mu paths, and `TPM_RC_EXT_MU` for Pure ML-DSA keys without `allowExternalMu`.
3. **ECC KEM arm of Encapsulate and Decapsulate.** Part 2 Sec.10.3.13 Table 100 has both `mlkem` and `ecdh` arms, but the table note allows implementations to modify the union based on the algorithms they support. The fwTPM supports the `mlkem` arm only.

23.8 Test Coverage

`tests/fwtpm_unit_tests.c` includes ten PQC tests that exercise the full path:

- CreatePrimary for ML-KEM-768 and ML-DSA-65
- Full Encapsulate and Decapsulate round-trip (shared secret byte match)
- HashML-DSA SignDigest and VerifyDigestSignature round-trip
- Pure ML-DSA sign sequence and verify sequence round-trip
- Dual-source known-answer tests (NIST ACVP and wolfSSL internal vectors) for ML-DSA-44 verify, ML-DSA-44 keygen determinism, ML-KEM-512 encapsulation with pinned randomness, and ML-KEM-512 keygen determinism
- LoadExternal of a NIST ACVP ML-DSA-44 public key through the fwTPM handler

23.9 See Also

- Overview
- Building
- Usage
- SPDM Responder
- Post-Quantum (library-wide)

24 fwTPM SPDM Responder

The fwTPM ships an SPDM 1.3 responder, so the full SPDM stack can be exercised against a TPM that has no silicon behind it. It supports both the TCG raw public key handshake (GET_PUBK and GIVE_PUB, no certificates) and the DSP0274 pre-shared key (PSK) handshake. This lets you develop and test SPDM-secured TPM communication in CI or on a workstation before real hardware is available. For the library-wide SPDM view, see SPDM.

24.1 How It Works

When SPDM is on, the responder sits above the existing transport HAL and dispatches TCG-framed messages into the SPDM state machine. The two message tags are:

Tag	Meaning
`0x8101`	Clear (unsecured) SPDM message
`0x8201`	Secured SPDM message

Plaintext TPM frames fall through to the regular command dispatcher until the requester issues `SPDMONLY LOCK`. After that, only `TPM2_GetCapability` is allowed through in plaintext, which matches the behavior of Nuvoton and Nations silicon.

24.2 Building

Build with `--enable-fwtpm --enable-spdm` plus at least one of `--enable-tcg` or `--enable-psk`:

```
./configure --enable-fwtpm --enable-swtpm --enable-spdm --enable-tcg
 ↪  --enable-psk --enable-nuvoton --enable-nations
make
```

24.3 Starting the Responder

Start the server in one of three modes:

```
SPDM_PSK=dbc2192291d807742441b963f6712841f7697e2e39c45931f3abc53658c8b9338bd3561cab5d90cf9e
./src/fwtpm/fwtpm_server --spdm-tcg                                  # TCG raw
 ↪  public key handshake
./src/fwtpm/fwtpm_server --spdm-psk --spdm-psk-hex "$SPDM_PSK"   # PSK
 ↪  handshake
./src/fwtpm/fwtpm_server --no-spdm                                   # plaintext
 ↪  only (default)
```

The responder takes a non-empty PSK of at most 64 bytes (128 hex characters); the exact 64-byte requirement applies to Nations hardware provisioning, not to this responder. The value above is the test value used by `spdm_test.sh`. For a manual PSK test, give the requester the same value, for example `spdm_ctrl --psk "$SPDM_PSK"`.

24.4 Responder Identity Key

The responder generates a fresh P-384 identity keypair at startup. It is used to sign GET_PUBK and KEY_EXCHANGE. The private key never leaves `fwtpm_server` memory, and the stack copy is zeroed with `wc_ForceZero` after it is handed to the responder context.

In TCG mode, the server prints the public half during startup so the local test harness can pass it to the requester through the responder-key pinning API.

!!! warning This printed public key is a test bootstrap channel. It is not a substitute for authenticated device provisioning, and it is not a trust anchor for hardware responders.

24.5 Testing

End-to-end coverage uses the same script that drives real silicon:

```
./examples/spdm/spdm_test.sh ./examples/spdm/spdm_ctrl fwtpm-tcg
./examples/spdm/spdm_test.sh ./examples/spdm/spdm_ctrl fwtpm-psk
```

CI exercises 7 build-only configure permutations plus the two end-to-end modes on `ubuntu-latest` through `spdm-test.yml`, against the fwTPM SPDM responder.

24.6 See Also

- Overview
- Building
- Usage
- Post-Quantum Support
- SPDM (library-wide)

25 Rust Wrapper

The `wolftpm` crate provides safe Rust bindings for wolfTPM. The raw FFI is generated with bindgen and kept in the `sys` module. The rest of the crate is a safe API: functions return `Result`, TPM handles are released when they go out of scope, and all `unsafe` stays inside the crate. The crate lives in `wrapper/rust/wolftpm` in the wolfTPM source tree.

25.1 Requirements

- Rust and Cargo, stable, rustc 1.81 or newer.
- A built wolfTPM C library (`libwolftpm`) and its wolfSSL dependency (`libwolfssl`). The crate links a prebuilt library. It does not build the C code itself.
- For the tests, a software TPM. wolfTPM's own `fwtpm_server` works, or any other TPM emulator or software TPM. See fwTPM and SWTPM.

25.2 Step 1: Build the C Library

Run these from the wolfTPM repository root:

```
./autogen.sh
./configure --enable-swtpm --enable-fwtpm
make
```

This produces `src/.libs/libwolftpm`. With `--enable-fwtpm` it also builds the software TPM server at `src/fwtpm/fwtpm_server`.

25.3 Step 2: Build the Crate

```
cd wrapper/rust/wolftpm
cargo build
```

The build finds the libraries in this order:

1. If `WOLFTPM_PREFIX` or `WOLFSSL_PREFIX` are set, it uses `$PREFIX/include` and `$PREFIX/lib` of an installed copy.
2. Otherwise it uses the in-tree build: headers from the repo root, libraries from `src/.libs`. wolfSSL is located with `pkg-config`, then a local `./wolfssl` or a sibling `../wolfssl` checkout.

Shared libraries are preferred. Static is used when no shared library is present.

The library is `no_std` and uses `alloc` for owned keys, blobs, and output buffers. Bare-metal applications must provide a global allocator. The build script translates Rust cross-compilation targets such as `riscv32imac-unknown-none-elf` to clang's target spelling when generating the FFI bindings.

The `wrapper/rust` directory also has a Makefile for the common tasks. After building the C library, `make -C wrapper/rust` builds, lints, and documents the crate, and `make -C wrapper/rust test` runs the tests (a software TPM must be listening on `localhost:2321`).

25.4 Step 3: Run the Tests

The integration tests talk to a software TPM over a socket. They are behind the `swtpm-tests` feature, so a plain `cargo test` does not need a running server.

Start a server on port 2321 (from the repo root):

```
./src/fwtpm/fwtpm_server --clear --port 2321 --platform-port 2322 &
```

Run the tests against it:

```
cd wrapper/rust/wolftpm
TPM2_SWTPM_HOST=localhost TPM2_SWTPM_PORT=2321 \
    cargo test --features swtpm-tests -- --test-threads=1
```

Use `--test-threads=1` because the software TPM serves one client at a time.

Run a single test file, for example the signing tests:

```
cargo test --features swtpm-tests --test sign -- --test-threads=1
```

The default host and port are `localhost:2321`, so the environment variables can be omitted when the server is on that address.

25.5 What the Tests Check

Each file under `tests/` exercises one area against the software TPM:

File	Checks
`tests/smoke.rs`	Random bytes differ across draws, and a primary key loads.
`tests/keys.rs`	Create and load a child key, blob round-trip, short-auth blob round-trip, and an auth-protected parent.
`tests/sign.rs`	Sign a digest and verify it, and reject a tampered signature.
`tests/seal.rs`	Seal a secret and unseal it, and fail with the wrong auth.
`tests/seal_pcr.rs`	PCR-bound seal and unseal, unseal fails after a PCR changes, and an invalid PCR selection is rejected.
`tests/nv.rs`	Define an NV index, write and read it, then delete it.
`tests/pcr.rs`	Read a PCR, extend it, and confirm the value changed.
`tests/certify.rs`	An attestation key (ECC and RSA) certifies another key.
`tests/quote.rs`	Quote PCRs with an ECC and RSA AIK, and reject a bad PCR selection.
`tests/credential.rs`	MakeCredential then ActivateCredential round-trip.
`tests/ek.rs`	Create the endorsement key and export its public part.
`tests/persist.rs`	Persist a key, read it back, and evict it.
`tests/rsa.rs`	RSA-OAEP encrypt and decrypt, and an explicit OAEP-SHA1 round-trip.
`tests/hmac.rs`	Raw-key HMAC and a TPM-resident keyed-hash key HMAC.
`tests/caps.rs`	Self-test and capability query.
`tests/ecdh.rs`	ECDH generate then recover the same shared secret.
`tests/symmetric.rs`	AES-CFB encrypt and decrypt round-trip.
`tests/import.rs`	Import an external RSA and ECC private key.

25.6 Example Test Output

```
running 2 tests
test certify_with_ecc_aik ... ok
test certify_with_rsa_aik ... ok
test result: ok. 2 passed; 0 failed; 0 ignored

running 4 tests
test create_and_load_child ... ok
test key_blob_roundtrip_then_load ... ok
test key_blob_roundtrip_preserves_short_auth ... ok
test auth_protected_parent_loads_child ... ok
test result: ok. 4 passed; 0 failed; 0 ignored

running 2 tests
test sign_then_verify ... ok
test verify_rejects_tampered_signature ... ok
test result: ok. 2 passed; 0 failed; 0 ignored

running 2 tests
test rsa_oaep_roundtrip ... ok
test rsa_oaep_sha1_roundtrip ... ok
test result: ok. 2 passed; 0 failed; 0 ignored

running 1 test
test make_and_activate_credential ... ok
test result: ok. 1 passed; 0 failed; 0 ignored
```

25.7 Step 4: Run the Examples

There are two examples. The first is minimal:

```
cargo run --example create_primary
```

It opens the software TPM, reads random bytes, and creates an RSA and an ECC storage root key.

The second runs the whole API in one pass:

```
cargo run --example full_flow
```

Expected output. The random bytes and handle values differ between runs:

```
device            connected to software TPM
get_random        a91b37b1838d002453f1632ba2c0adbc
create_primary    ECC SRK handle 0x80000000
create_and_load   signing key handle 0x80000001
sign/verify       64 byte signature, verified
seal/unseal       recovered "my secret"
key blob          257 bytes, reloaded as handle 0x80000002
pcr read/extend   PCR16 000000000000.. -> debb3e7acfff..
nv define/rw      32 bytes at index 0x01500100
certify           157 byte attestation
done              all operations succeeded
```

If the wolfTPM C library was built with debug output, you will also see verbose TPM2_* traces from the library. A normal build prints only the lines above.

25.8 Using the Wrapper

```rust
use wolftpm::{Device, HashAlg, Hierarchy, KeyAlg, KeyBlob, Template};

fn main() -> Result<(), wolftpm::TpmError> {
    // Connect to a software TPM. Use Device::open() for the platform default.
    let dev = Device::open_swtpm()?;

    // Random bytes from the TPM.
    let mut nonce = [0u8; 32];
    dev.get_random(&mut nonce)?;

    // Storage root key under the owner hierarchy.
    let srk = dev.create_primary(Hierarchy::Owner, KeyAlg::EccP256, None)?;

    // Signing key under the SRK, then sign and verify a digest.
    let signer = dev.create_and_load(&srk,
     ↪  &Template::signing(KeyAlg::EccP256)?, None)?;
    let digest = [0x11u8; 32];
    let sig = signer.sign_hash(&digest)?;
    signer.verify_hash(&digest, &sig)?;

    // Seal a secret to the TPM and read it back.
    let sealed = dev.seal(&srk, b"my secret", None)?;
    let _secret = dev.unseal(sealed, &srk, None)?;

    // Persist a key as bytes, then load it again. Scope the reloaded key so it
    // releases its TPM handle before more transient objects are created below
    // (many TPMs allow only three transient objects at once).
    let bytes = {
        let blob = dev.create_key(&srk, &Template::signing(KeyAlg::EccP256)?,
         ↪  None)?;
        blob.to_bytes()?
    };
    {
        let restored = KeyBlob::from_bytes(&dev, &bytes)?;
        let _loaded = restored.load(&srk, None)?;
    }

    // Read and extend a PCR.
    let _value = dev.pcr_read(16, HashAlg::Sha256)?;
    dev.pcr_extend(16, HashAlg::Sha256, &[0xAB; 32])?;

    // Define, write, read, and delete an NV index.
    let mut slot = dev.nv_create(0x0150_0100, 32, None)?;
    dev.nv_write(&mut slot, b"metadata", 0)?;
    let mut buf = [0u8; 32];
    dev.nv_read(&mut slot, &mut buf, 0)?;
    dev.nv_delete(0x0150_0100)?;

    // Attest that a key lives in this TPM, signed by an attestation key.
    let aik = dev.create_and_load(&srk,
     ↪  &Template::attestation(KeyAlg::EccP256)?, None)?;
    let _attestation = dev.certify(&signer, &aik, &nonce)?;
```

```
    Ok(())
}
```

Keys and the device release their TPM handles automatically when they drop.

25.9 What the Crate Covers

- Device open and cleanup, TPM random numbers, self-test, and capability query.
- Primary and child keys. Templates for storage, signing, attestation, EK, RSA decrypt, keyed-hash HMAC, symmetric AES, and ECDH keys.
- Key blob serialize and load for persistence, and external RSA and ECC key import.
- Persistent key handles: store, read back, and evict.
- Sign and verify.
- RSA-OAEP encrypt and decrypt, including an explicit label hash (SHA-1 for Microsoft enrollment interop).
- Symmetric AES-CFB encrypt and decrypt.
- ECDH key agreement.
- HMAC, both raw-key and with a TPM-resident keyed-hash key.
- Seal and unseal, plain and bound to a PCR policy.
- PCR read and extend.
- NV define, write, read, delete, and certificate read (EK certificate).
- Attestation: certify a key, and quote PCRs.
- Credential activation: MakeCredential and ActivateCredential.

Recovered secrets (unseal, RSA decrypt, ECDH, AES decrypt, credential activation) are returned in a `Secret` that zeroizes its buffer on drop.

25.10 Transport Security

For confidentiality on the TPM transport, start a parameter-encryption session with `Device::start_encrypted_ses` before the secret-bearing operations:

```
let srk = dev.create_primary(Hierarchy::Owner, KeyAlg::EccP256, None)?;
let _session = dev.start_encrypted_session(&srk)?;  // salted HMAC + AES-CFB
let sealed = dev.seal(&srk, b"secret", None)?;      // command param encrypted
let plain = dev.unseal(sealed, &srk, None)?;        // response param encrypted
```

While the session is alive its salted HMAC session occupies auth slot 1, so wolfTPM encrypts the sensitive command and response parameters of seal/unseal, RSA and AES encrypt/decrypt, HMAC, NV, ECDH, and key create/load. Only one session is allowed at a time. The attestation commands (certify, quote, activate_credential) need the same auth slot and are refused while a session is active, so drop the session before calling them.

!!! warning Without a session, parameters cross the transport in the clear. Either use a session or run over a trusted local transport (the Linux kernel device or a local socket) rather than a remote `TPM2_SWTPM_HOST` or an observable physical bus.

25.11 License

GPLv3, or a commercial wolfSSL license, matching wolfTPM.

25.12 See Also

- fwTPM
- SWTPM

- Build Options
- C# Wrapper

26 C# Wrapper

The wrapper/CSharp directory contains a C# wrapper for the wolfTPM TPM 2.0 API. It binds to the native wolftpm library through P/Invoke, so the native library must be built first. The tests use NUnit and run on .NET (Windows) or Mono (Linux).

Build wolfSSL as described in the wolfTPM README.md, then build wolfTPM as described below for your platform. On Linux, tests use the swtpm TCP simulator.

26.1 Windows

A Visual Studio solution is provided for building the wrappers. To run the tests, update the .runsettings file to add the location of wolftpm.dll. The file has a placeholder for a vcpkg build, but CMake can also be used to build wolfTPM with Visual Studio.

Example CMake settings for building wolfTPM on Windows:

```
"WOLFTPM_INTERFACE": "WINAPI",
"WOLFTPM_EXAMPLES": "no",
"WOLFTPM_DEBUG": "yes",
"WITH_WOLFSSL": "C:/Users/[username]/wolfssl/out/install/windows-default"
```

26.2 Linux

The wrapper has been tested with the swtpm TCP protocol for use with the simulator. See SWTPM for building and running the simulator.

Build wolfTPM:

```
./autogen.sh
./configure --enable-swtpm
make all
make check
```

Install the prerequisites for Mono and NUnit:

```
apt install mono-tools-devel nunit nunit-console
```

Then build the wrapper and its tests, and run them:

```
cd wrapper/CSharp
mcs wolfTPM.cs wolfTPM-tests.cs
 ↪  -r:/usr/lib/cli/nunit.framework-2.6.3/nunit.framework.dll -t:library

# run the selftest case
LD_LIBRARY_PATH=../../src/.libs/ nunit-console wolfTPM.dll
 ↪  -run=tpm_csharp_test.WolfTPMTest.TrySelfTest

# run all tests
LD_LIBRARY_PATH=../../src/.libs/ nunit-console wolfTPM.dll
```

The selftest run prints output similar to the following:

```
Selected test(s): tpm_csharp_test.WolfTPMTest.TrySelfTest

wolfSSL Entering wolfCrypt_Init
.
Tests run: 1, Errors: 0, Failures: 0, Inconclusive: 0, Time: 0.1530346 seconds
```

```
  Not run: 0, Invalid: 0, Ignored: 0, Skipped: 0

wolfSSL Entering wolfCrypt_Cleanup
```

26.3 API Overview

All wrapper types are in the `wolfTPM` namespace in `wrapper/CSharp/wolfTPM.cs`. Each one is a thin class around a native wolfTPM object that is allocated and freed through P/Invoke calls into the `wolftpm` library.

Type	Purpose
`Device`	The TPM connection. Holds the native `WOLFTPM2_DEV` and exposes every TPM operation.
`Key`	A loaded TPM key, such as the storage root key (SRK) or a primary key.
`KeyBlob`	A created key (public and private parts) that can be loaded, used, and saved to a byte array.
`Template`	A TPM public template that describes the type and attributes of a new key.
`Session`	A TPM authorization session, used for HMAC sessions with parameter encryption.
`Csr`	A certificate signing request helper that holds the subject, key usage, and custom extensions.
`WolfTpm2Exception`	The exception thrown when a native call fails.
`Status`	Enum of common return codes: `TPM_RC_SUCCESS`, `TPM_RC_HANDLE`, `TPM_RC_NV_UNAVAILABLE`, `TPM_RC_SIGNATURE`, `BAD_FUNC_ARG`, and `NOT_COMPILED_IN`.

The file also defines enums that mirror native values: `TPM2_Object` (object attribute bits such as `sensitiveDataOrigin`, `userWithAuth`, `decrypt`, `sign`, `noDA`), `TPM2_Alg` (for example `RSA`, `ECC`, `SHA256`, `RSASSA`, `CFB`, `XOR`, `NULL`), `TPM2_ECC` (curves), `SE` (session type), `SESSION_mask`, `TPM_RH` (hierarchies such as `OWNER`, `ENDORSEMENT`, `PLATFORM`), and `X509_Format` (`PEM` or `DER`).

26.3.1 Device Lifetime

`Device` implements `IDisposable`. The constructor calls the native `wolfTPM2_New()`, which also initializes the TPM, so a new `Device` is ready to use. `Dispose()` calls `wolfTPM2_Free()` and clears the pointer. A finalizer calls the same cleanup if you forget, but you should wrap the device in a `using` statement or call `Dispose()` yourself.

`Key`, `KeyBlob`, `Template`, `Session`, and `Csr` follow the same pattern: the constructor allocates the native object and `Dispose()` frees it. The native return code from the free call is ignored.

`Device.Ref` returns the native device pointer. `Device` also defines these constants: `MAX_KEYBLOB_BYTES` (2048), `MAX_TPM_BUFFER` (2048), and `INVALID_DEVID` (-2). The first two are buffer sizes used by the tests and may need to be larger on your platform.

26.3.2 Errors and Return Values

Most methods return an `int` and throw `WolfTpm2Exception` on failure, so you do not need to check the result for the common case. The exception has an `ErrorCode` property with the native return code. Its `Message` includes the native function name, the code in hex, and the text from `TPM2_GetRCString`. `Device.GetErrorString(int)` and `Device.GetErrorString(Status)` give the same text for any code.

A few methods treat some codes as non-fatal and return them instead of throwing:

- `ReadPublicKey` returns `TPM_RC_HANDLE` when no object exists at the handle.
- `StoreKey` returns `TPM_RC_NV_UNAVAILABLE`.
- `VerifyHashScheme` returns `TPM_RC_SIGNATURE` when the signature does not match.
- `Csr.SetCustomExtension` returns `NOT_COMPILED_IN` when the native library was built without support for it.

Methods that produce data return a positive size on success: `KeyBlob.GetKeyBlobAsBuffer`, `Device.RsaEncrypt`, `Device.RsaDecrypt`, `Device.SignHashScheme`, `Device.GenerateCSR`, and `Csr.MakeAndSign`. `UnloadHandle` calls the native function directly and returns its code without throwing.

26.3.3 Keys, Blobs, and Sessions

- A `Key` is filled by `CreateSRK`, `CreatePrimaryKey`, `ReadPublicKey`, `LoadRsaPublicKey`, `LoadRsaPrivateKey`, or `ImportRsaPrivateKey`. `GetHandle()` returns the native handle pointer, and `SetKeyAuthPassword` sets the key password.
- A `KeyBlob` is filled by `CreateKey` using a parent `Key` and a `Template`, then loaded with `LoadKey`. `GetKeyBlobAsBuffer` exports it so it can be stored on disk and restored in another process with `SetKeyBlobFromBuffer`. After restoring, load it again with `LoadKey` and call `SetKeyAuthPassword` before using it.
- `Device.StoreKey` and `Device.DeleteKey` move a key or key blob into or out of persistent storage (NV) under a hierarchy such as `TPM_RH.OWNER`.
- Loaded TPM objects stay loaded in the TPM until you free them. Call `Device.UnloadHandle` with the `Key`, `KeyBlob`, or `Session` when you are done. `Dispose()` only frees the managed wrapper and its native memory.
- A `Session` is started with `StartAuth(device, parentKey, encDecAlg)` where `encDecAlg` is `TPM2_Alg.NULL`, `CFB`, or `XOR`. It starts an HMAC session, binds it to authorization slot 1 (or the index given to `Session(int index)`), and enables parameter encryption. End it with `StopAuth(device)`. `Device.StartSession`, `SetAuthSession`, and `ClearAuthSession` are the lower-level calls behind this.

26.3.4 Template and Csr

`Template` fills a native key template: `GetKeyTemplate_RSA`, `GetKeyTemplate_ECC`, `GetKeyTemplate_Symmetric`, the EK, SRK, and AIK variants (`GetKeyTemplate_RSA_EK`, `GetKeyTemplate_ECC_EK`, `GetKeyTemplate_RSA_SRK`, `GetKeyTemplate_ECC_SRK`, `GetKeyTemplate_RSA_AIK`, `GetKeyTemplate_ECC_AIK`), and `SetKeyTemplate_Unique`.

For a one-call certificate request, use `Device.GenerateCSR` with a subject string, a key usage string, and an `X509_Format`. For more control, build a `Csr` with `SetSubject`, `SetKeyUsage`, and `SetCustomExtension`, then call `MakeAndSign`. Set the `selfSign` argument of the extended overloads to a non-zero value to get a self-signed certificate instead of a request. The extended `Csr.MakeAndSign(..., sigType, selfSign)` overload currently throws on every successful call (it treats the returned output size as an error), so for a self-signed certificate use `Device.GenerateCSR(..., selfSignCert)` until the wrapper is fixed.

26.3.5 Other Device Methods

`SelfTest`, `GetRandom`, `RsaEncrypt`, `RsaDecrypt`, `SignHashScheme`, `VerifyHashScheme`, and `GetHandleValue` are also on `Device`. See the XML comments in `wolfTPM.cs` for each parameter.

26.3.6 Example

This example creates a storage root key, creates and loads an RSA key under it, signs a digest, and verifies the signature. It follows the pattern used in `wolfTPM-tests.cs`.

```
using System;
using wolfTPM;

class Example
{
    static void Main()
    {
        using (Device device = new Device())
        using (Key srk = new Key())
        using (KeyBlob blob = new KeyBlob())
        using (Template template = new Template())
        {
            try
            {
                device.SelfTest();
                device.CreateSRK(srk, TPM2_Alg.RSA, "StorageKeyAuth");

                template.GetKeyTemplate_RSA((ulong)(
                    TPM2_Object.sensitiveDataOrigin |
                    TPM2_Object.userWithAuth |
                    TPM2_Object.decrypt |
                    TPM2_Object.sign |
                    TPM2_Object.noDA));

                device.CreateKey(blob, srk, template, "MyKeyAuth");
                device.LoadKey(blob, srk);

                byte[] digest = new byte[32];
                device.GetRandom(digest);

                byte[] sig = new byte[256];
                int sigSz = device.SignHashScheme(blob, digest, sig,
                    TPM2_Alg.RSASSA, TPM2_Alg.SHA256);
                Console.WriteLine("Signature is {0} bytes", sigSz);

                int rc = device.VerifyHashScheme(blob, sig, digest,
                    TPM2_Alg.RSASSA, TPM2_Alg.SHA256);
                Console.WriteLine(rc == (int)Status.TPM_RC_SUCCESS ?
                    "Signature verified" : "Signature invalid");

                device.UnloadHandle(blob);
                device.UnloadHandle(srk);
            }
            catch (WolfTpm2Exception e)
            {
```

```
                Console.WriteLine("TPM error: " + e.Message);
            }
        }
    }
}
```

26.4 See Also

- SWTPM
- Build Options
- Rust Wrapper

27 STM32CubeIDE

wolfTPM is available as an STM32 Cube Pack, `I-CUBE-wolfTPM.pack`, downloadable from https://www.wolfssl.com/files/ide/I-CUBE-wolfTPM.pack. The pack has an optional but recommended dependency on the wolfCrypt library. The files live in `IDE/STM32CUBE` in the wolfTPM source tree.

!!! note This page is short and will be expanded later.

27.1 Setup

1. Set up the wolfCrypt library in your ST project by following the wolfSSL STM32Cube guide: https://github.com/wolfSSL/wolfssl/blob/master/IDE/STM32Cube/README.md. To run the wolfTPM unit tests, name the entry function `wolfTPMTest` instead of `wolfCryptDemo`.
2. Install the wolfTPM Cube Pack in the same way as the wolfSSL pack, using CubeMX.
3. Open the project `.ioc` file, click the `Software Packs` drop-down menu, then `Select Components`. Expand the `wolfTPM` pack and check all the components.
4. In the `Software Packs` configuration category of the `.ioc` file, click the wolfTPM pack and enable the library by checking the box.
5. In the `Connectivity` category, find and enable SPI for your project.
6. In the `Software Packs` configuration category, open the wolfTPM pack and set the `Enable wolfCrypt` parameter to True.
7. Save your changes and answer yes to the prompt asking about generating code.
8. Build the project and run the unit tests on the target.

27.2 Notes

Redirect `printf` to the UART so the test output is visible. See the STM32 printf changes in the wolfSSL guide.

27.3 See Also

- Building, for the bare-metal build options
- System Interfaces, for the SWTPM over UART example on STM32H5
- Embedded Integrations

COPYRIGHT ©2026 wolfSSL Inc.

28 Embedded Integrations

This page collects the platform and IDE integrations shipped with wolfTPM: Espressif ESP-IDF, Zephyr, QNX, IAR Embedded Workbench, Visual Studio, and Das U-Boot. For the STM32 Cube Pack, see STM32CubeIDE.

28.1 Espressif ESP-IDF

The Espressif project lives in `IDE/Espressif`. Wolf-specific settings for wolfTPM are in the wolfSSL `user_settings.h` file, typically found in `[project]/components/wolfssl/include`.

Build from a shell with ESP-IDF available (shown here for VisualGDB using v5.2):

```
git clone https://github.com/wolfSSL/wolfTPM.git
cd wolfTPM/IDE/Espressif

# set your path to ESP-IDF
WRK_IDF_PATH=/mnt/c/SysGCC/esp32/esp-idf/v5.2

. ${WRK_IDF_PATH}/export.sh
idf.py build
```

28.1.1 Memory

The initial minimum memory requirement is 35KB of stack. See `sdkconfig.defaults`. The memory currently assigned is 50960.

28.1.2 Pin Assignments (I2C)

The following pin assignments are used by default. You can change them in `menuconfig`.

	SDA	SCL
ESP I2C Master	I2C_MASTER_SDA	I2C_MASTER_SCL
TPM2 Device	SDA	SCL

For the default values of `I2C_MASTER_SDA` and `I2C_MASTER_SCL`, see `Example Configuration` in `menuconfig`. No external pull-up resistors are needed on SDA and SCL, because the driver enables the internal pull-ups.

28.1.3 Troubleshooting I2C

- Printing to the UART during an I2C transaction can affect timing and cause errors.
- Make sure the TPM module has been reset after a flash update.
- Check the wiring: SCL to SCL, SDA to SDA. Also connect GND. Vcc is 3.3V only.
- Make sure the proper pins are connected on the ESP32. The default SCL is `GPIO 19` and the default SDA is `GPIO 18`.
- Test with a single I2C device before testing alongside other I2C boards.
- When using multiple I2C boards, check for appropriate pull-ups. See the data sheet.
- Reset the TPM device again. Press the button on the TPM SLB9673 eval board, or set TPM pin 17 as appropriate.

COPYRIGHT ©2026 wolfSSL Inc.

28.2 Zephyr

The Zephyr port is in the `zephyr` directory of the wolfTPM source tree. It targets the Zephyr Project and provides the following:

Path	Contents
`modules/lib/wolftpm`	wolfTPM library code
`modules/lib/wolftpm/zephyr/`	Configuration and CMake files for wolfTPM as a Zephyr module
`modules/lib/wolftpm/zephyr/samples/wolftpm_wrap_caps`	wolfTPM capabilities sample application
`modules/lib/wolftpm/zephyr/samples/wolftpm_wrap_test`	wolfTPM wrapper test application

28.2.1 Set Up as a Zephyr Module

Follow the Zephyr getting started guide to set up a Zephyr project. Then add wolfTPM as a project in your `west.yml`:

```
manifest:
  remotes:
    # <your other remotes>
    - name: wolftpm
      url-base: https://github.com/wolfssl

  projects:
    # <your other projects>
    - name: wolftpm
      path: modules/lib/wolftpm
      revision: master
      remote: wolftpm
```

!!! note wolfTPM depends on wolfSSL, so also add wolfSSL to the `west.yml` in the same way.

Update west's modules:

```
west update
```

West now recognizes wolftpm as a module and includes its Kconfig and `CMakeLists.txt` in the build system.

28.2.2 Build and Run the Wrap Test

To build apps without running `west zephyr-export`, set `CMAKE_PREFIX_PATH` to the location of the Zephyr SDK and build from the `zephyr` directory. For example:

```
CMAKE_PREFIX_PATH=/path/to/zephyr-sdk-<VERSION> west build -p always -b
 ↪  qemu_x86 ../modules/lib/wolftpm/zephyr/samples/wolftpm_wrap_test/
```

Build and run `wolftpm_wrap_test`:

```
cd [zephyrproject]
west build -p auto -b qemu_x86
 ↪  modules/lib/wolftpm/zephyr/samples/wolftpm_wrap_test
west build -t run
```

28.2.3 Build and Run the Capabilities Sample

Build and run `wolftpm_wrap_caps`:

```
cd [zephyrproject]
west build -p auto -b qemu_x86
 ↪  modules/lib/wolftpm/zephyr/samples/wolftpm_wrap_caps
west build -t run
```

28.3 QNX

These steps create a QNX Momentics project that uses wolfTPM over the QNX SPI driver. The files are in `IDE/QNX`.

28.3.1 Create a QNX Application

1. Create folders for libraries (`lib`) and includes (`inc`).
2. Add the library sources to the `lib` directory as `wolfssl` and `wolftpm`.
3. Edit the Makefile to build the sources and include directories:

```
# wolfSSL and wolfTPM library includes/sources
INCLUDES += -I./inc -I./lib/wolftpm -I./lib/wolfssl
CCFLAGS_all += -DWOLFSSL_USER_SETTINGS -DWOLFTPM_USER_SETTINGS

SRCS += $(call wildcard, lib/wolfssl/wolfcrypt/src/*.c)
SRCS += $(call wildcard, lib/wolfssl/wolfcrypt/src/port/arm/*.c)
SRCS += $(call wildcard, lib/wolfssl/wolfcrypt/src/port/xilinx/*.c)
SRCS += $(call wildcard, lib/wolftpm/src/*.c)

# The QNX SPI Driver
LIBS += -lspi-master
```

4. Create `inc/user_settings.h` for all wolf-specific settings. Here is a template:

```
#ifndef WOLF_USER_SETTINGS_H
#define WOLF_USER_SETTINGS_H

/* TPM */
#define WOLFTPM_AUTODETECT
#define WOLFTPM_CHECK_WAIT_STATE
#define WOLFTPM_ADV_IO /* use advanced IO HAL callback */
#define TPM_TIMEOUT_TRIES 100000

/* always perform self-test (some chips require) */
#define WOLFTPM_PERFORM_SELFTEST

/* Reduce stack use */
#define MAX_COMMAND_SIZE    1024
#define MAX_RESPONSE_SIZE   1350
#define MAX_DIGEST_BUFFER   896

/* Debugging */
#if 1
   #define DEBUG_WOLFTPM
   //#define WOLFTPM_DEBUG_VERBOSE
   //#define WOLFTPM_DEBUG_IO
   //#define WOLFTPM_DEBUG_TIMEOUT
#endif
```

```
/* Platform */
#define WOLFCRYPT_ONLY
#define SINGLE_THREADED
#define NO_FILESYSTEM
#define WOLFSSL_IGNORE_FILE_WARN
#define WOLFSSL_HAVE_MIN
#define WOLFSSL_HAVE_MAX

/* Math */
#define ECC_TIMING_RESISTANT
#define TFM_TIMING_RESISTANT
#define USE_FAST_MATH
#define FP_MAX_BITS (2 * 4096)
#define WOLFSSL_NO_HASH_RAW
#define ALT_ECC_SIZE

/* Enables */
#define HAVE_ECC
#define ECC_SHAMIR
#define HAVE_AESGCM
#define GCM_TABLE_4BIT

/* Disables */
#define NO_MAIN_DRIVER
#define NO_WOLFSSL_MEMORY
#define NO_ASN
#define NO_ASN_TIME
#define NO_CODING
#define NO_CERTS
#define NO_PSK

#define NO_PWDBASED
#define NO_DSA
#define NO_RC4
#define NO_MD4
#define NO_MD5
#define NO_SHA
#define NO_HC128
#define NO_RABBIT
#define NO_DES3

#endif /* !WOLF_USER_SETTINGS_H */
```

5. For the wolfTPM HAL, use `tpm_io.c` directly or copy the required HAL interface into your own `.c` file. See HAL I/O Callback.
6. Add the wolfTPM example code to your own `.c` file.
7. Consider the QNX BSP SPI master patch below. It lets multiple calls run with chip select asserted, which the SPI wait states require.

28.3.2 QNX SPI Master Patch for Manual Chip Select

Edit the following QNX BSP files.

1. `bsp/src/hardware/spi/xzynq/aarch64/dll.le.zcu102/xzynq_spi.c`:

```
@@ -442,7 +442,7 @@ static void xzynq_setup(xzynq_spi_t *dev, uint32_t device)
```

```
     spi_debug1("%s: CONFIG_SPI_REG = 0x%x", __func__, dev->ctrl[id]);
 #endif

-    if(dev->fcs) {
+    if(dev->fcs || (devlist[id].cfg.mode & SPI_MODE_MAN_CS)) {
        out32(base + XZYNQ_SPI_CR_OFFSET, dev->ctrl[id] | XZYNQ_SPI_CR_MAN_CS);
     } else {
         out32(base + XZYNQ_SPI_CR_OFFSET, dev->ctrl[id]);
@@ -621,7 +621,7 @@ void *xzynq_xfer(void *hdl, uint32_t device, uint8_t *buf,
 ↪  int *len)
         reset = 1;
     }

-    if(!dev->fcs) {
+    if(!dev->fcs && !(devlist[id].cfg.mode & SPI_MODE_MAN_CS)) {
         xzynq_spi_slave_select(dev, id, 0);
     }
```

2. bsp/src/hardware/spi/xzynq/config.c:

```
@@ -72,6 +73,16 @@ int xzynq_cfg(void *hdl, spi_cfg_t *cfg, int cs)
     /* Enable ModeFail generation */
     ctrl |= XZYNQ_SPI_CR_MFAIL_EN;

+    if (cfg->mode & SPI_MODE_MAN_CS)
+        ctrl |= XZYNQ_SPI_CR_MAN_CS; /* enable manual CS mode */
+
+    if (cfg->mode & SPI_MODE_CLEAR_CS) {
+        /* make sure all chip selects are de-asserted */
+        /* set all CS bits high to de-assert */
+        out32(base + XZYNQ_SPI_CR_OFFSET,
+            in32(base + XZYNQ_SPI_CR_OFFSET) | XZYNQ_SPI_CR_CS);
+    }
+
```

3. target/qnx7/usr/include/hw/spi-master.h:

```
@@ -71,6 +71,8 @@ typedef struct {
 #define    SPI_MODE_RDY_LEVEL      (2 << 14)   /* Low level signal */
 #define    SPI_MODE_IDLE_INSERT    (1 << 16)
+#define    SPI_MODE_MAN_CS         (1 << 17)   /* Manual Chip select */
+#define    SPI_MODE_CLEAR_CS       (1 << 18)   /* Clear all chip selects (used
 ↪  with SPI_MODE_MAN_CS) */

 #define    SPI_MODE_LOCKED         (1 << 31)   /* The device is locked by
 ↪  another client */
```

For questions, email support@wolfssl.com.

28.4 IAR-EWARM

The IDE/IAR-EWARM directory holds an IAR Embedded Workbench for ARM project for the TPM 2.0 wrapper API. It has no README, so the notes below come from the project files.

Path	Contents
ewarm-tpm2.eww	IAR workspace
ewarm-tpm2.ewp	IAR project
source/main.c	Application entry point
source/tpm_main.c	TPM example code using wolftpm/tpm2.h and wolftpm/tpm2_wrap.h
header/tpm_main.h	Header for the example code

Open ewarm-tpm2.eww in IAR Embedded Workbench to build. The example uses fixed handles for the storage key (0x81000000), RSA key (0x81000010), RSA public key (0x81000011), and an NV certificate index (0x01800000).

28.4.1 IAR Project Settings

These settings come from ewarm-tpm2.ewp. The project uses the ARM toolchain and has Debug and Release configurations.

Setting	Value
Include paths	$PROJ_DIR$\..\.. (the wolfTPM root, so #include <wolftpm/tpm2.h> resolves) and $PROJ_DIR$\header
Debug preprocessor define	WOLFTPM2_NO_WOLFCRYPT
Release preprocessor define	NDEBUG
wolfTPM sources in the lib/wolftpm group	src/tpm2.c, src/tpm2_packet.c, src/tpm2_tis.c, src/tpm2_wrap.c
Application sources	source/main.c, source/tpm_main.c

The project does not add wolfSSL sources or include paths, and it does not define WOLFTPM_USER_SETTINGS. The Debug configuration builds wolfTPM without wolfCrypt through WOLFTPM2_NO_WOLFCRYPT. The Release configuration does not define it, so a Release build needs wolfSSL headers and sources added to the project. Adjust the include paths and defines to match how you build wolfSSL for your target.

!!! note The project file does not set a device or core. Select your target device in Options, General Options, Target before building. The project file was last saved by IAR EWARM 8.30.1 (build 17146). No other IAR versions are recorded in the repository, so none are listed as tested.

28.4.2 IAR HAL

The project compiles src/tpm2_tis.c, so it uses the standard TPM TIS layer with the IO callback HAL (see HAL I/O Callback). It does not include a ready-made SPI driver. source/tpm_main.c defines a stub callback, TPM2_IoCb, that returns TPM_RC_FAILURE until you replace the TODO line with a call to your own SPI transfer routine. The callback is passed to wolfTPM2_Init in TPM2_Cust_Example.

The example then reads the persistent storage key at 0x81000000. If it is missing, it creates an RSA primary storage key, makes it persistent, and does the same for the RSA key at 0x81000010. It uses the passwords ThisIsMyStorageKeyAuth and ThisIsMyKeyAuth, and it unloads both handles and calls wolfTPM2_Cleanup before it returns.

28.5 Visual Studio

The IDE/VisualStudio directory has a Visual Studio solution, wolftpm.sln, with projects for building wolfSSL, wolfTPM, and some examples: wolfssl.vcxproj, wolftpm.vcxproj,

wrap_test.vcxproj, wolfcrypt_test.vcxproj, and tls_server.vcxproj. The solution and projects are based on Visual Studio 2015 and can be retargeted to a newer version when opened.

All build settings are in IDE/VisualStudio/user_settings.h. The projects assume the wolftpm and wolfssl directories sit next to each other.

The solution supports the FIPS Ready bundle from the wolfSSL website. To use it, enable the #if 0 FIPS section in user_settings.h. See wolfssl/IDE/WIN10/README.txt in the wolfSSL source for how to set the FIPS integrity check in fips_test.c at run time.

28.5.1 Build Steps

1. Place the wolftpm and wolfssl source directories next to each other. The projects use include paths such as ../../ and ../../../wolfssl/.
2. Open IDE/VisualStudio/wolftpm.sln. The projects specify platform toolset v110, so Visual Studio asks to retarget them to the toolset you have installed.
3. Choose a configuration (Debug, Release, DLL Debug, or DLL Release) and a platform (Win32 or x64).
4. Build the solution. wolftpm references the wolfssl project, so wolfSSL builds first.

The wolfTPM CI workflow builds the solution from the command line with MSBuild and the v142 toolset, using the Debug configuration on x64:

```
msbuild /m /p:PlatformToolset=v142 /p:Platform=x64 /p:Configuration=Debug
↪ wolftpm\IDE\VisualStudio\wolftpm.sln
```

The solution contains five projects: wolfssl, wolftpm, wolfcrypt_test, wrap_test (built from examples/wrap/wrap_test.c), and tls_server.

28.5.2 Role of user_settings.h

The wolftpm project defines WOLFSSL_USER_SETTINGS and WOLFTPM_USER_SETTINGS, so both libraries read IDE/VisualStudio/user_settings.h instead of a generated options.h. The file is a template for wolfTPM with TLS 1.2 and 1.3. Settings that matter to wolfTPM include:

Define	Purpose
WOLFTPM_WINAPI	Set when _WIN32 is defined. Selects the Windows TBS transport.
WOLFSSL_AES_CFB	Required for TPM parameter encryption.
WOLFSSL_PUBLIC_MP	Exposes mp_ math functions, needed for TPM ECC secret encryption.
WOLFTPM_AUTODETECT	Supports any TPM model with safe defaults.
WOLF_CRYPTO_CB and HAVE_PK_CALLBACKS	Callbacks used to run crypto on the TPM.
WOLFSSL_CERT_GEN, WOLFSSL_CERT_REQ, WOLFSSL_CERT_EXT	Certificate and CSR generation.

The file also has a disabled #if 0 FIPS section, a math option (WOLFSSL_SP_MATH_ALL without FIPS), and a debug section with DEBUG_WOLFSSL turned on.

28.5.3 Windows TPM Transport

The Visual Studio projects use the Windows TBS (TPM Base Services) interface, not SPI or a simulator. user_settings.h defines WOLFTPM_WINAPI on Windows, wolftpm.vcxproj compiles src/tpm2_winapi.c, and the example projects (wrap_test, tls_server) and the DLL configurations of wolftpm link tbs.lib. In this mode wolfTPM calls the TBS API from tbs.h. It rejects an IO callback

or user context, so pass NULL for both to wolfTPM2_Init. See Windows TBS for NV access limits and how to run the examples.

28.6 U-Boot

wolfTPM provides experimental support for Das U-Boot, with these features:

- Uses the software SPI driver in U-Boot for TPM communication.
- Implements TPM 2.0 driver functionality through its internal TIS layer.
- Provides native API access to all TPM 2.0 commands.
- Includes the wrapper API for common TPM 2.0 operations.
- Supports two integration paths:
  - __linux__: uses the existing tpm interface through tpm2_linux.c.
  - __UBOOT__: direct SPI communication through tpm_io_uboot.c.

The example files are in examples/u-boot.

28.6.1 U-Boot Commands

These commands are available through the wolftpm interface.

Basic commands:

Command	Description
help	Show help text.
device [num device]	Show all devices or set the specified device.
info	Show information about the TPM.
state	Show internal state from the TPM, if available.
autostart	Initialize the TPM, perform a Startup(clear), and run a full selftest sequence.
init	Initialize the software stack. Must be the first command.
startup <mode> [<op>]	Issue a TPM2_Startup command. <mode> is TPM2_SU_CLEAR (reset state) or TPM2_SU_STATE (preserved state). [<op>] is an optional shutdown with “off”.
self_test <type>	Test TPM capabilities. <type> is “full” (all tests) or “continue” (untested tests only).

PCR operations:

Command	Description
pcr_extend <pcr> <digest_addr> [<digest_algo>]	Extend a PCR with a digest.
pcr_read <pcr> <digest_addr> [<digest_algo>]	Read a PCR to memory.
pcr_allocate <algorithm> <on/off> [<password>]	Reconfigure a PCR bank algorithm.
pcr_setauthpolicy or pcr_setauthvalue <pcr> <key> [<password>]	Change the PCR access key.
pcr_print	Print the current PCR state.

Security management:

Command	Description
`clear <hierarchy>`	Issue TPM2_Clear. `<hierarchy>` is TPM2_RH_LOCKOUT or TPM2_RH_PLATFORM.
`change_auth <hierarchy> <new_pw> [<old_pw>]`	Change a hierarchy password. `<hierarchy>` is TPM2_RH_LOCKOUT, TPM2_RH_ENDORSEMENT, TPM2_RH_OWNER, or TPM2_RH_PLATFORM.
`dam_reset [<password>]`	Reset the internal error counter.
`dam_parameters <max_tries> <recovery_time> <lockout_recovery> [<password>]`	Set dictionary attack mitigation (DAM) parameters.
`caps`	Show TPM capabilities and info.

Firmware management:

Command	Description
`firmware_update <manifest_addr> <manifest_sz> <firmware_addr> <firmware_sz>`	Update the TPM firmware.
`firmware_cancel`	Cancel a TPM firmware update.

28.6.2 Enable wolfTPM in U-Boot

Add these options to your board's defconfig:

```
CONFIG_TPM=y
CONFIG_TPM_V2=y
CONFIG_TPM_WOLF=y
CONFIG_CMD_WOLFTPM=y
```

Or use `make menuconfig` and enable:

- Device Drivers, TPM, TPM 2.0 Support
- Device Drivers, TPM, wolfTPM Support
- Command line interface, Security commands, Enable wolfTPM commands

28.6.3 Build and Run with QEMU

This procedure runs U-Boot with wolfTPM under QEMU, using a TPM simulator.

1. Install swtpm:

```
git clone git@github.com:stefanberger/swtpm.git
cd swtpm
./autogen.sh
make
```

2. Build U-Boot:

```
make distclean
export CROSS_COMPILE=aarch64-linux-gnu-
export ARCH=aarch64
make qemu_arm64_defconfig
make -j4
```

3. Create the TPM state directory:

```
mkdir -p /tmp/mytpm1
```

4. Start swtpm in the first terminal:

```
swtpm socket --tpm2 --tpmstate dir=/tmp/mytpm1 --ctrl
 ↪  type=unixio,path=/tmp/mytpm1/swtpm-sock --log level=20
```

5. Start QEMU in a second terminal:

```
qemu-system-aarch64 -machine virt -nographic -cpu cortex-a57 -bios u-boot.bin
 ↪  -chardev socket,id=chrtpm,path=/tmp/mytpm1/swtpm-sock -tpmdev
 ↪  emulator,id=tpm0,chardev=chrtpm -device tpm-tis-device,tpmdev=tpm0
```

6. Example boot output:

```
U-Boot 2025.07-rc1-ge15cbf232ddf-dirty (May 06 2025 - 16:25:56 -0700)

DRAM:  128 MiB
using memory 0x46658000-0x47698000 for malloc()
Core:  52 devices, 15 uclasses, devicetree: board
Flash: 64 MiB
Loading Environment from Flash... *** Warning - bad CRC, using default
   environment

In:    serial,usbkbd
Out:   serial,vidconsole
Err:   serial,vidconsole
No USB controllers found
Net:   eth0: virtio-net#32

Hit any key to stop autoboot:  0
=> tpm2 help
tpm2 - Issue a TPMv2.x command

Usage:
tpm2 <command> [<arguments>]

device [num device]
    Show all devices or set the specified device
info
    Show information about the TPM.
```

7. Example commands:

```
=> tpm2 info
tpm_tis@0 v2.0: VendorID 0x1014, DeviceID 0x0001, RevisionID 0x01 [open]
=> tpm2 startup TPM2_SU_CLEAR
=> tpm2 get_capability 0x6 0x20e 0x200 1
Capabilities read from TPM:
Property 0x6a2e45a9: 0x6c3646a9
=> tpm2 pcr_read 10 0x100000
PCR #10 sha256 32 byte content (20 known updates):
 20 25 73 0a 00 56 61 6c 75 65 3a 0a 00 23 23 20
 4f 75 74 20 6f 66 20 6d 65 6d 6f 72 79 0a 00 23
```

8. To exit QEMU, press Ctrl-A followed by X.

28.7 See Also

- STM32CubeIDE
- Building
- System Interfaces
- HAL I/O Callback
- Supported Hardware
- Windows TBS

29 API Reference

wolfTPM exposes three layers of headers. This page explains which one to start with. The function-level reference is generated from the header comments with Doxygen and appears on the pages that follow in this section.

29.1 Wrapper API

The wrapper API is declared in `wolftpm/tpm2_wrap.h`. All of its functions are named `wolfTPM2_*`. It hides command sequencing, session handling, parameter encryption and key blob management behind a small set of calls, and it is the recommended starting point for applications. Key creation, loading, signing, sealing and NV storage are covered in Key Management, and quoting and PCR use in Attestation.

29.2 Native API

The native API is declared in `wolftpm/tpm2.h`. It provides the raw `TPM2_*` commands, one function per TPM 2.0 command, along with the structures and constants from the TCG specification. Use it when you need direct control over command parameters or need a command the wrapper does not cover. You build the command arguments yourself and handle sessions and authorization explicitly.

29.3 HAL IO

`hal/tpm_io.h` declares the hardware abstraction layer used to move bytes between wolfTPM and the TPM. It is the header to read when you port wolfTPM to a new board or bus. See HAL IO Callback.

29.4 Generated Reference

The full function-level reference is generated from the header Doxygen comments and appears as the following pages in this section:

- TPM2 API
- TPM2 Wrapper API
- TPM2 Header File
- TPM2 Wrapper Header File
- TPM2 HAL IO

29.5 See Also

- Key Management
- Attestation
- HAL IO Callback

29.6 TPM2 Proprietary

More...

29.6.1 Functions

	Name
WOLFTPM_API TPM_RC	**TPM2_PCR_Allocate * out)Set which PCR banks the TPM allocates.
WOLFTPM_API TPM_RC ioCb, void * userCtx)Initializes a TPM with HAL IO callback and user supplied context. When using wolfTPM with –enable-devtpm or –enable-swtpm configuration, the ioCb and userCtx are not used.	
WOLFTPM_API TPM_RC ioCb, void * userCtx, int timeoutTries)Initializes a TPM with timeoutTries, HAL IO callback and user supplied context.	
WOLFTPM_API TPM_RC	**TPM2_Init_minimal * ctx)Initializes a TPM and sets the wolfTPM2 context that will be used. This function is typically used for rich operating systems, like Windows.
WOLFTPM_API TPM_RC	**TPM2_Cleanup * ctx)Deinitializes a TPM and wolfcrypt (if it was initialized)
WOLFTPM_API TPM_RC	**TPM2_ChipStartup * ctx, int timeoutTries)Makes sure the TPM2 startup has completed and extracts the TPM device information.
WOLFTPM_API TPM_RC ioCb, void * userCtx)Sets the user's context and IO callbacks needed for TPM communication.	
WOLFTPM_API TPM_RC	**TPM2_SetCommandRetries * ctx, int retries)Sets the number of times a command is transparently resubmitted on TPM_RC_RETRY.
WOLFTPM_API int	**TPM2_GetCommandRetries * ctx)Returns the number of times a command is transparently resubmitted on TPM_RC_RETRY.
WOLFTPM_API TPM_RC	**TPM2_SetSessionAuth * session)Sets the structure holding the TPM Authorizations.
WOLFTPM_API int	**TPM2_GetSessionAuthCount * ctx)Determine the number of currently set TPM Authorizations.
WOLFTPM_API void	**TPM2_SetActiveCtx * ctx)Sets a new TPM2 context for use.
WOLFTPM_API TPM2_CTX**(void)Provides a pointer to the TPM2 context in use.	
WOLFTPM_API int	**TPM2_GetHashDigestSize hashAlg)Determine the size in bytes of a TPM 2.0 hash digest.
WOLFTPM_API int	**TPM2_GetHashType hashAlg)Translate a TPM2 hash type to its corresponding wolfcrypt hash type.
WOLFTPM_API TPMI_ALG_HASH	**TPM2_GetTpmHashType**(int hashType)Translate a wolfCrypt hash type to TPM2 hash type.
WOLFTPM_API int	**TPM2_GetNonce**(byte * nonceBuf, int nonceSz)Generate a fresh nonce of random numbers.

	Name
WOLFTPM_API void	**TPM2_SetupPCRSel alg, int pcrIndex)Helper function to prepare a correct PCR selection For example, when preparing to create a TPM2_Quote.
WOLFTPM_API void	**TPM2_SetupPCRSelArray alg, byte * pcrArray, word32 pcrArraySz)Helper function to prepare a correct PCR selection with multiple indices For example, when preparing to create a TPM2_Quote.
WOLFTPM_API int	**TPM2_IsAlgSupported alg, int * isSupported)Report whether the TPM implements a given algorithm.
WOLFTPM_API int	**TPM2_IsPcrBankAllocated hashAlg, int pcrIndex, int * isAllocated)Report whether a PCR index is allocated in a bank of a given hash.
WOLFTPM_API const char *	**TPM2_GetRCString**(int rc)Get a human readable string for any TPM 2.0 return code.
WOLFTPM_API const char *	**TPM2_GetAlgName alg)Get a human readable string for any TPM 2.0 algorithm.
WOLFTPM_API TPM_ALG_ID	**TPM2_GetAlgId**(const char * name)Translates a TPM algorithm name to its TPM algorithm ID.
WOLFTPM_API const char *	**TPM2_GetHierarchyDesc authHandle)Get readable string for TPM 2.0 hierarchy.
WOLFTPM_API int	**TPM2_GetCurveSize curveID)Determine the size in bytes of any TPM ECC Curve.
WOLFTPM_API TPM_ALG_ID curveID)Determine the recommended message digest for a TPM ECC Curve.	
WOLFTPM_API int	**TPM2_GetTpmCurve**(int curveID)Translate a wolfcrypt curve type to its corresponding TPM curve type.
WOLFTPM_API int	**TPM2_GetWolfCurve**(int curve_id)Translate a TPM curve type to its corresponding wolfcrypt curve type.
WOLFTPM_API int	**TPM2_ParseAttest structure.
WOLFTPM_API int	**TPM2_HashNvPublic * nvPublic, byte * buffer, UINT16 * size)Computes fresh NV Index name based on a nvPublic structure.
WOLFTPM_API int	**TPM2_AppendPublic structure based on a user provided buffer.
WOLFTPM_API int	**TPM2_ParsePublic structure and stores in a user provided buffer.
WOLFTPM_LOCAL int	**TPM2_GetName * name)Provides the Name of a TPM object.
WOLFTPM_API UINT16	**TPM2_GetVendorID**(void)Provides the vendorID of the active TPM2 context.

	Name
WOLFTPM_LOCAL void	**TPM2_ForceZero**(void * mem, word32 len)Forcefully zero a memory region. Unlike memset, this call is not subject to dead-store elimination, so it is safe to use for erasing sensitive material (keys, intermediate HMAC state, etc.) just before the memory goes out of scope.
WOLFTPM_TEST_API int	**TPM2_ConstantCompare**(const byte * a, const byte * b, word32 len)Constant-time memory comparison. Returns 0 on byte-for-byte match and non-zero on mismatch. Timing is independent of where the first differing byte is located, which is required when comparing HMAC digests or authentication values.
WOLFTPM_API void	**TPM2_PrintBin**(const byte * buffer, word32 length)Helper function to print a binary buffer in a formatted way.
WOLFTPM_API void	**TPM2_PrintAuth type in a human readable way.
WOLFTPM_API void	**TPM2_PrintPublicArea type in a human readable way.

29.6.2 Detailed Description

This module describes TPM2 commands specific only to wolfTPM.

Typically, these commands include helpers for handling TPM 2.0 data structures.

There are also functions to help debugging and testing during development.

29.6.3 Functions Documentation

```
WOLFTPM_API TPM_RC TPM2_PCR_Allocate(
    PCR_Allocate_In * in,
    PCR_Allocate_Out * out
)
```

Set which PCR banks the TPM allocates.

Parameters:

- **in** pointer to a PCR_Allocate_In struct
- **out** pointer to a PCR_Allocate_Out struct

See:

- wolfTPM2_AllocatePCRBanks
- TPM2_IsPcrBankAllocated

Return:

- TPM_RC_SUCCESS: the TPM processed the request - check out->allocationSuccess, which is NO when the TPM lacks the space
- TPM_RC_HIERARCHY: the platform hierarchy is disabled
- BAD_FUNC_ARG: check the provided arguments, or no session is set

Note:

- The selection REPLACES the current allocation - banks not named in it are deallocated. Algorithms the TPM does not implement are silently ignored (TPM 2.0 Part 3 22.5), so a selection naming only unimplemented algorithms can leave the TPM with no PCR banks at all. Prefer wolfTPM2_AllocatePCRBanks, which pre-checks each algorithm and refuses an empty result.
- The change takes effect at the next Startup(CLEAR), not on return.
- Requires an active session (TPM2_SetAuthPassword or wolfTPM2_SetAuthSession on slot 0) and the platform hierarchy; without one this returns BAD_FUNC_ARG rather than a TPM response code.

```
WOLFTPM_API TPM_RC TPM2_Init(
    TPM2_CTX * ctx,
    TPM2HalIoCb ioCb,
    void * userCtx
)
```

Initializes a TPM with HAL IO callback and user supplied context. When using wolfTPM with –enable-devtpm or –enable-swtpm configuration, the ioCb and userCtx are not used.

Parameters:

- **ctx** pointer to a TPM2_CTX struct
- **ioCb** pointer to TPM2HalIoCb (HAL IO) callback function
- **userCtx** pointer to the user's context that will be stored as a member of the ctx struct

See:

- TPM2_Startup
- TPM2_GetRCString
- TPM2_Init_minimal
- TPM2_Init_ex
- wolfTPM2_Init

Return:

- TPM_RC_SUCCESS: successful
- TPM_RC_FAILURE: general error (possibly IO)
- BAD_FUNC_ARG check arguments provided

Note: TPM2_Init_minimal() with both ioCb and userCtx set to NULL. In other modes, the ioCb shall be set in order to use TIS. Example ioCB for baremetal and RTOS applications are provided in hal/tpm_io.c

Example

```
int rc;
TPM2_CTX tpm2Ctx;

rc = TPM2_Init(&tpm2Ctx, TPM2_IoCb, userCtx);
if (rc != TPM_RC_SUCCESS) {
    // TPM2_Init failed
}
```

```
WOLFTPM_API TPM_RC TPM2_Init_ex(
    TPM2_CTX * ctx,
    TPM2HalIoCb ioCb,
    void * userCtx,
    int timeoutTries
)
```

Initializes a TPM with timeoutTries, HAL IO callback and user supplied context.

Parameters:

- **ctx** pointer to a TPM2_CTX struct
- **ioCb** pointer to TPM2HalIoCb (HAL IO) callback function
- **userCtx** pointer to the user's context that will be stored as a member of the ctx struct
- **timeoutTries** specifies the number of attempts to confirm that TPM2 startup has completed

See:

- TPM2_GetRCString
- TPM2_Init_minimal
- TPM2_Init
- wolfTPM2_Init_ex

Return:

- TPM_RC_SUCCESS: successful
- TPM_RC_FAILURE: general error (possibly IO)
- BAD_FUNC_ARG check arguments provided

Note: It is recommended to use TPM2_Init instead of using TPM2_Init_ex directly.

```
WOLFTPM_API TPM_RC TPM2_Init_minimal(
    TPM2_CTX * ctx
)
```

Initializes a TPM and sets the wolfTPM2 context that will be used. This function is typically used for rich operating systems, like Windows.

Parameters:

- **ctx** pointer to a TPM2_CTX struct

See:

- TPM2_GetRCString
- TPM2_Init

Return:

- TPM_RC_SUCCESS: successful
- TPM_RC_FAILURE: general error (possibly IO)
- BAD_FUNC_ARG check arguments provided

Note: It is recommended to use TPM2_Init instead of using TPM2_Init_minimal directly.

```
WOLFTPM_API TPM_RC TPM2_Cleanup(
    TPM2_CTX * ctx
)
```

Deinitializes a TPM and wolfcrypt (if it was initialized)

Parameters:

- **ctx** pointer to a TPM2_CTX struct

See:

- TPM2_GetRCString
- TPM2_Init
- wolfTPM2_Cleanup

Return:

- TPM_RC_SUCCESS: successful
- TPM_RC_FAILURE: could not acquire the lock on the wolfTPM2 context
- BAD_FUNC_ARG: the TPM2 device structure is a NULL pointer

Example

```
int rc;
TPM2_CTX tpm2Ctx;

rc = TPM2_Cleanup(&tpm2Ctx->dev);
if (rc != TPM_RC_SUCCESS) {
    // TPM2_Cleanup failed
}
```

```
WOLFTPM_API TPM_RC TPM2_ChipStartup(
    TPM2_CTX * ctx,
    int timeoutTries
)
```

Makes sure the TPM2 startup has completed and extracts the TPM device information.

Parameters:

- **ctx** pointer to a TPM2_CTX struct
- **timeoutTries** specifies the number of attempts to check if TPM2 startup has completed

See:

- TPM2_GetRCString
- TPM2_TIS_StartupWait
- TPM2_TIS_RequestLocality
- TPM2_TIS_GetInfo
- TPM2_Init_ex

Return:

- TPM_RC_SUCCESS: successful
- TPM_RC_FAILURE: general error (possibly IO)
- BAD_FUNC_ARG: check the provided arguments
- TPM_RC_TIMEOUT: timeout occurred

Note: This function is used in TPM2_Init_ex

```
WOLFTPM_API TPM_RC TPM2_SetHalIoCb(
    TPM2_CTX * ctx,
    TPM2HalIoCb ioCb,
    void * userCtx
)
```

Sets the user's context and IO callbacks needed for TPM communication.

Parameters:

- **ctx** pointer to a TPM2_CTX struct
- **ioCb** pointer to TPM2HalIoCb (HAL IO) callback function
- **userCtx** pointer to the user's context that will be stored as a member of the ctx struct

See:

- TPM2_GetRCString
- TPM2_Init
- wolfTPM2_Init

Return:

- TPM_RC_SUCCESS: successful
- TPM_RC_FAILURE: could not acquire the lock on the wolfTPM2 context
- BAD_FUNC_ARG: the TPM2 device structure is a NULL pointer

Note: SetHalIoCb will fail if built with devtpm or swtpm as the callback is not used for TPM. For other configuration builds, ioCb must be set to a non-NULL function pointer and userCtx is optional.

Typically, TPM2_Init or wolfTPM2_Init are used to set the HAL IO.

```
WOLFTPM_API TPM_RC TPM2_SetCommandRetries(
    TPM2_CTX * ctx,
    int retries
)
```

Sets the number of times a command is transparently resubmitted on TPM_RC_RETRY.

Parameters:

- **ctx** pointer to a TPM2_CTX struct
- **retries** number of additional resubmit attempts on TPM_RC_RETRY (0 disables)

See: TPM2_GetCommandRetries

Return:

- TPM_RC_SUCCESS: successful
- BAD_FUNC_ARG: the TPM2 context is NULL or retries is negative

The TPM returns TPM_RC_RETRY when momentarily busy (for example persisting the daUsed flag on first auth use of a non-noDA key). Disabled by default (WOLFTPM_MAX_RETRIES is 0); pass a count > 0 to enable transparent resubmit. A count of 0 returns TPM_RC_RETRY to the caller.

```
WOLFTPM_API int TPM2_GetCommandRetries(
    TPM2_CTX * ctx
)
```

Returns the number of times a command is transparently resubmitted on TPM_RC_RETRY.

Parameters:

- **ctx** pointer to a TPM2_CTX struct

See: TPM2_SetCommandRetries

Return:

- the configured retry count on success
- BAD_FUNC_ARG: the TPM2 context is a NULL pointer

Note: Callers must synchronize concurrent calls to this function and TPM2_SetCommandRetries.

```
WOLFTPM_API TPM_RC TPM2_SetSessionAuth(
    TPM2_AUTH_SESSION * session
)
```

Sets the structure holding the TPM Authorizations.

Parameters:

- **session** pointer to an array of type TPM2_AUTH_SESSION

See:

- TPM2_GetRCString
- TPM2_Init
- wolfTPM2_Init

Return:

- TPM_RC_SUCCESS: successful
- TPM_RC_FAILURE: could not acquire the lock on the wolfTPM2 context
- BAD_FUNC_ARG: the TPM2 context structure is a NULL pointer

Rarely used, because TPM2_Init functions and wolfTPM2_Init perform this initialization as well TPM 2.0 Commands can have up to three authorization slots, therefore it is recommended to supply an array of size MAX_SESSION_NUM to TPM2_SetSessionAuth(see example below).

Example

```
int rc;
TPM2_AUTH_SESSION session[MAX_SESSION_NUM];

XMEMSET(session, 0, sizeof(session));
session[0].sessionHandle = TPM_RS_PW;

rc = TPM2_SetSessionAuth(session);
if (rc != TPM_RC_SUCCESS) {
    // TPM2_SetSessionAuth failed
}
```

```
WOLFTPM_API int TPM2_GetSessionAuthCount(
    TPM2_CTX * ctx
)
```

Determine the number of currently set TPM Authorizations.

Parameters:

- **ctx** pointer to a TPM2_CTX struct

See:

- TPM2_CTX
- TPM2_AUTH_SESSION

Return:

- the number of active TPM Authorizations (between one and three)
- BAD_FUNC_ARG: check the arguments provided for a NULL pointer

Example

```
int authCount;
TPM2_CTX tpm2Ctx;

authCount = TPM2_GetSessionAuthCount(&tpm2Ctx);
if (authCount == BAD_FUNC_ARG) {
    // TPM2_GetSessionAuthCount failed
}
```

```
WOLFTPM_API void TPM2_SetActiveCtx(
    TPM2_CTX * ctx
)
```

Sets a new TPM2 context for use.

Parameters:

- **ctx** pointer to a TPM2_CTX struct

See:

- TPM2_CTX
- TPM2_AUTH_SESSION

Example

```
TPM2_CTX tpm2Ctx;

TPM2_SetActiveCtx(&tpm2Ctx);
```

```
WOLFTPM_API TPM2_CTX * TPM2_GetActiveCtx(
    void
)
```

Provides a pointer to the TPM2 context in use.

See:

- TPM2_CTX
- TPM2_AUTH_SESSION

Return: ctx pointer to a TPM2_CTX struct

Example

```
TPM2_CTX *tpm2Ctx;

tpm2Ctx = TPM2_GetActiveCtx();
```

```
WOLFTPM_API int TPM2_GetHashDigestSize(
    TPMI_ALG_HASH hashAlg
)
```

Determine the size in bytes of a TPM 2.0 hash digest.

Parameters:

- **hashAlg** a valid TPM 2.0 hash type

Return:

- the size of a TPM 2.0 hash digest as number of bytes
- 0 if hash type is invalid

Example

```
int digestSize = 0;
TPMI_ALG_HASH hashAlg = TPM_ALG_SHA256;

digestSize = TPM2_GetHashDigestSize(hashAlg);
if (digestSize > 0) {
    //digestSize contains a valid value
}
```

```
WOLFTPM_API int TPM2_GetHashType(
    TPMI_ALG_HASH hashAlg
)
```

Translate a TPM2 hash type to its corresponding wolfcrypt hash type.

Parameters:

- **hashAlg** a valid TPM 2.0 hash type

Return:

- a value specifying a hash type to use with wolfcrypt
- 0 if hash type is invalid

Example

```
int wc_hashType;
TPMI_ALG_HASH hashAlg = TPM_ALG_SHA256;

wc_hashType = TPM2_GetHashDigestSize(hashAlg);
if (wc_hashType > 0) {
    //wc_hashType contains a valid wolfcrypt hash type
}
```

```
WOLFTPM_API TPMI_ALG_HASH TPM2_GetTpmHashType(
    int hashType
)
```

Translate a wolfCrypt hash type to TPM2 hash type.

Parameters:

- **hashType** a wolfCrypt hash type

Return:

- a TPM2 hash type (TPM_ALG_*)
- TPM_ALG_ERROR when wolfCrypt hash type is invalid or not found

Example

```
int wc_hashType = WC_HASH_TYPE_SHA256;
TPMI_ALG_HASH hashAlg;

hashAlg = TPM2_GetHashDigestSize(wc_hashType);
if (hashAlg != TPM_ALG_ERROR) {
    //hashAlg contains a valid TPM2 hash type
}
```

```
WOLFTPM_API int TPM2_GetNonce(
    byte * nonceBuf,
    int nonceSz
)
```

Generate a fresh nonce of random numbers.

Parameters:

- **nonceBuf** pointer to a BYTE buffer

- **nonceSz** size of the nonce in bytes

Return:

- TPM_RC_SUCCESS: successful
- TPM_RC_FAILURE: generic failure (TPM IO issue or wolfcrypt configuration)
- BAD_FUNC_ARG: check the provided arguments

Note: Can use the TPM random number generator if WOLFTPM2_USE_HW_RNG is defined. WARNING: Using WOLFTPM2_USE_HW_RNG exposes nonces to physical bus tampering (SPI/I2C). An attacker with bus access can force a known nonce, which can significantly weaken session security, especially for unsalted sessions. Use the default local wolfCrypt RNG unless the bus is physically secure.

Example

```
int rc, nonceSize = 32;
BYTE freshNonce[32];

rc = TPM2_GetNonce(freshNonce, nonceSize);
if (rc != TPM_RC_SUCCESS) {
    //TPM2_GetNonce failed
}
```

```
WOLFTPM_API void TPM2_SetupPCRSel(
    TPML_PCR_SELECTION * pcr,
    TPM_ALG_ID alg,
    int pcrIndex
)
```

Helper function to prepare a correct PCR selection For example, when preparing to create a TPM2_Quote.

Parameters:

- **pcr** pointer to a structure of type TPML_PCR_SELECTION. Note: Caller must zeroize/memset(0)
- **alg** value of type TPM_ALG_ID specifying the type of hash algorithm used
- **pcrIndex** value between 0 and 23 specifying the PCR register for use

See:

- TPM2_PCR_Read
- TPM2_PCR_Extend
- TPM2_PCR_Reset
- TPM2_Quote

Example

```
int pcrIndex = 16; // This is a PCR register for DEBUG & testing purposes
PCR_Read_In pcrRead;
```

```
XMEMSET(&pcrRead, 0, sizeof(pcrRead));
TPM2_SetupPCRSel(&pcrRead.pcrSelectionIn, TPM_ALG_SHA256, pcrIndex);
```

```
WOLFTPM_API void TPM2_SetupPCRSelArray(
    TPML_PCR_SELECTION * pcr,
    TPM_ALG_ID alg,
    byte * pcrArray,
    word32 pcrArraySz
)
```

Helper function to prepare a correct PCR selection with multiple indices For example, when preparing to create a TPM2_Quote.

Parameters:

- **pcr** pointer to a structure of type TPML_PCR_SELECTION. Note: Caller must zeroize/memset(0)
- **alg** value of type TPM_ALG_ID specifying the type of hash algorithm used
- **pcrArray** array of values between 0 and 23 specifying the PCR register for use
- **pcrArraySz** length of the pcrArray

See:

- TPM2_PCR_Read
- TPM2_PCR_Extend
- TPM2_PCR_Reset
- TPM2_Quote

Example

```
PCR_Read_In pcrRead;
byte   pcrArray[PCR_SELECT_MAX];
word32 pcrArraySz = 0;

XMEMSET(&pcrRead, 0, sizeof(pcrRead));
XMEMSET(pcrArray, 0, sizeof(pcrArray));
pcrArray[pcrArraySz++] = 16; // This is a PCR register for DEBUG & testing
 ↪  purposes

TPM2_SetupPCRSelArray(&pcrRead.pcrSelectionIn, TPM_ALG_SHA256, pcrArray,
 ↪  pcrArraySz);
```

```
WOLFTPM_API int TPM2_IsAlgSupported(
    TPM_ALG_ID alg,
    int * isSupported
)
```

Report whether the TPM implements a given algorithm.

Parameters:

- **alg** the algorithm identifier to test (for example TPM_ALG_SHA512)
- **isSupported** output, set to 1 if implemented by the TPM, else 0

See: TPM2_IsPcrBankAllocated

Return:

- TPM_RC_SUCCESS: query completed; *isSupported is 1 or 0
- BAD_FUNC_ARG: isSupported is NULL

Note: Queries TPM_CAP_ALGS. Fails closed: *isSupported is 0 on any error, so a query failure cannot be mistaken for “supported”.

```
WOLFTPM_API int TPM2_IsPcrBankAllocated(
    TPM_ALG_ID hashAlg,
    int pcrIndex,
    int * isAllocated
)
```

Report whether a PCR index is allocated in a bank of a given hash.

Parameters:

- **hashAlg** the PCR bank hash algorithm (for example TPM_ALG_SHA256)
- **pcrIndex** the PCR index to test
- **isAllocated** output, set to 1 if allocated in that bank, else 0

See:

- TPM2_SetupPCRSel
- TPM2_IsAlgSupported

Return:

- TPM_RC_SUCCESS: query completed; *isAllocated is 1 or 0
- BAD_FUNC_ARG: isAllocated is NULL or pcrIndex is negative

Note: Queries TPM_CAP_PCRS. Implementing a hash and allocating a bank for it are separate: a TPM may offer SHA-1 while allocating no SHA-1 bank, and a selection naming an unallocated bank is rejected. Ask before building a selection with TPM2_SetupPCRSel(). Fails closed: isAllocated is 0 on any error.

```
WOLFTPM_API const char * TPM2_GetRCString(
    int rc
)
```

Get a human readable string for any TPM 2.0 return code.

Parameters:

- **rc** integer value representing a TPM return code

Return: pointer to a string constant

Example

```
int rc;

rc = wolfTPM2_Init(&dev, TPM2_IoCb, userCtx);
if (rc != TPM_RC_SUCCESS) {
    printf("wolfTPM2_Init failed 0x%x: %s\n", rc, TPM2_GetRCString(rc));
    return rc;
}
```

```
WOLFTPM_API const char * TPM2_GetAlgName(
    TPM_ALG_ID alg
)
```

Get a human readable string for any TPM 2.0 algorithm.

Parameters:

- **alg** value of type TPM_ALG_ID specifying a valid TPM 2.0 algorithm

Return: pointer to a string constant

Example

```
int paramEncAlg = TPM_ALG_CFB;

printf("\tUse Parameter Encryption: %s\n", TPM2_GetAlgName(paramEncAlg));
```

```
WOLFTPM_API TPM_ALG_ID TPM2_GetAlgId(
    const char * name
)
```

Translates a TPM algorithm name to its TPM algorithm ID.

Parameters:

- **name** a pointer to a string constant specifying a valid TPM algorithm name

Return:

- a TPM algorithm ID
- TPM_ALG_ERROR if invalid algorithm name

Example

```
TPM_ALG_ID alg = TPM2_GetAlgId("SHA256");
if (alg == TPM_ALG_ERROR) {
    //TPM2_GetAlgId failed
}
```

```
WOLFTPM_API const char * TPM2_GetHierarchyDesc(
    TPMI_RH_HIERARCHY_AUTH authHandle
)
```

Get readable string for TPM 2.0 hierarchy.

Parameters:

- **authHandle** value of type TPMI_RH_HIERARCHY_AUTH specifying a valid TPM 2.0 hierarchy

Return: pointer to a string constant

Example

```
TPMI_RH_HIERARCHY_AUTH authHandle = TPM_RH_OWNER;

printf("\tHierarchy: %s\n", TPM2_GetHierarchyDesc(authHandle));
```

```
WOLFTPM_API int TPM2_GetCurveSize(
    TPM_ECC_CURVE curveID
)
```

Determine the size in bytes of any TPM ECC Curve.

Parameters:

- **curveID** value of type TPM_ECC_CURVE

Return:

- 0 in case of invalid curve type
- integer value representing the number of bytes

Example

```
int bytes;
TPM_ECC_CURVE curve = TPM_ECC_NIST_P256;

bytes = TPM2_GetCurveSize(curve);
if (bytes == 0) {
    //TPM2_GetCurveSize failed
}
```

```
WOLFTPM_API TPM_ALG_ID TPM2_GetCurveHashAlg(
    TPM_ECC_CURVE curveID
)
```

Determine the recommended message digest for a TPM ECC Curve.

Parameters:

- **curveID** value of type TPM_ECC_CURVE

See: TPM2_GetCurveSize

Return: TPM_ALG_ID of the hash paired to the curve size (P256->SHA256, P384->SHA384, P521->SHA512). Defaults to TPM_ALG_SHA256.

```
WOLFTPM_API int TPM2_GetTpmCurve(
    int curveID
)
```

Translate a wolfcrypt curve type to its corresponding TPM curve type.

Parameters:

- **curveID** pointer to a BYTE buffer

See: TPM2_GetWolfCurve

Return:

- integer value representing a wolfcrypt curve type
- ECC_CURVE_OID_E in case of invalid curve type

Example

```
int tpmCurve;
int wc_curve = ECC_SECP256R1;

tpmCurve = TPM2_GetTpmCurve(curve);
\in this case tpmCurve will be TPM_ECC_NIST_P256
if (tpmCurve = ECC_CURVE_OID_E) {
    //TPM2_GetTpmCurve failed
}
```

```
WOLFTPM_API int TPM2_GetWolfCurve(
    int curve_id
)
```

Translate a TPM curve type to its corresponding wolfcrypt curve type.

Parameters:

- **curve_id** pointer to a BYTE buffer

See: TPM2_GetTpmCurve

Return:

- integer value representing a TPM curve type
- -1 or ECC_CURVE_OID_E in case of invalid curve type

Example

```
int tpmCurve = TPM_ECC_NIST_P256;
int wc_curve;

wc_curve = TPM2_GetWolfCurve(tpmCurve);
\in this case tpmCurve will be ECC_SECP256R1
if (wc_curve = ECC_CURVE_OID_E || wc_curve == -1) {
    //TPM2_GetWolfCurve failed
}
```

```
WOLFTPM_API int TPM2_ParseAttest(
    const TPM2B_ATTEST * in,
    TPMS_ATTEST * out
)
```

Parses TPM2B_ATTEST structure.

Parameters:

- **in** pointer to a structure of a TPM2B_ATTEST type
- **out** pointer to a structure of a TPMS_ATTEST type

Return:

- TPM_RC_SUCCESS: successful
- BAD_FUNC_ARG: check the provided arguments
- TPM_RC_VALUE: invalid attestation magic or type
- TPM_RC_SIZE: malformed or truncated attestation data

Note: This is public API of the helper function TPM2_Packet_ParseAttest

Example

```
TPM2B_ATTEST in; //for example, as part of a TPM2_Quote
TPMS_ATTEST out

rc = TPM2_GetNonce(&in, &out);
if (rc != TPM_RC_SUCCESS) {
```

```
    //TPM2_ParseAttest failed
}
```

```
WOLFTPM_API int TPM2_HashNvPublic(
    TPMS_NV_PUBLIC * nvPublic,
    byte * buffer,
    UINT16 * size
)
```

Computes fresh NV Index name based on a nvPublic structure.

Parameters:

- **nvPublic**
- **buffer** pointer to a structure of a TPMS_ATTEST type
- **size** pointer to a variable of UINT16 type to store the size of the nvIndex

Return:

- TPM_RC_SUCCESS: successful
- negative integer value in case of an error
- BAD_FUNC_ARG: check the provided arguments
- NOT_COMPILED_IN: check if wolfcrypt is enabled

Example

```
TPMS_NV_PUBLIC nvPublic;
BYTE buffer[TPM_MAX_DIGEST_SIZE];
UINT16 size;

rc = TPM2_HashNvPublic(&nvPublic, &buffer, &size);
if (rc != TPM_RC_SUCCESS) {
    //TPM2_HashNvPublic failed
}
```

```
WOLFTPM_API int TPM2_AppendPublic(
    byte * buf,
    word32 size,
    int * sizeUsed,
    TPM2B_PUBLIC * pub
)
```

Populates TPM2B_PUBLIC structure based on a user provided buffer.

Parameters:

- **buf** pointer to a user buffer
- **size** integer value of word32 type, specifying the size of the user buffer

- **sizeUsed** pointer to an integer variable, stores the used size of pub->buffer
- **pub** pointer to an empty structure of TPM2B_PUBLIC type

See: TPM2_ParsePublic

Return:

- TPM_RC_SUCCESS: successful
- TPM_RC_FAILURE: insufficient buffer size
- BAD_FUNC_ARG: check the provided arguments

Note: Public API of the helper function TPM2_Packet_AppendPublic

Example

```
TPM2B_PUBLIC pub; //empty
int sizeUsed, rc;
BYTE buffer[sizeof(TPM2B_PUBLIC)];
word32 size = sizeof(buffer);

rc = TPM2_AppendPublic(&buffer, size, &sizeUsed, &pub);
if (rc != TPM_RC_SUCCESS) {
    //TPM2_AppendPublic failed
}
```

```
WOLFTPM_API int TPM2_ParsePublic(
    TPM2B_PUBLIC * pub,
    byte * buf,
    word32 size,
    int * sizeUsed
)
```

Parses TPM2B_PUBLIC structure and stores in a user provided buffer.

Parameters:

- **pub** pointer to a populated structure of TPM2B_PUBLIC type
- **buf** pointer to an empty user buffer
- **size** integer value of word32 type, specifying the available size of the user buffer
- **sizeUsed** pointer to an integer variable, stores the used size of the user buffer

See: TPM2_AppendPublic

Return:

- TPM_RC_SUCCESS: successful
- TPM_RC_FAILURE: insufficient buffer size
- BAD_FUNC_ARG: check the provided arguments

Note: Public API of the helper function TPM2_Packet_ParsePublic

Example

```
TPM2B_PUBLIC pub; //populated
int sizeUsed, rc;
BYTE buffer[sizeof(TPM2B_PUBLIC)];
word32 size = sizeof(buffer);

rc = TPM2_ParsePublic(&pub, buffer, size, &sizeUsed);
if (rc != TPM_RC_SUCCESS) {
    //TPM2_ParsePublic failed
}
```

```
WOLFTPM_LOCAL int TPM2_GetName(
    TPM2_CTX * ctx,
    UINT32 handleValue,
    int handleCnt,
    int idx,
    TPM2B_NAME * name
)
```

Provides the Name of a TPM object.

Parameters:

- **ctx** pointer to a TPM2 context
- **handleValue** value of UINT32 type, specifying a valid TPM handle
- **handleCnt** total number of handles used in the current TPM command/session
- **idx** index value, between one and three, specifying a valid TPM Authorization session
- **name** pointer to an empty structure of TPM2B_NAME type

Return:

- TPM_RC_SUCCESS: successful
- BAD_FUNC_ARG: check the provided arguments

Note: The object is reference by its TPM handle and session index

Example

```
int rc;
UINT32 handleValue = TRANSIENT_FIRST;
handleCount = 1;
sessionIdx = 0;
TPM2B_NAME name;

rc = TPM2_GetName(ctx, handleValue, handleCount, sessionIdx, &name);
if (rc != TPM_RC_SUCCESS) {
    //TPM2_GetName failed
}
```

```
WOLFTPM_API UINT16 TPM2_GetVendorID(
    void
)
```

Provides the vendorID of the active TPM2 context.

See:

- TPM2_GetCapabilities
- TPM2_Init

Return:

- integer value of UINT16 type, specifying the vendor ID
- 0 if TPM2 context is invalid or NULL

Note: Depends on correctly read TPM device info during TPM Init

Example

```
TPM2_CTX *tpm2Ctx;

tpm2Ctx = TPM2_GetActiveCtx();
```

```
WOLFTPM_LOCAL void TPM2_ForceZero(
    void * mem,
    word32 len
)
```

Forcefully zero a memory region. Unlike memset, this call is not subject to dead-store elimination, so it is safe to use for erasing sensitive material (keys, intermediate HMAC state, etc.) just before the memory goes out of scope.

Parameters:

- **mem** pointer to the buffer to zero (may be NULL; call is a no-op)
- **len** number of bytes to zero

See: TPM2_ConstantCompare

```
WOLFTPM_TEST_API int TPM2_ConstantCompare(
    const byte * a,
    const byte * b,
    word32 len
)
```

Constant-time memory comparison. Returns 0 on byte-for-byte match and non-zero on mismatch. Timing is independent of where the first differing byte is located, which is required when comparing HMAC digests or authentication values.

Parameters:

- **a** first buffer
- **b** second buffer
- **len** number of bytes to compare

Return:

- 0 if len bytes of a and b are identical
- non-zero on any mismatch

```
WOLFTPM_API void TPM2_PrintBin(
    const byte * buffer,
    word32 length
)
```

Helper function to print a binary buffer in a formatted way.

Parameters:

- **buffer** pointer to a buffer of BYTE type
- **length** integer value of word32 type, containing the size of the buffer

See:

- TPM2_PrintAuth
- TPM2_PrintPublicArea

Note: Requires DEBUG_WOLFTPM to be defined

Example

```
BYTE buffer[] = {0x01,0x02,0x03,0x04};
word32 length = sizeof(buffer);

TPM2_PrintBin(buffer, length);
```

```
WOLFTPM_API void TPM2_PrintAuth(
    const TPMS_AUTH_COMMAND * authCmd
)
```

Helper function to print a structure of TPMS_AUTH_COMMAND type in a human readable way.

Parameters:

- **authCmd** pointer to a populated structure of TPMS_AUTH_COMMAND type

See:

- TPM2_PrintBin
- TPM2_PrintPublicArea

Note: Requires DEBUG_WOLFTPM to be defined

Example

```
TPMS_AUTH_COMMAND authCmd; //for example, part of a TPM Authorization session

TPM2_PrintAuth(&authCmd);
```

```
WOLFTPM_API void TPM2_PrintPublicArea(
    const TPM2B_PUBLIC * pub
)
```

Helper function to print a structure of TPM2B_PUBLIC type in a human readable way.

Parameters:

- **pub** pointer to a populated structure of TPM2B_PUBLIC type

See:

- TPM2_PrintBin
- TPM2_PrintAuth
- TPM2_Create
- TPM2_ReadPublic

Note: Requires DEBUG_WOLFTPM to be defined

Example

```
TPM2B_PUBLIC pub; //for example, part of the output of a successful TPM2_Create

TPM2_PrintPublicArea(&pub);
```

29.7 wolfTPM2 Wrappers

29.6.3.40 function TPM2_PrintPublicArea More...

29.7.1 Classes

	Name
struct	**TpmSignCbCtx** Context structure for TPM-based certificate signing callback.

29.7.2 Types

	Name
typedef struct TpmSignCbCtx** Context structure for TPM-based certificate signing callback.	

29.7.3 Functions

	Name
WOLFTPM_API int	**wolfTPM2_Test * caps)Test initialization of a TPM and optionally the TPM capabilities can be received.
WOLFTPM_API int	**wolfTPM2_Init ioCb, void * userCtx)Complete initialization of a TPM On failure after argument validation, the device is fully torn down, except for TPM_RC_UPGRADE, which leaves an active context for the vendor firmware-recovery flow.
WOLFTPM_API int	**wolfTPM2_InitWithSpdmKey ioCb, void * userCtx, const byte * rspPubKey, word32 rspPubKeySz)Initialize a TPM and pin a trusted SPDM responder public key. A successful call establishes an authenticated SPDM session even when the TPM accepted the initial cleartext startup probe. When the TPM is already in SPDM-only identity mode, startup is retried over that secure session. The identity mode is chosen for you: the one vendor adapter the build carries, generic TCG when it carries none, and otherwise from the TPM's DID/VID. That last case needs a transport that exposes DID/VID; use wolfTPM2_InitWithSpdmKey_ex to name the mode when it does not. On failure after argument validation, the device is fully torn down, except for TPM_RC_UPGRADE, which leaves an active context for the vendor firmware-recovery flow.

	Name
WOLFTPM_API int	**wolfTPM2_InitWithSpdmKey_ex ioCb, void * userCtx, const byte * rspPubKey, word32 rspPubKeySz, WOLFSPDM_MODE mode)Initialize with a trusted responder key and explicit identity mode. Use this form when the build carries more than one identity adapter and the transport does not expose TPM DID/VID. WOLFSPDM_MODE_TCG is the generic TCG identity flow and needs no vendor adapter; a vendor mode is accepted only when its adapter is compiled in, and BAD_FUNC_ARG is returned before the device is touched when it is not. WOLFSPDM_MODE_AUTO resolves from DID/VID and fails closed rather than guessing a vendor handshake. TPM_RC_UPGRADE leaves an active context for the vendor firmware-recovery flow; call cleanup when that flow is complete.
WOLFTPM_API int	**wolfTPM2_InitWithSpdmPsk ioCb, void * userCtx, const byte * psk, word32 pskSz, const byte * hint, word32 hintSz)Initialize a TPM and authenticate an SPDM session with a PSK. This entry point also recovers a PSK-configured TPM that is already locked in SPDM-only mode by retrying TPM2_Startup over the secure session. On failure after argument validation, the device is fully torn down, except for TPM_RC_UPGRADE, which leaves an active context for the vendor firmware-recovery flow.
WOLFTPM_API int	**wolfTPM2_OpenExisting ioCb, void * userCtx)Use an already initialized TPM, in its current TPM locality.
WOLFTPM_API int	**wolfTPM2_Cleanup * dev)Easy to use TPM and wolfcrypt deinitialization.
WOLFTPM_API int	**wolfTPM2_Cleanup_ex * dev, int doShutdown)Deinitialization of a TPM (and wolfcrypt if it was used)
WOLFTPM_API int	**wolfTPM2_GetTpmDevId * dev)Provides the device ID of a TPM.
WOLFTPM_API int	**wolfTPM2_SelfTest * dev)Asks the TPM to perform its self test.
WOLFTPM_API int	**wolfTPM2_GetCapabilities * caps)Reports the available TPM capabilities.
WOLFTPM_TEST_API int	**wolfTPM2_ParseCapabilities.
WOLFTPM_API int	**wolfTPM2_IsAlgSupported alg, int * isSupported)Report whether the TPM implements a given algorithm.
WOLFTPM_API int	**wolfTPM2_GetHandles * handles)Gets a list of handles.

	Name
WOLFTPM_API int	**wolfTPM2_SpdmInit * dev)Initialize SPDM support on a wolfTPM2 device. Allocates and configures the SPDM context using wolfSPDM. After init, call wolfTPM2_SpdmConnect to establish a secure session.
WOLFTPM_API int	**wolfTPM2_SpdmConnect * dev)Establish an SPDM secure session (full handshake). Uses standard SPDM flow: GET_VERSION -> GET_CAPABILITIES -> NEGOTIATE_ALGORITHMS -> KEY_EXCHANGE -> FINISH. Identity-key modes require a responder key pinned with wolfTPM2_SpdmSetResponderPubKey before this call.
WOLFTPM_API int	**wolfTPM2_SpdmIsConnected * dev)Check if an SPDM secure session is currently active.
WOLFTPM_API word32	**wolfTPM2_SpdmGetSessionId * dev)Get the current SPDM session ID.
WOLFTPM_API int	**wolfTPM2_GetCapability_SPDMSessionInfo * spdmSessionInfo)Read the TPM's view of the active SPDM session (GetCapability TPM_CAP_SPDM_SESSION_INFO, TPM 2.0 Library v1.84).
WOLFTPM_API int	**wolfTPM2_SpdmDisconnect * dev)Disconnect the SPDM secure session. After this, TPM commands are sent in the clear.
WOLFTPM_API int	**wolfTPM2_SpdmGetVersion * dev)Send the SPDM GET_VERSION handshake step. Required before any vendor-defined SPDM command (e.g. PSK provisioning, GET_STATUS).
WOLFTPM_API int	**wolfTPM2_SpdmCleanup * dev)Free SPDM context and resources.
WOLFTPM_API int	**wolfTPM2_SpdmSetResponderPubKey * dev, const byte * pubKey, word32 pubKeySz)Pin the trusted responder key before an identity-key connection.
WOLFTPM_API int	**wolfTPM2_SpdmGetPubKey * dev, byte * pubKey, word32 * pubKeySz)Get the TPM's SPDM-Identity public key (shared TCG function).
WOLFTPM_API int	**wolfTPM2_SpdmSetNuvotonMode for Nuvoton TPMs.
WOLFTPM_API int	**wolfTPM2_SpdmEnable * dev)Enable SPDM on the TPM via NTC2_PreConfig vendor command. Requires platform hierarchy auth. TPM must be reset after this call.
WOLFTPM_API int	**wolfTPM2_SpdmDisable * dev)Disable SPDM on Nuvoton TPM via NTC2_PreConfig. Sets Cfg_H bit 1 to disable SPDM. Requires TPM reset to take effect.

	Name
WOLFTPM_API int	**wolfTPM2_SpdmConnectNuvoton * dev, const byte * reqPubKey, word32 reqPubKeySz, const byte * reqPrivKey, word32 reqPrivKeySz)Establish Nuvoton SPDM secure session with mutual authentication. Uses Nuvoton flow: GET_VERSION -> GET_PUB_KEY -> KEY_EXCHANGE -> GIVE_PUB_KEY -> FINISH. Requires a responder key pinned with wolfTPM2_SpdmSetResponderPubKey.
WOLFTPM_API int	**wolfTPM2_SpdmGetStatus * dev, WOLFSPDM_NUVOTON_STATUS * status)Get SPDM status from the TPM (GET_STS_ vendor command).
WOLFTPM_API int	**wolfTPM2_SpdmSetOnlyMode * dev, int lock)Lock or unlock SPDM-only mode. When locked, TPM only accepts commands over SPDM secure channel.
WOLFTPM_API int	**wolfTPM2_SpdmSetNationsMode * dev)Configure for Nations Technology NS350 SPDM mode. Must be called before wolfTPM2_SpdmConnectNations().
WOLFTPM_API int	**wolfTPM2_SpdmConnectNations * dev, const byte * reqPubKey, word32 reqPubKeySz, const byte * reqPrivKey, word32 reqPrivKeySz)Establish Nations SPDM secure session (identity key mode). Uses TCG flow: GET_VERSION -> GET_PUB_KEY -> KEY_EXCHANGE -> GIVE_PUB_KEY -> FINISH. Requires a responder key pinned with wolfTPM2_SpdmSetResponderPubKey.
WOLFTPM_API int	**wolfTPM2_SpdmNationsIdentityKeySet * dev, int set)Set/unset SPDM identity key provisioning on Nations NS350. Uses vendor command TPM2_VendorSpdmIdentityKeySet (0x20000708).
WOLFTPM_API int	**wolfTPM2_SpdmNationsGetStatus * dev, WOLFSPDM_NATIONS_STATUS * status)Get SPDM status from Nations TPM (PSK mode).
WOLFTPM_API int	**wolfTPM2_SpdmNationsSetOnlyMode * dev, int lock)Lock or unlock SPDM-only mode on Nations TPM (PSK mode).
WOLFTPM_API int	**wolfTPM2_SpdmNationsPskSet * dev, const byte * psk, word32 pskSz)Provision PSK on Nations TPM.
WOLFTPM_API int	**wolfTPM2_SpdmNationsPskClear * dev, const byte * clearAuth, word32 clearAuthSz)Clear PSK from Nations TPM. Requires ClearAuth from PSK_SET.
WOLFTPM_API int	**wolfTPM2_SpdmSetInfineonMode * dev)Configure for Infineon SLB9678 SPDM (PSK mode, Algorithm Set B). Must be called before PSK provisioning vendor commands.

	Name
WOLFTPM_API int	**wolfTPM2_SpdmConnectInfineonTcg * dev, const byte * reqPubKey, word32 reqPubKeySz, const byte * reqPrivKey, word32 reqPrivKeySz)Establish an SLB9678 asymmetric session using the shared TCG flow. Requires firmware advertising asymmetric SPDM and a responder key pinned with wolfTPM2_SpdmSetResponderPubKey. No PSK fallback is attempted.
WOLFTPM_API int	**wolfTPM2_SpdmInfineonGetStatus * dev, WOLFSPDM_INFINEON_STATUS * status)Get SPDM status from Infineon SLB9678 (PSK mode).
WOLFTPM_API int	**wolfTPM2_SpdmInfineonSetOnlyMode * dev, int lock)Lock or unlock SPDM-only mode on Infineon SLB9678.
WOLFTPM_API int	**wolfTPM2_SpdmInfineonEnable * dev, int enable, int lock)Enable or disable the SLB9678 SPDM-PSK secure channel via TPM2_SetCapability(TPM_CAP_CONFIGS, TPM_PT_CONFIG_SPDM_PSK). Requires Platform authorization configured in session slot zero. An unused slot uses empty-password authorization. Pass lock=0 to keep it reversible (enable is undone by disabling); lock=1 lifetime-locks the configuration.
WOLFTPM_API int	**wolfTPM2_SpdmConnectPsk * dev, const byte * psk, word32 pskSz, const byte * hint, word32 hintSz)Establish SPDM secure session via the DSP0274 PSK handshake (PSK_EXCHANGE / PSK_FINISH). Vendor-neutral spec implementation.
WOLFTPM_API int	**wolfTPM2_UnsetAuth * dev, int index)Clears one of the TPM Authorization slots, pointed by its index number.
WOLFTPM_API int	**wolfTPM2_UnsetAuthSession * session)Clears one of the TPM Authorization session slots, pointed by its index number and saves the nonce from the TPM so the session can continue to be used again with wolfTPM2_SetAuthSession.
WOLFTPM_API int	**wolfTPM2_SetAuth * name)Sets a TPM Authorization slot using the provided index, session handle, attributes and auth.
WOLFTPM_API int	**wolfTPM2_SetAuthPassword * auth)Sets a TPM Authorization slot using the provided user auth, typically a password.
WOLFTPM_API int	**wolfTPM2_SetAuthHandle * dev, int index, const WOLFTPM2_HANDLE * handle)Sets a TPM Authorization slot using the user auth associated with a wolfTPM2 Handle.

	Name
WOLFTPM_API int	**wolfTPM2_SetAuthSession sessionAttributes)Sets a TPM Authorization slot using the provided TPM session handle, index and session attributes.
WOLFTPM_API int	**wolfTPM2_SetSessionHandle * tpmSession)Sets a TPM Authorization slot using the provided wolfTPM2 session object.
WOLFTPM_API int	**wolfTPM2_SetAuthHandleName * dev, int index, const WOLFTPM2_HANDLE * handle)Updates the Name used in a TPM Session with the Name associated with wolfTPM2 Handle.
WOLFTPM_API int	**wolfTPM2_StartSession sesType, int encDecAlg)Create a TPM session, Policy, HMAC or Trial.
WOLFTPM_API int	**wolfTPM2_StartSession_ex authHash)Create a TPM session selecting the session auth/hash algorithm.
WOLFTPM_API int	**wolfTPM2_CreateAuthSession_EkPolicy * tpmSession)Creates a TPM session with Policy Secret to satisfy the default EK policy.
WOLFTPM_API int	**wolfTPM2_CreateAuthSession_EkPolicy_ex authHash)Creates a TPM EK policy session selecting the session auth/hash algorithm.
WOLFTPM_API int	**wolfTPM2_CreatePrimaryKey * publicTemplate, const byte * auth, int authSz)Single function to prepare and create a TPM 2.0 Primary Key.
WOLFTPM_API int	**wolfTPM2_CreatePrimaryKey_ex * publicTemplate, const byte * auth, int authSz)Single function to prepare and create a TPM 2.0 Primary Key.
WOLFTPM_API int	**wolfTPM2_ChangeAuthKey * key, WOLFTPM2_HANDLE * parent, const byte * auth, int authSz)Change the authorization secret of a TPM 2.0 key.
WOLFTPM_API int	**wolfTPM2_CreateKey * publicTemplate, const byte * auth, int authSz)Single function to prepare and create a TPM 2.0 Key.
WOLFTPM_API int	**wolfTPM2_LoadKey * keyBlob, WOLFTPM2_HANDLE * parent)Single function to load a TPM 2.0 key.
WOLFTPM_API int	**wolfTPM2_CreateAndLoadKey * publicTemplate, const byte * auth, int authSz)Single function to create and load a TPM 2.0 Key in one step.
WOLFTPM_API int	**wolfTPM2_CreateLoadedKey * publicTemplate, const byte * auth, int authSz)Creates and loads a key using single TPM 2.0 operation, and stores encrypted private key material.

	Name
WOLFTPM_API int	**wolfTPM2_LoadPublicKey * pub)Wrapper to load the public part of an external key.
WOLFTPM_API int	**wolfTPM2_LoadPrivateKey * sens)Single function to import an external private key and load it into the TPM in one step.
WOLFTPM_API int	**wolfTPM2_ImportPrivateKey * sens)Single function to import an external private key and load it into the TPM in one step.
WOLFTPM_API int	**wolfTPM2_LoadRsaPublicKey * key, const byte * rsaPub, word32 rsaPubSz, word32 exponent)Helper function to import the public part of an external RSA key.
WOLFTPM_API int	**wolfTPM2_LoadRsaPublicKey_ex hashAlg)Advanced helper function to import the public part of an external RSA key.
WOLFTPM_API int	**wolfTPM2_ImportRsaPrivateKey hashAlg)Import an external RSA private key.
WOLFTPM_API int	**wolfTPM2_ImportRsaPrivateKeySeed attributes, byte * seed, word32 seedSz)Import an external RSA private key with custom seed.
WOLFTPM_API int	**wolfTPM2_LoadRsaPrivateKey * key, const byte * rsaPub, word32 rsaPubSz, word32 exponent, const byte * rsaPriv, word32 rsaPrivSz)Helper function to import and load an external RSA private key in one step.
WOLFTPM_API int	**wolfTPM2_LoadRsaPrivateKey_ex hashAlg)Advanced helper function to import and load an external RSA private key in one step.
WOLFTPM_API int	**wolfTPM2_LoadEccPublicKey * key, int curveId, const byte * eccPubX, word32 eccPubXSz, const byte * eccPubY, word32 eccPubYSz)Helper function to import the public part of an external ECC key.
WOLFTPM_API int	**wolfTPM2_LoadEccPublicKey_ex objectAttributes)Variant of wolfTPM2_LoadEccPublicKey that lets the caller select the signing scheme, hash algorithm, and object attributes - required for loading ECDH peer keys (decrypt attribute) or non-default scheme/hash combinations.
WOLFTPM_API int	**wolfTPM2_ImportEccPrivateKey * keyBlob, int curveId, const byte * eccPubX, word32 eccPubXSz, const byte * eccPubY, word32 eccPubYSz, const byte * eccPriv, word32 eccPrivSz)Helper function to import the private material of an external ECC key.
WOLFTPM_API int	**wolfTPM2_ImportEccPrivateKeySeed attributes, byte * seed, word32 seedSz)Helper function to import the private material of an external ECC key.

	Name
WOLFTPM_API int	**wolfTPM2_LoadEccPrivateKey * key, int curveId, const byte * eccPubX, word32 eccPubXSz, const byte * eccPubY, word32 eccPubYSz, const byte * eccPriv, word32 eccPrivSz)Helper function to import and load an external ECC private key in one step.
WOLFTPM_API int	**wolfTPM2_ReadPublicKey handle)Helper function to receive the public part of a loaded TPM object using its handle.
WOLFTPM_API int	**wolfTPM2_CreateKeySeal * publicTemplate, const byte * auth, int authSz, const byte * sealData, int sealSize)Using this wrapper a secret can be sealed inside a TPM 2.0 Key.
WOLFTPM_API int	**wolfTPM2_CreateKeySeal_ex pcrAlg, byte * pcrArray, word32 pcrArraySz, const byte * sealData, int sealSize)Using this wrapper a secret can be sealed inside a TPM 2.0 Key with pcr selection.
WOLFTPM_API int	**wolfTPM2_ComputeName * out)Helper function to generate a hash of the public area of an object in the format expected by the TPM.
WOLFTPM_API int	**wolfTPM2_SensitiveToPrivate.
WOLFTPM_API word32	**wolfTPM2_RsaKey_Exponent**(const byte * e, word32 eSz)Converts a big-endian byte array to a native word32 value. Used for RSA exponent conversion from ASN.1/DER format.
WOLFTPM_API int	**wolfTPM2_ImportPrivateKeyBuffer objectAttributes, byte * seed, word32 seedSz)Helper function to import PEM/DER or RSA/ECC private key.
WOLFTPM_API int	**wolfTPM2_ImportPublicKeyBuffer objectAttributes)Helper function to import PEM/DER formatted RSA/ECC public key.
WOLFTPM_API int	**wolfTPM2_ExportPublicKeyBuffer * tpmKey, int encodingType, byte * out, word32 * outSz)Helper function to export a TPM RSA/ECC public key with PEM/DER formatting.
WOLFTPM_API int	**wolfTPM2_RsaPrivateKeyImportDer hashAlg)Helper function to import Der rsa key directly.
WOLFTPM_API int	**wolfTPM2_RsaPrivateKeyImportPem hashAlg)Helper function to import Pem rsa key directly.
WOLFTPM_API int	**wolfTPM2_RsaKey_TpmToWolf * tpmKey, RsaKey * wolfKey)Extract a RSA TPM key and convert it to a wolfcrypt key.
WOLFTPM_API int	**wolfTPM2_RsaKey_TpmToPemPub * keyBlob, byte * pem, word32 * pemSz)Convert a public RSA TPM key to PEM format public key. Note: This API is a wrapper around wolfTPM2_ExportPublicKeyBuffer.

	Name
WOLFTPM_API int	**wolfTPM2_RsaKey_WolfToTpm * tpmKey)Import a RSA wolfcrypt key into the TPM.
WOLFTPM_API int	**wolfTPM2_RsaKey_WolfToTpm_ex * tpmKey)Import a RSA wolfcrypt key into the TPM under a specific Primary Key or Hierarchy.
WOLFTPM_API int	**wolfTPM2_CreateRsaKeyBlob * tpmKey)Create an encrypted RSA key blob from a wolfCrypt key under a specific parent key.
WOLFTPM_API int	**wolfTPM2_RsaKey_PubPemToTpm * tpmKey, const byte * pem, word32 pemSz)Import a PEM format public key from a file into the TPM.
WOLFTPM_API int	**wolfTPM2_DecodeRsaDer attributes)Import DER RSA private or public key into TPM public and sensitive structures. This does not make any calls to TPM hardware.
WOLFTPM_API int	**wolfTPM2_EccKey_TpmToWolf * tpmKey, ecc_key * wolfKey)Extract a ECC TPM key and convert to to a wolfcrypt key.
WOLFTPM_API int	**wolfTPM2_EccKey_WolfToTpm * tpmKey)Import a ECC wolfcrypt key into the TPM.
WOLFTPM_API int	**wolfTPM2_EccKey_WolfToTpm_ex * tpmKey)Import ECC wolfcrypt key into the TPM under a specific Primary Key or Hierarchy.
WOLFTPM_API int	**wolfTPM2_CreateEccKeyBlob * tpmKey)Create an encrypted ECC key blob from a wolfCrypt key under a specific parent key.
WOLFTPM_API int	**wolfTPM2_EccKey_WolfToPubPoint * pubPoint)Import a ECC public key generated from wolfcrypt key into the TPM.
WOLFTPM_API int	**wolfTPM2_DecodeEccDer attributes)Import DER ECC private or public key into TPM public and sensitive structures. This does not make any calls to TPM hardware.
WOLFTPM_API int	**wolfTPM2_SignHash * key, const byte * digest, int digestSz, byte * sig, int * sigSz)Helper function to sign arbitrary data using a TPM key.
WOLFTPM_API int	**wolfTPM2_SignHashScheme hashAlg)Advanced helper function to sign arbitrary data using a TPM key, and specify the signature scheme and hashing algorithm.
WOLFTPM_API int	**wolfTPM2_VerifyHash * key, const byte * sig, int sigSz, const byte * digest, int digestSz)Helper function to verify a TPM generated signature.
WOLFTPM_API int	**wolfTPM2_VerifyHash_ex * key, const byte * sig, int sigSz, const byte * digest, int digestSz, int hashAlg)Helper function to verify a TPM generated signature.

	Name
WOLFTPM_API int	**wolfTPM2_VerifyHashScheme hashAlg)Advanced helper function to verify a TPM generated signature.
WOLFTPM_API int	**wolfTPM2_VerifyHashTicket * checkTicket)Advanced helper function to verify a TPM generated signature and return ticket.
WOLFTPM_API int	**wolfTPM2_SignSequenceStart * sequenceHandle)Post-Quantum Cryptography: Start a signing sequence with context support.
WOLFTPM_API int	**wolfTPM2_SignSequenceUpdate sequenceHandle, const byte * data, int dataSz)Post-Quantum Cryptography: Update a signing sequence with data.
WOLFTPM_API int	**wolfTPM2_SignSequenceComplete * key, const byte * data, int dataSz, byte * sig, int * sigSz)Post-Quantum Cryptography: Complete a signing sequence.
WOLFTPM_API int	**wolfTPM2_VerifySequenceStart * sequenceHandle)Post-Quantum Cryptography: Start a verification sequence with context support.
WOLFTPM_API int	**wolfTPM2_VerifySequenceUpdate sequenceHandle, const byte * data, int dataSz)Post-Quantum Cryptography: Update a verification sequence with data.
WOLFTPM_API int	**wolfTPM2_VerifySequenceComplete * validation)Post-Quantum Cryptography: Complete a verification sequence.
WOLFTPM_API int	**wolfTPM2_SignDigest * key, const byte * digest, int digestSz, const byte * context, int contextSz, byte * sig, int * sigSz)Post-Quantum Cryptography: Sign a digest with context support.
WOLFTPM_API int	**wolfTPM2_VerifyDigestSignature * validation)Post-Quantum Cryptography: Verify a digest signature with context support.
WOLFTPM_API int	**wolfTPM2_Encapsulate * key, byte * ciphertext, int * ciphertextSz, byte * sharedSecret, int * sharedSecretSz)Post-Quantum Cryptography: KEM Encapsulate (public key operation)
WOLFTPM_API int	**wolfTPM2_Decapsulate * key, const byte * ciphertext, int ciphertextSz, byte * sharedSecret, int * sharedSecretSz)Post-Quantum Cryptography: KEM Decapsulate (private key operation)
WOLFTPM_API int	**wolfTPM2_GetKeyTemplate_MLDSA parameterSet, int allowExternalMu)Create a key template for ML-DSA signing keys (TCG v185)

	Name
WOLFTPM_API int	**wolfTPM2_GetKeyTemplate_MLDSA_ex nameAlg)Create an ML-DSA key template selecting the name algorithm (TCG v185)
WOLFTPM_API int	**wolfTPM2_GetKeyTemplate_HASH_MLDSA hashAlg)Create a key template for Pre-Hash ML-DSA signing keys (TCG v185)
WOLFTPM_API int	**wolfTPM2_GetKeyTemplate_HASH_MLDSA_ex nameAlg)Create a Pre-Hash ML-DSA key template selecting the name algorithm (TCG v185)
WOLFTPM_API int	**wolfTPM2_GetKeyTemplate_MLKEM parameterSet)Create a key template for ML-KEM decryption keys (TCG v185)
WOLFTPM_API int	**wolfTPM2_GetKeyTemplate_MLKEM_ex nameAlg)Create an ML-KEM key template selecting the name algorithm (TCG v185)
WOLFTPM_API int	**wolfTPM2_ECDHGenKey * ecdhKey, int curve_id, const byte * auth, int authSz)Generates and then loads a ECC key-pair with NULL hierarchy for Diffie-Hellman exchange.
WOLFTPM_API int	**wolfTPM2_ECDHGen * pubPoint, byte * out, int * outSz)Generates ephemeral key and computes Z (shared secret)
WOLFTPM_API int	**wolfTPM2_ECDHGenZ * pubPoint, byte * out, int * outSz)Computes Z (shared secret) using pubPoint and loaded private ECC key.
WOLFTPM_API int	**wolfTPM2_ECDHEGenKey * ecdhKey, int curve_id)Generates ephemeral ECC key and returns array index (2 phase method)
WOLFTPM_API int	**wolfTPM2_ECDHEGenZ * pubPoint, byte * out, int * outSz)Computes Z (shared secret) using pubPoint and counter (2 phase method)
WOLFTPM_API int	**wolfTPM2_RsaEncrypt padScheme, const byte * msg, int msgSz, byte * out, int * outSz)Perform RSA encryption using a TPM 2.0 key.
WOLFTPM_API int	**wolfTPM2_RsaEncrypt_ex hashAlg)Perform RSA encryption selecting the padding scheme hash algorithm.
WOLFTPM_API int	**wolfTPM2_RsaDecrypt padScheme, const byte * in, int inSz, byte * msg, int * msgSz)Perform RSA decryption using a TPM 2.0 key.
WOLFTPM_API int	**wolfTPM2_RsaDecrypt_ex hashAlg)Perform RSA decryption selecting the padding scheme hash algorithm.
WOLFTPM_API int	**wolfTPM2_ReadPCR * dev, int pcrIndex, int hashAlg, byte * digest, int * pDigestLen)Read the values of a specified TPM 2.0 Platform Configuration Registers(PCR)

	Name
WOLFTPM_API int	**wolfTPM2_ResetPCR * dev, int pcrIndex)Reset a PCR register to its default value.
WOLFTPM_API int	**wolfTPM2_SetLocality * dev, int locality)Change the active TPM locality at runtime.
WOLFTPM_API int	**wolfTPM2_ExtendPCR * dev, int pcrIndex, int hashAlg, const byte * digest, int digestLen)Extend a PCR register with a user provided digest.
WOLFTPM_API int	**wolfTPM2_AllocatePCRBanks * allocOut)Re-provision which PCR banks the TPM has allocated.
WOLFTPM_API int	**wolfTPM2_AllocatePCRBanks_ex * allocOut)Re-provision the TPM's PCR banks using a caller supplied session.
WOLFTPM_API int	**wolfTPM2_NVCreateAuth * nv, word32 nvIndex, word32 nvAttributes, word32 maxSize, const byte * auth, int authSz)Creates a new NV Index to be later used for storing data into the TPM's NVRAM.
WOLFTPM_API int	**wolfTPM2_NVCreateAuthPolicy * nv, word32 nvIndex, word32 nvAttributes, word32 maxSize, const byte * auth, int authSz, const byte * authPolicy, int authPolicySz)Creates a new NV Index to be later used for storing data into the TPM's NVRAM.
WOLFTPM_API int	**wolfTPM2_NVCreateAuthPolicy_ex nameAlg)Variant of wolfTPM2_NVCreateAuthPolicy that lets the caller pass an explicit nameAlg. Required when authPolicy was computed with SM3-256, SHA3-256, or any other non-SHA hash that the size-based legacy inference cannot distinguish from SHA-256.
WOLFTPM_API int	**wolfTPM2_NVWriteAuth * nv, word32 nvIndex, byte * dataBuf, word32 dataSz, word32 offset)Stores user data to a NV Index, at a given offset.
WOLFTPM_API int	**wolfTPM2_NVWriteAuthPolicy * nv, word32 nvIndex, byte * dataBuf, word32 dataSz, word32 offset)Stores user data to a NV Index, at a given offset. Allows using a policy session and PCR's for authentication.
WOLFTPM_API int	**wolfTPM2_NVExtend * nv, word32 nvIndex, byte * dataBuf, word32 dataSz)Extend data to an NV index.
WOLFTPM_API int	**wolfTPM2_NVReadAuth * nv, word32 nvIndex, byte * dataBuf, word32 * pDataSz, word32 offset)Reads user data from a NV Index, starting at the given offset.

	Name
WOLFTPM_API int	**wolfTPM2_NVReadAuthPolicy * nv, word32 nvIndex, byte * dataBuf, word32 * pDataSz, word32 offset)Reads user data from a NV Index, starting at the given offset. Allows using a policy session and PCR's for authentication.
WOLFTPM_API int	**wolfTPM2_NVReadCert handle, uint8_t * buffer, uint32_t * len)Helper to get size of NV and read buffer without authentication. Typically used for reading a certificate from an NV.
WOLFTPM_API int	**wolfTPM2_NVIncrement * nv)Increments an NV one-way counter.
WOLFTPM_API int	**wolfTPM2_NVOpen * nv, word32 nvIndex, const byte * auth, word32 authSz)Open an NV and populate the required authentication and name hash.
WOLFTPM_API int	**wolfTPM2_NVWriteLock * nv)Lock writes on the specified NV Index.
WOLFTPM_API int	**wolfTPM2_NVDeleteAuth * dev, WOLFTPM2_HANDLE * parent, word32 nvIndex)Destroys an existing NV Index.
WOLFTPM_API int	**wolfTPM2_NVCreate authHandle, word32 nvIndex, word32 nvAttributes, word32 maxSize, const byte * auth, int authSz)Deprecated, use newer API.
WOLFTPM_API int	**wolfTPM2_NVWrite authHandle, word32 nvIndex, byte * dataBuf, word32 dataSz, word32 offset)Deprecated, use newer API.
WOLFTPM_API int	**wolfTPM2_NVRead authHandle, word32 nvIndex, byte * dataBuf, word32 * dataSz, word32 offset)Deprecated, use newer API.
WOLFTPM_API int	**wolfTPM2_NVDelete authHandle, word32 nvIndex)Deprecated, use newer API.
WOLFTPM_API int	**wolfTPM2_NVReadPublic * nvPublic)Extracts the public information about an nvIndex, such as maximum size.
WOLFTPM_API int	**wolfTPM2_NVStoreKey persistentHandle)Helper function to store a TPM 2.0 Key into the TPM's NVRAM.
WOLFTPM_API int	**wolfTPM2_NVDeleteKey * key)Helper function to delete a TPM 2.0 Key from the TPM's NVRAM.
WOLFTPM_API struct WC_RNG *	**wolfTPM2_GetRng * dev)Get the wolfcrypt RNG instance used for wolfTPM.
WOLFTPM_API int	**wolfTPM2_GetRandom * dev, byte * buf, word32 len)Get a set of random number, generated with the TPM RNG or wolfcrypt RNG.
WOLFTPM_API int	**wolfTPM2_UnloadHandle * dev, WOLFTPM2_HANDLE * handle)Use to discard any TPM loaded object.

	Name
WOLFTPM_API int	**wolfTPM2_Clear * dev)Clears the TPM with TPM2_Clear, authorized by the lockout hierarchy. This removes the owner and endorsement hierarchy objects and the NV indices created by the owner, and resets their authorizations. NV indices created by the platform (TPMA_NV_PLATFORMCREATE) are preserved. It does not deinitialize wolfTPM; use wolfTPM2_Cleanup for that.
WOLFTPM_API int	**wolfTPM2_DictionaryAttackLockReset * dev)Reset the dictionary attack lockout failed-tries counter.
WOLFTPM_API int	**wolfTPM2_DictionaryAttackParameters * dev, word32 newMaxTries, word32 newRecoveryTime, word32 lockoutRecovery)Set the dictionary attack lockout parameters.
WOLFTPM_API int	**wolfTPM2_HashStart hashAlg, const byte * usageAuth, word32 usageAuthSz)Helper function to start a TPM generated hash.
WOLFTPM_API int	**wolfTPM2_HashUpdate * hash, const byte * data, word32 dataSz)Update a TPM generated hash with new user data.
WOLFTPM_API int	**wolfTPM2_HashFinish * hash, byte * digest, word32 * digestSz)Finalize a TPM generated hash and get the digest output in a user buffer.
WOLFTPM_API int	**wolfTPM2_LoadKeyedHashKey * key, WOLFTPM2_HANDLE * parent, int hashAlg, const byte * keyBuf, word32 keySz, const byte * usageAuth, word32 usageAuthSz)Creates and loads a new TPM key of KeyedHash type, typically used for HMAC operations.
WOLFTPM_API int	**wolfTPM2_HmacStart hashAlg, const byte * keyBuf, word32 keySz, const byte * usageAuth, word32 usageAuthSz)Helper function to start a TPM generated hmac.
WOLFTPM_API int	**wolfTPM2_HmacUpdate * hmac, const byte * data, word32 dataSz)Update a TPM generated hmac with new user data.
WOLFTPM_API int	**wolfTPM2_HmacFinish * hmac, byte * digest, word32 * digestSz)Finalize a TPM generated hmac and get the digest output in a user buffer.
WOLFTPM_API int	**wolfTPM2_LoadSymmetricKey * key, int alg, const byte * keyBuf, word32 keySz)Loads an external symmetric key into the TPM.
WOLFTPM_API int	**wolfTPM2_EncryptDecryptBlock * key, const byte * in, byte * out, word32 inOutSz, byte * iv, word32 ivSz, int isDecrypt)Encrypt or decrypt a single block of data with the symmetric key.

	Name
WOLFTPM_API int	**wolfTPM2_SetCommand commandCode, int enableFlag)Vendor specific TPM command, used to enable other restricted TPM commands.
WOLFTPM_API int	**wolfTPM2_Reset * dev, int doShutdown, int doStartup)Helper function to shutdown, startup or reset the TPM.
WOLFTPM_API int	**wolfTPM2_Shutdown * dev, int doStartup)Helper function to shutdown or reset the TPM.
WOLFTPM_API int	**wolfTPM2_UnloadHandles * dev, word32 handleStart, word32 handleCount)One-shot API to unload subsequent TPM handles.
WOLFTPM_API int	**wolfTPM2_UnloadHandles_AllTransient * dev)One-shot API to unload all transient TPM handles.
WOLFTPM_API int	**wolfTPM2_GetKeyTemplate_RSA objectAttributes)Prepares a TPM public template for new RSA key based on user selected object attributes.
WOLFTPM_API int	**wolfTPM2_GetKeyTemplate_RSA_ex sigHash)Prepares a TPM public template for new RSA key based on user selected object attributes.
WOLFTPM_API int	**wolfTPM2_GetKeyTemplate_ECC sigScheme)Prepares a TPM public template for new ECC key based on user selected object attributes.
WOLFTPM_API int	**wolfTPM2_GetKeyTemplate_ECC_ex sigHash)Prepares a TPM public template for new ECC key based on user selected object attributes.
WOLFTPM_API int	**wolfTPM2_GetKeyTemplate_Symmetric algMode, int isSign, int isDecrypt)Prepares a TPM public template for new Symmetric key.
WOLFTPM_API int	**wolfTPM2_GetKeyTemplate_KeyedHash hashAlg, int isSign, int isDecrypt)Prepares a TPM public template for new KeyedHash key.
WOLFTPM_API int	**wolfTPM2_GetKeyTemplate_KeySeal nameAlg)Prepares a TPM public template for new key for sealing secrets.
WOLFTPM_API int	**wolfTPM2_GetKeyTemplate_EK nameAlg, int highRange)Prepares a TPM public template for generating the TPM Endorsement Key.
WOLFTPM_API int	**wolfTPM2_GetKeyTemplate_EKIndex * publicTemplate)Helper to get the Endorsement public key template by NV index.
WOLFTPM_API int	**wolfTPM2_GetKeyTemplate_RSA_EK * publicTemplate)Prepares a TPM public template for generating the TPM Endorsement Key of RSA type.

	Name
WOLFTPM_API int	**wolfTPM2_GetKeyTemplate_ECC_EK * publicTemplate)Prepares a TPM public template for generating the TPM Endorsement Key of ECC type.
WOLFTPM_API int	**wolfTPM2_GetKeyTemplate_RSA_SRK * publicTemplate)Prepares a TPM public template for generating a new TPM Storage Key of RSA type.
WOLFTPM_API int	**wolfTPM2_GetKeyTemplate_ECC_SRK * publicTemplate)Prepares a TPM public template for generating a new TPM Storage Key of ECC type.
WOLFTPM_API int	**wolfTPM2_GetKeyTemplate_RSA_AIK * publicTemplate)Prepares a TPM public template for generating a new TPM Attestation Key of RSA type.
WOLFTPM_API int	**wolfTPM2_GetKeyTemplate_ECC_AIK * publicTemplate)Prepares a TPM public template for generating a new TPM Attestation Key of ECC type.
WOLFTPM_API int	**wolfTPM2_SetKeyTemplate_Unique * publicTemplate, const byte * unique, int uniqueSz)Sets the unique area of a public template used by Create or CreatePrimary.
WOLFTPM_API int	**wolfTPM2_GetNvAttributesTemplate auth, word32 * nvAttributes)Prepares a TPM NV Index template.
WOLFTPM_API int	**wolfTPM2_CreateEK alg)Generates a new TPM Endorsement key, based on the user selected algorithm, RSA or ECC.
WOLFTPM_API int	**wolfTPM2_CreateSRK alg, const byte * auth, int authSz)Generates a new TPM Primary Key that will be used as a Storage Key for other TPM keys.
WOLFTPM_API int	**wolfTPM2_CreateAndLoadAIK * srkKey, const byte * auth, int authSz)Generates a new TPM Attestation Key under the provided Storage Key.
WOLFTPM_API int	**wolfTPM2_GetTime * getTimeOut)One-shot API to generate a TPM signed timestamp.
WOLFTPM_API int	**wolfTPM2_CSR_SetCustomExt structure.
WOLFTPM_API int	**wolfTPM2_CSR_SetKeyUsage structure. Pass either extended key usage or key usage values. Mixed string types are not supported, however you can call `wolfTPM2_CSR_SetKeyUsage` twice (once for extended key usage strings and once for standard key usage strings).
WOLFTPM_API int	**wolfTPM2_CSR_SetSubject structure.
WOLFTPM_API int	**wolfTPM2_CSR_MakeAndSign_ex structure with subject and key usage already set.
WOLFTPM_API int	**wolfTPM2_CSR_MakeAndSign structure with subject and key usage already set.

	Name
WOLFTPM_API int	**wolfTPM2_CSR_Generate_ex). Single shot API for outputting a CSR or self-signed cert based on TPM key.
WOLFTPM_API int	**wolfTPM2_CSR_Generate). Single shot API for outputting a CSR or self-signed cert based on TPM key.
WOLFTPM_API int	**wolfTPM2_ChangePlatformAuth * session)Helper to set the platform hierarchy authentication value to random. Setting the platform auth to random value is used to prevent application from being able to use platform hierarchy. This is defined in section 10 of the TCG PC Client Platform specification.
WOLFTPM_API int	**wolfTPM2_ChangeHierarchyAuth authHandle)Helper to set the hierarchy authentication value to random. Setting the platform auth to random value is used to prevent application from being able to use platform hierarchy. This is defined in section 10 of the TCG PC Client Platform specification.
WOLFTPM_API int	**wolfTPM2_CryptoDevCb**(int devId, wc_CryptoInfo * info, void * ctx)A reference crypto callback API for using the TPM for crypto offload. This callback function is registered using wolfTPM2_SetCryptoDevCb or wc_CryptoDev_RegisterDevice.
WOLFTPM_API int	**wolfTPM2_SetCryptoDevCb * tpmCtx, int * pDevId)Register a crypto callback function and return assigned devId.
WOLFTPM_API int	**wolfTPM2_ClearCryptoDevCb * dev, int devId)Clears the registered crypto callback.
WOLFTPM_API WOLFTPM2_DEV.	
WOLFTPM_API int	**wolfTPM2_Free that was allocated by wolfTPM2_New.
WOLFTPM_API WOLFTPM2_KEYBLOB.	
WOLFTPM_API int	**wolfTPM2_FreeKeyBlob that was allocated with wolfTPM2_NewKeyBlob.
WOLFTPM_API TPMT_PUBLIC.	
WOLFTPM_API int	**wolfTPM2_FreePublicTemplate that was allocated with wolfTPM2_NewPublicTemplate.
WOLFTPM_API WOLFTPM2_KEY.	
WOLFTPM_API int	**wolfTPM2_FreeKey that was allocated with wolfTPM2_NewKey.
WOLFTPM_API WOLFTPM2_SESSION.	
WOLFTPM_API int	**wolfTPM2_FreeSession that was allocated with wolfTPM2_NewSession.
WOLFTPM_API WOLFTPM2_CSR.	
WOLFTPM_API int	**wolfTPM2_FreeCSR that was allocated with wolfTPM2_NewCSR.
WOLFTPM_API WOLFTPM2_HANDLE *	**wolfTPM2_GetHandleRefFromKey.
WOLFTPM_API WOLFTPM2_HANDLE *	**wolfTPM2_GetHandleRefFromKeyBlob.
WOLFTPM_API WOLFTPM2_HANDLE *	**wolfTPM2_GetHandleRefFromSession.

	Name
WOLFTPM_API TPM_HANDLE	**wolfTPM2_GetHandleValue**(WOLFTPM2_HANDLE * handle)Get the 32-bit handle value from the WOLFTPM2_HANDLE.
WOLFTPM_API int	**wolfTPM2_SetKeyAuthPassword * key, const byte * auth, int authSz)Set the authentication data for a key.
WOLFTPM_API int	**wolfTPM2_GetKeyBlobAsBuffer * key)Marshal data from a keyblob to a binary buffer. This can be stored to disk for loading in a separate process or after power cycling. If buffer is not provided then size only will be returned.
WOLFTPM_API int	**wolfTPM2_GetKeyBlobAsSeparateBuffers * key)Marshal data from a keyblob to a binary buffer. This can be stored to disk for loading in a separate process or after power cycling. If either buffer is NULL then the size will be returned for each part.
WOLFTPM_API int	**wolfTPM2_SetKeyBlobFromBuffer struct. This can be used to load a keyblob that was previously marshaled by wolfTPM2_GetKeyBlobAsBuffer.
WOLFTPM_API int	**wolfTPM2_PolicyRestart sessionHandle)Restart the policy digest for a policy session.
WOLFTPM_API int	**wolfTPM2_GetPolicyDigest sessionHandle, byte * policyDigest, word32 * policyDigestSz)Get the policy digest of the session that was passed in wolfTPM2_GetPolicyDigest.
WOLFTPM_API int	**wolfTPM2_PolicyPCR pcrAlg, byte * pcrArray, word32 pcrArraySz)Apply the PCR's to the policy digest for the policy session.
WOLFTPM_API int	**wolfTPM2_PolicyTransportSPDM * tpmKeyName)Add TPM2_PolicyTransportSPDM (TPM 2.0 Library v1.84) to a policy session. An entity whose authPolicy contains this term can only be authorized by a command that arrives inside an SPDM session. Optional key names further restrict which SPDM session qualifies.
WOLFTPM_API int	**wolfTPM2_PolicyAuthorize * checkTicket, const byte * pcrDigest, word32 pcrDigestSz, const byte * policyRef, word32 policyRefSz)Apply the PCR's to the policy digest for the policy session.
WOLFTPM_API int	**wolfTPM2_PCRGetDigest pcrAlg, byte * pcrArray, word32 pcrArraySz, byte * pcrDigest, word32 * pcrDigestSz)Get a cumulative digest of the PCR's specified.

	Name
WOLFTPM_API int	**wolfTPM2_PolicyRefMake pcrAlg, byte * digest, word32 * digestSz, const byte * policyRef, word32 policyRefSz)Utility for generating a policy ref digest. If no policy reference (nonce) used then just rehash the provided digest again (update -> final)
WOLFTPM_API int	**wolfTPM2_PolicyPCRMake pcrAlg, byte * pcrArray, word32 pcrArraySz, const byte * pcrDigest, word32 pcrDigestSz, byte * digest, word32 * digestSz)Utility for generating a policy PCR digest.
WOLFTPM_API int	**wolfTPM2_PolicyTransportSPDMMake * tpmKeyName, byte * digest, word32 * digestSz)Compute the policy digest that TPM2_PolicyTransportSPDM produces, without a TPM. Use it to build the authPolicy of an NV index or key that must only be reachable over an SPDM session.
WOLFTPM_API int	**wolfTPM2_PolicyCommandCodeMake cc)Compute the policy digest for PolicyCommandCode on a fresh session.
WOLFTPM_API int	**wolfTPM2_PolicyHash cc, const byte * input, word32 inputSz)Utility for creating a policy hash. Generic helper that takes command code and input array. policyDigestnew = hash(policyDigestOld
WOLFTPM_API int	**wolfTPM2_PolicyAuthorizeMake * pub, byte * digest, word32 * digestSz, const byte * policyRef, word32 policyRefSz)Utility for generating a policy authorization digest based on a public key.
WOLFTPM_API int	**wolfTPM2_PolicyPassword * tpmSession, const byte * auth, int authSz)Wrapper for setting a policy password and calling TPM2_PolicyPassword. This will set a password (in clear) for the policy session instead of HMAC.
WOLFTPM_API int	**wolfTPM2_PolicyAuthValue * tpmSession, const byte * auth, int authSz)Wrapper for setting a policy auth value that is added to the HMAC key for a policy session.
WOLFTPM_API int	**wolfTPM2_PolicyCommandCode cc)Wrapper for setting a policy command code.
WOLFTPM_API int	**wolfTPM2_PolicyOR * pHashList)Wrapper for satisfying a policy session with a compound OR of digests.
WOLFTPM_API int	**wolfTPM2_SetPrimaryPolicy hashAlg, const byte * authPolicy, word32 authPolicySz)Set (or clear) the authPolicy of a hierarchy.

	Name
WOLFTPM_API int	**wolfTPM2_SetIdentityAuth * dev, WOLFTPM2_HANDLE * handle, uint8_t * masterPassword, uint16_t masterPasswordSz)Set authentication for pre-provisioned identity keys.
WOLFTPM_LOCAL int	**GetKeyTemplateRSA sigHash)Internal helper to create RSA key template.
WOLFTPM_LOCAL int	**GetKeyTemplateECC sigHash)Internal helper to create ECC key template.
WOLFTPM_API int	**wolfTPM2_FirmwareUpgradeHash cb, void * cb_ctx)Perform TPM firmware upgrade with pre-computed manifest hash.
WOLFTPM_API int	**wolfTPM2_FirmwareUpgradeHash_ex * startSession)Perform TPM firmware upgrade using a caller-supplied authorization session.
WOLFTPM_API int	**wolfTPM2_FirmwareUpgrade cb, void * cb_ctx)Perform TPM firmware upgrade.
WOLFTPM_API int	**wolfTPM2_FirmwareUpgrade_ex * startSession)Perform TPM firmware upgrade using a caller-supplied authorization session.
WOLFTPM_API int	**wolfTPM2_FirmwareUpgradeRecover cb, void * cb_ctx)Recover from failed TPM firmware upgrade.
WOLFTPM_API int	**wolfTPM2_FirmwareUpgradeRecover_ex * startSession)Recover from a failed firmware upgrade using a caller-supplied session.
WOLFTPM_API int	**wolfTPM2_FirmwareUpgradeCancel * dev)Cancel ongoing TPM firmware upgrade.
WOLFTPM_API int	**wolfTPM2_ST33_ManifestVersion**(const uint8_t * manifest, uint32_t manifest_sz, word16 * major, word16 * minor)Read the firmware version an ST33 firmware image upgrades to.
WOLFTPM_API int	**wolfTPM2_ST33_CmdImplemented cc, int * isImpl)Report whether the TPM implements a command code.
WOLFTPM_API int	**wolfTPM2_ST33_GetFwUpgradeCommands * ccData, int * fromTpm)Choose the ST33 field upgrade command codes for the attached TPM.
WOLFTPM_TEST_API int	**wolfTPM2_ST33_FwUpgradeCommands * ccData)Choose the ST33 field upgrade command codes from the version fields.

29.7.4 Detailed Description

This module describes the rich API of wolfTPM called wrappers.

wolfTPM wrappers are used in two main cases:

- Perform common TPM 2.0 tasks, like key generation and storage
- Perform complex TPM 2.0 tasks, like attestation and parameter encryption

wolfTPM enables quick and rapid use of TPM 2.0 thanks to its many wrapper functions.

29.7.5 Types Documentation

```
typedef struct TpmSignCbCtx TpmSignCbCtx;
```

Context structure for TPM-based certificate signing callback.

See:

- wolfTPM2_CSR_MakeAndSign_ex
- wc_SignCert_cb

This structure holds the TPM device and key references needed for the certificate signing callback (wc_SignCertCb). It is used internally by wolfTPM2_CSR_MakeAndSign_ex when using the callback-based signing approach.

29.7.6 Functions Documentation

29.7.5.1 typedef TpmSignCbCtx

```
WOLFTPM_API int wolfTPM2_Test(
    TPM2HalIoCb ioCb,
    void * userCtx,
    WOLFTPM2_CAPS * caps
)
```

Test initialization of a TPM and optionally the TPM capabilities can be received.

Parameters:

- **ioCb** function pointer to a IO callback (see [hal#file-tpm-io.h))
- **userCtx** pointer to a user context (can be NULL)
- **caps** to a structure of WOLFTPM2_CAPS type for returning the TPM capabilities (can be NULL)

See:

- wolfTPM2_Init
- TPM2_Init

Return:

- TPM_RC_SUCCESS: successful
- TPM_RC_FAILURE: generic failure (check TPM IO and TPM return code)
- BAD_FUNC_ARG: check the provided arguments

```
WOLFTPM_API int wolfTPM2_Init(
    WOLFTPM2_DEV * dev,
    TPM2HalIoCb ioCb,
    void * userCtx
)
```

Complete initialization of a TPM On failure after argument validation, the device is fully torn down, except for TPM_RC_UPGRADE, which leaves an active context for the vendor firmware-recovery flow.

Parameters:

- **dev** pointer to an empty structure of WOLFTPM2_DEV type
- **ioCb** function pointer to a IO callback (see [hal#file-tpm-io.h))
- **userCtx** pointer to a user context (can be NULL)

See:

- wolfTPM2_OpenExisting
- wolfTPM2_InitWithSpdmKey
- wolfTPM2_InitWithSpdmPsk
- wolfTPM2_Test
- TPM2_Init

Return:

- TPM_RC_SUCCESS: successful
- TPM_RC_FAILURE: generic failure (check TPM IO communication)
- TPM_RC_UPGRADE: TPM firmware recovery is active; call cleanup when the recovery flow is complete
- BAD_FUNC_ARG: check the provided arguments
- WOLFSPDM_E_BAD_STATE: TPM is in SPDM-only mode and no trusted identity key or PSK was supplied; use the matching SPDM init API

Example

```
int rc;
WOLFTPM2_DEV dev;

rc = wolfTPM2_Init(&dev, TPM2_IoCb, userCtx);
if (rc != TPM_RC_SUCCESS) {
    //wolfTPM2_Init failed
    goto exit;
}
```

```
WOLFTPM_API int wolfTPM2_InitWithSpdmKey(
    WOLFTPM2_DEV * dev,
    TPM2HalIoCb ioCb,
    void * userCtx,
    const byte * rspPubKey,
    word32 rspPubKeySz
```

)

Initialize a TPM and pin a trusted SPDM responder public key. A successful call establishes an authenticated SPDM session even when the TPM accepted the initial cleartext startup probe. When the TPM is already in SPDM-only identity mode, startup is retried over that secure session. The identity mode is chosen for you: the one vendor adapter the build carries, generic TCG when it carries none, and otherwise from the TPM's DID/VID. That last case needs a transport that exposes DID/VID; use wolfTPM2_InitWithSpdmKey_ex to name the mode when it does not. On failure after argument validation, the device is fully torn down, except for TPM_RC_UPGRADE, which leaves an active context for the vendor firmware-recovery flow.

Parameters:

- **dev** pointer to an empty WOLFTPM2_DEV structure
- **ioCb** function pointer to an I/O callback
- **userCtx** pointer to a user context (can be NULL)
- **rspPubKey** trusted raw P-384 responder key in X||Y format
- **rspPubKeySz** size of rspPubKey; must be 96 bytes

See:

- wolfTPM2_Init
- wolfTPM2_SpdmSetResponderPubKey
- wolfTPM2_SpdmConnect
- wolfTPM2_SpdmIsConnected

Return:

- TPM_RC_SUCCESS: successful
- TPM_RC_UPGRADE: TPM firmware recovery is active; call cleanup when the recovery flow is complete
- BAD_FUNC_ARG: invalid parameters or responder key size
- WOLFSPDM_E_BAD_STATE: SPDM identity mode cannot be initialized, or the DID/VID matches no compiled adapter
- a WOLFSPDM_E_* error if the authenticated handshake fails

```
WOLFTPM_API int wolfTPM2_InitWithSpdmKey_ex(
    WOLFTPM2_DEV * dev,
    TPM2HalIoCb ioCb,
    void * userCtx,
    const byte * rspPubKey,
    word32 rspPubKeySz,
    WOLFSPDM_MODE mode
)
```

Initialize with a trusted responder key and explicit identity mode. Use this form when the build carries more than one identity adapter and the transport does not expose TPM DID/VID. WOLFSPDM_MODE_TCG is the generic TCG identity flow and needs no vendor adapter; a vendor mode is accepted only when its adapter is compiled in, and BAD_FUNC_ARG is returned before the device is touched when it is not. WOLFSPDM_MODE_AUTO resolves from DID/VID and fails closed rather

than guessing a vendor handshake. TPM_RC_UPGRADE leaves an active context for the vendor firmware-recovery flow; call cleanup when that flow is complete.

Parameters:

- **dev** pointer to an empty WOLFTPM2_DEV structure
- **ioCb** function pointer to an I/O callback
- **userCtx** pointer to a user context (can be NULL)
- **rspPubKey** trusted raw P-384 responder key in X||Y format
- **rspPubKeySz** size of rspPubKey; must be 96 bytes
- **mode** WOLFSPDM_MODE_AUTO, WOLFSPDM_MODE_TCG, WOLFSPDM_MODE_NUVOTON, or WOLFSPDM_MODE_NATIONS

See: wolfTPM2_InitWithSpdmKey

Return:

- TPM_RC_SUCCESS: successful
- TPM_RC_UPGRADE: TPM firmware recovery is active
- BAD_FUNC_ARG: invalid parameters, or an identity mode whose vendor adapter is not compiled in
- WOLFSPDM_E_BAD_STATE: automatic vendor selection is unavailable
- a WOLFSPDM_E_* error if the authenticated handshake fails

```
WOLFTPM_API int wolfTPM2_InitWithSpdmPsk(
    WOLFTPM2_DEV * dev,
    TPM2HalIoCb ioCb,
    void * userCtx,
    const byte * psk,
    word32 pskSz,
    const byte * hint,
    word32 hintSz
)
```

Initialize a TPM and authenticate an SPDM session with a PSK. This entry point also recovers a PSK-configured TPM that is already locked in SPDM-only mode by retrying TPM2_Startup over the secure session. On failure after argument validation, the device is fully torn down, except for TPM_RC_UPGRADE, which leaves an active context for the vendor firmware-recovery flow.

Parameters:

- **dev** pointer to an empty WOLFTPM2_DEV structure
- **ioCb** function pointer to an I/O callback
- **userCtx** pointer to a user context (can be NULL)
- **psk** pre-shared key bytes
- **pskSz** size of psk; must be nonzero
- **hint** optional PSK hint
- **hintSz** size of hint

See:

- wolfTPM2_Init
- wolfTPM2_SpdmConnectPsk

Return:

- TPM_RC_SUCCESS: successful
- TPM_RC_UPGRADE: TPM firmware recovery is active; call cleanup when the recovery flow is complete
- BAD_FUNC_ARG: invalid parameters
- a WOLFSPDM_E_* error if the authenticated handshake fails

```
WOLFTPM_API int wolfTPM2_OpenExisting(
    WOLFTPM2_DEV * dev,
    TPM2HalIoCb ioCb,
    void * userCtx
)
```

Use an already initialized TPM, in its current TPM locality.

Parameters:

- **dev** pointer to an empty structure of WOLFTPM2_DEV type
- **ioCb** function pointer to a IO callback (see [hal#file-tpm-io.h))
- **userCtx** pointer to a user context (can be NULL)

See:

- wolfTPM2_Init
- wolfTPM2_Cleanup
- TPM2_Init

Return:

- TPM_RC_SUCCESS: successful
- TPM_RC_FAILURE: generic failure (check TPM IO communication)
- BAD_FUNC_ARG: check the provided arguments

```
WOLFTPM_API int wolfTPM2_Cleanup(
    WOLFTPM2_DEV * dev
)
```

Easy to use TPM and wolfcrypt deinitialization.

Parameters:

- **dev** pointer to a populated structure of WOLFTPM2_DEV type

See:

- wolfTPM2_OpenExisting
- wolfTPM2_Test
- TPM2_Init

Return:

- TPM_RC_SUCCESS: successful
- TPM_RC_FAILURE: generic failure (check TPM IO communication)
- BAD_FUNC_ARG: check the provided arguments

Note: Calls wolfTPM2_Cleanup_ex with appropriate doShutdown parameter

Example

```
int rc;

rc = wolfTPM2_Cleanup(&dev);
if (rc != TPM_RC_SUCCESS) {
    //wolfTPM2_Cleanup failed
    goto exit;
}
```

```
WOLFTPM_API int wolfTPM2_Cleanup_ex(
    WOLFTPM2_DEV * dev,
    int doShutdown
)
```

Deinitialization of a TPM (and wolfcrypt if it was used)

Parameters:

- **dev** pointer to a populated structure of WOLFTPM2_DEV type
- **doShutdown** flag value, if true a TPM2_Shutdown command will be executed

See:

- wolfTPM2_OpenExisting
- wolfTPM2_Test
- TPM2_Init

Return:

- TPM_RC_SUCCESS: successful
- TPM_RC_FAILURE: generic failure (check TPM IO communication)
- BAD_FUNC_ARG: check the provided arguments

Example

```
int rc;

//perform TPM2_Shutdown after deinitialization
rc = wolfTPM2_Cleanup_ex(&dev, 1);
if (rc != TPM_RC_SUCCESS) {
    //wolfTPM2_Cleanup_ex failed
    goto exit;
}
```

```
WOLFTPM_API int wolfTPM2_GetTpmDevId(
    WOLFTPM2_DEV * dev
)
```

Provides the device ID of a TPM.

Parameters:

- **dev** pointer to an populated structure of WOLFTPM2_DEV type

See:

- wolfTPM2_GetCapabilities
- wolfTPM2_Init

Return:

- an integer value of a valid TPM device ID
- or INVALID_DEVID if the TPM initialization could not extract DevID

Example

```
int tpmDevId;

tpmDevId = wolfTPM2_GetTpmDevId(&dev);
if (tpmDevId != INVALID_DEVID) {
    //wolfTPM2_Cleanup_ex failed
    goto exit;
}
```

```
WOLFTPM_API int wolfTPM2_SelfTest(
    WOLFTPM2_DEV * dev
)
```

Asks the TPM to perform its self test.

Parameters:

- **dev** pointer to a populated structure of WOLFTPM2_DEV type

See:

- wolfTPM2_OpenExisting
- wolfTPM2_Test
- TPM2_Init

Return:

- TPM_RC_SUCCESS: successful
- TPM_RC_FAILURE: generic failure (check TPM IO communication and TPM return code)
- BAD_FUNC_ARG: check the provided arguments

Example

```
int rc;

//perform TPM2_Shutdown after deinitialization
rc = wolfTPM2_SelfTest(&dev);
if (rc != TPM_RC_SUCCESS) {
    //wolfTPM2_SelfTest failed
    goto exit;
}
```

```
WOLFTPM_API int wolfTPM2_GetCapabilities(
    WOLFTPM2_DEV * dev,
    WOLFTPM2_CAPS * caps
)
```

Reports the available TPM capabilities.

Parameters:

- **dev** pointer to a populated structure of WOLFTPM2_DEV type
- **caps** pointer to an empty structure of WOLFTPM2_CAPS type to store the capabilities

See:

- wolfTPM2_GetTpmDevId
- wolfTPM2_SelfTest
- wolfTPM2_Init

Return:

- TPM_RC_SUCCESS: successful
- TPM_RC_FAILURE: generic failure (check TPM IO communication and TPM return code)
- BAD_FUNC_ARG: check the provided arguments

Example

```
int rc;
WOLFTPM2_CAPS caps;

//perform TPM2_Shutdown after deinitialization
rc = wolfTPM2_GetCapabilities(&dev, &caps);
if (rc != TPM_RC_SUCCESS) {
    //wolfTPM2_GetCapabilities failed
    goto exit;
}
```

```
WOLFTPM_TEST_API int wolfTPM2_ParseCapabilities(
    WOLFTPM2_CAPS * caps,
    TPML_TAGGED_TPM_PROPERTY * props
)
```

Decode a TPM property list into WOLFTPM2_CAPS.

Parameters:

- **caps** pointer to a WOLFTPM2_CAPS struct to fill
- **props** pointer to the tagged property list to decode

See: wolfTPM2_GetCapabilities

Return: TPM_RC_SUCCESS: successful

Note: Exposed so the decode can be unit tested against synthetic property lists; callers want wolfTPM2_GetCapabilities

```
WOLFTPM_API int wolfTPM2_IsAlgSupported(
    WOLFTPM2_DEV * dev,
    TPM_ALG_ID alg,
    int * isSupported
)
```

Report whether the TPM implements a given algorithm.

Parameters:

- **dev** pointer to a TPM2_DEV struct
- **alg** the algorithm identifier to test (for example TPM_ALG_SHA512)
- **isSupported** output, set to 1 if implemented by the TPM, else 0

See: wolfTPM2_GetCapabilities

Return:

- TPM_RC_SUCCESS: query completed; *isSupported is 1 or 0

- BAD_FUNC_ARG: dev or isSupported is NULL
- a TPM_RC (or other non-zero error) if the capability query fails

Note:

- Queries TPM_CAP_ALGS. Useful to skip a hash the TPM does not support (for example SHA2-512 on parts limited to SHA2-256/384) before starting a session with it.
- The result is returned through isSupported, not the return value, so a capability-query failure cannot be mistaken for "supported". On any error *isSupported is set to 0, so the call fails closed.
- This reports what the TPM implements, not what the local wolfCrypt build supports. A caller that also hashes locally (for example to precompute a policy digest) must check its own build as well.

```
WOLFTPM_API int wolfTPM2_GetHandles(
    TPM_HANDLE handle,
    TPML_HANDLE * handles
)
```

Gets a list of handles.

Parameters:

- **handle** handle to start from (example: PCR_FIRST, NV_INDEX_FIRST, HMAC_SESSION_FIRST, POLICY_SESSION_FIRST, PERMANENT_FIRST, TRANSIENT_FIRST or PERSISTENT_FIRST)
- **handles** pointer to TPML_HANDLE to return handle results (optional)

See: wolfTPM2_GetCapabilities

Return:

- 0 or greater: successful, count of handles
- TPM_RC_FAILURE: generic failure (check TPM IO communication and TPM return code)
- BAD_FUNC_ARG: check the provided arguments
- BUFFER_E: the response did not carry a handle list

Example

```
int persistent_handle_count;

// get count of persistent handles
persistent_handle_count = wolfTPM2_GetHandles(PERSISTENT_FIRST, NULL);
```

```
WOLFTPM_API int wolfTPM2_SpdmInit(
    WOLFTPM2_DEV * dev
)
```

Initialize SPDM support on a wolfTPM2 device. Allocates and configures the SPDM context using wolfSPDM. After init, call wolfTPM2_SpdmConnect to establish a secure session.

Parameters:

- **dev** pointer to a WOLFTPM2_DEV structure

Return:

- TPM_RC_SUCCESS: successful
- BAD_FUNC_ARG: invalid parameters
- MEMORY_E: memory allocation failed

```
WOLFTPM_API int wolfTPM2_SpdmConnect(
    WOLFTPM2_DEV * dev
)
```

Establish an SPDM secure session (full handshake). Uses standard SPDM flow: GET_VERSION -> GET_CAPABILITIES -> NEGOTIATE_ALGORITHMS -> KEY_EXCHANGE -> FINISH. Identity-key modes require a responder key pinned with wolfTPM2_SpdmSetResponderPubKey before this call.

Parameters:

- **dev** pointer to a WOLFTPM2_DEV structure

See: wolfTPM2_SpdmSetResponderPubKey

Return:

- TPM_RC_SUCCESS: session established
- TPM_RC_FAILURE: handshake failed
- WOLFSPDM_E_BAD_STATE: trusted responder key is not pinned

```
WOLFTPM_API int wolfTPM2_SpdmIsConnected(
    WOLFTPM2_DEV * dev
)
```

Check if an SPDM secure session is currently active.

Parameters:

- **dev** pointer to a WOLFTPM2_DEV structure

Return: 1 if connected, 0 if not

```
WOLFTPM_API word32 wolfTPM2_SpdmGetSessionId(
    WOLFTPM2_DEV * dev
)
```

Get the current SPDM session ID.

Parameters:

- **dev** pointer to a WOLFTPM2_DEV structure

Return: Session ID, or 0 if not connected

```
WOLFTPM_API int wolfTPM2_GetCapability_SPDMSessionInfo(
    WOLFTPM2_DEV * dev,
    TPML_SPDM_SESSION_INFO * spdmSessionInfo
)
```

Read the TPM's view of the active SPDM session (GetCapability TPM_CAP_SPDM_SESSION_INFO, TPM 2.0 Library v1.84).

Parameters:

- **dev** pointer to a WOLFTPM2_DEV structure
- **spdmSessionInfo** output list of requester / TPM key names

See:

- wolfTPM2_PolicyTransportSPDM
- wolfTPM2_PolicyTransportSPDMMake

Return:

- TPM_RC_SUCCESS: successful
- TPM_RC_VALUE: the TPM does not implement the capability
- BAD_FUNC_ARG: check the provided arguments

Note: The TPM returns an empty list unless the command itself arrived inside an SPDM session. The names returned are the ones a PolicyTransportSPDM policy can bind to.

```
WOLFTPM_API int wolfTPM2_SpdmDisconnect(
    WOLFTPM2_DEV * dev
)
```

Disconnect the SPDM secure session. After this, TPM commands are sent in the clear.

Parameters:

- **dev** pointer to a WOLFTPM2_DEV structure

Return: TPM_RC_SUCCESS: successful

```
WOLFTPM_API int wolfTPM2_SpdmGetVersion(
    WOLFTPM2_DEV * dev
)
```

Send the SPDM GET_VERSION handshake step. Required before any vendor-defined SPDM command (e.g. PSK provisioning, GET_STATUS).

Parameters:

- **dev** pointer to a WOLFTPM2_DEV structure

Return: TPM_RC_SUCCESS: successful

```
WOLFTPM_API int wolfTPM2_SpdmCleanup(
    WOLFTPM2_DEV * dev
)
```

Free SPDM context and resources.

Parameters:

- **dev** pointer to a WOLFTPM2_DEV structure

Return: TPM_RC_SUCCESS: successful

```
WOLFTPM_API int wolfTPM2_SpdmSetResponderPubKey(
    WOLFTPM2_DEV * dev,
    const byte * pubKey,
    word32 pubKeySz
)
```

Pin the trusted responder key before an identity-key connection.

Parameters:

- **dev** pointer to an initialized WOLFTPM2_DEV structure
- **pubKey** trusted raw P-384 responder key in X||Y format
- **pubKeySz** size of pubKey; must be 96 bytes

See:

- wolfTPM2_InitWithSpdmKey
- wolfTPM2_SpdmConnect

Return:

- WOLFSPDM_SUCCESS: successful
- BAD_FUNC_ARG: invalid device context
- WOLFSPDM_E_INVALID_ARG: invalid key pointer or size

```
WOLFTPM_API int wolfTPM2_SpdmGetPubKey(
    WOLFTPM2_DEV * dev,
    byte * pubKey,
    word32 * pubKeySz
)
```

Get the TPM's SPDM-Identity public key (shared TCG function).

Parameters:

- **dev** pointer to a WOLFTPM2_DEV structure
- **pubKey** output buffer for the public key
- **pubKeySz** in/out: buffer size / actual key size

See: wolfTPM2_SpdmSetResponderPubKey

Return:

- TPM_RC_SUCCESS: successful
- BAD_FUNC_ARG: invalid parameters

Note: This command runs in cleartext and its result is not a trust source. Use it only for diagnostics or compare it with a key obtained from a trusted provisioning channel. Do not install the returned key as its own trust anchor.

```
WOLFTPM_API int wolfTPM2_SpdmSetNuvotonMode(
    WOLFTPM2_DEV * dev
)
```

Configure for Nuvoton TPM SPDM mode. Must be called before wolfTPM2_SpdmConnect() for Nuvoton TPMs.

Parameters:

- **dev** pointer to a WOLFTPM2_DEV structure

Return: TPM_RC_SUCCESS: successful

```
WOLFTPM_API int wolfTPM2_SpdmEnable(
    WOLFTPM2_DEV * dev
)
```

Enable SPDM on the TPM via NTC2_PreConfig vendor command. Requires platform hierarchy auth. TPM must be reset after this call.

Parameters:

- **dev** pointer to a WOLFTPM2_DEV structure

Return: TPM_RC_SUCCESS: successful

```
WOLFTPM_API int wolfTPM2_SpdmDisable(
    WOLFTPM2_DEV * dev
)
```

Disable SPDM on Nuvoton TPM via NTC2_PreConfig. Sets Cfg_H bit 1 to disable SPDM. Requires TPM reset to take effect.

Parameters:

- **dev** pointer to a WOLFTPM2_DEV structure

```
WOLFTPM_API int wolfTPM2_SpdmConnectNuvoton(
    WOLFTPM2_DEV * dev,
    const byte * reqPubKey,
    word32 reqPubKeySz,
    const byte * reqPrivKey,
    word32 reqPrivKeySz
)
```

Establish Nuvoton SPDM secure session with mutual authentication. Uses Nuvoton flow: GET_VERSION -> GET_PUB_KEY -> KEY_EXCHANGE -> GIVE_PUB_KEY -> FINISH. Requires a responder key pinned with wolfTPM2_SpdmSetResponderPubKey.

Parameters:

- **dev** pointer to a WOLFTPM2_DEV structure
- **reqPubKey** host's ECDSA P-384 public key (TPMT_PUBLIC format)
- **reqPubKeySz** size of reqPubKey in bytes
- **reqPrivKey** host's ECDSA P-384 private key (raw 48 bytes)
- **reqPrivKeySz** size of reqPrivKey in bytes

See: wolfTPM2_SpdmSetResponderPubKey

Return:

- TPM_RC_SUCCESS: session established
- TPM_RC_FAILURE: handshake failed
- WOLFSPDM_E_BAD_STATE: trusted responder key is not pinned

```
WOLFTPM_API int wolfTPM2_SpdmGetStatus(
    WOLFTPM2_DEV * dev,
    WOLFSPDM_NUVOTON_STATUS * status
)
```

Get SPDM status from the TPM (GET_STS_ vendor command).

Parameters:

- **dev** pointer to a WOLFTPM2_DEV structure
- **status** output: SPDM status information

Return:

- TPM_RC_SUCCESS: successful
- BAD_FUNC_ARG: invalid parameters

```
WOLFTPM_API int wolfTPM2_SpdmSetOnlyMode(
    WOLFTPM2_DEV * dev,
    int lock
)
```

Lock or unlock SPDM-only mode. When locked, TPM only accepts commands over SPDM secure channel.

Parameters:

- **dev** pointer to a WOLFTPM2_DEV structure
- **lock** 1 to lock SPDM-only mode, 0 to unlock

Return: TPM_RC_SUCCESS: successful

```
WOLFTPM_API int wolfTPM2_SpdmSetNationsMode(
    WOLFTPM2_DEV * dev
)
```

Configure for Nations Technology NS350 SPDM mode. Must be called before wolfTPM2_SpdmConnectNations().

Parameters:

- **dev** pointer to a WOLFTPM2_DEV structure

Return: TPM_RC_SUCCESS: successful

```
WOLFTPM_API int wolfTPM2_SpdmConnectNations(
    WOLFTPM2_DEV * dev,
    const byte * reqPubKey,
    word32 reqPubKeySz,
    const byte * reqPrivKey,
    word32 reqPrivKeySz
)
```

Establish Nations SPDM secure session (identity key mode). Uses TCG flow: GET_VERSION -> GET_PUB_KEY -> KEY_EXCHANGE -> GIVE_PUB_KEY -> FINISH. Requires a responder key pinned with wolfTPM2_SpdmSetResponderPubKey.

Parameters:

- **dev** pointer to a WOLFTPM2_DEV structure
- **reqPubKey** host's ECDSA P-384 public key (TPMT_PUBLIC format, or NULL for auto-gen)
- **reqPubKeySz** size of reqPubKey in bytes
- **reqPrivKey** host's ECDSA P-384 private key (raw 48 bytes, or NULL for auto-gen)
- **reqPrivKeySz** size of reqPrivKey in bytes

See: wolfTPM2_SpdmSetResponderPubKey

Return:

- TPM_RC_SUCCESS: session established
- TPM_RC_FAILURE: handshake failed
- WOLFSPDM_E_BAD_STATE: trusted responder key is not pinned

```
WOLFTPM_API int wolfTPM2_SpdmNationsIdentityKeySet(
    WOLFTPM2_DEV * dev,
    int set
)
```

Set/unset SPDM identity key provisioning on Nations NS350. Uses vendor command TPM2_VendorSpdmIdentityKeySe (0x20000708).

Parameters:

- **dev** pointer to a WOLFTPM2_DEV structure
- **set** 1 to provision identity key, 0 to un-provision

Return:

- TPM_RC_SUCCESS: successful
- BAD_FUNC_ARG: invalid parameters

```
WOLFTPM_API int wolfTPM2_SpdmNationsGetStatus(
    WOLFTPM2_DEV * dev,
    WOLFSPDM_NATIONS_STATUS * status
)
```

Get SPDM status from Nations TPM (PSK mode).

```
WOLFTPM_API int wolfTPM2_SpdmNationsSetOnlyMode(
    WOLFTPM2_DEV * dev,
    int lock
)
```

Lock or unlock SPDM-only mode on Nations TPM (PSK mode).

```
WOLFTPM_API int wolfTPM2_SpdmNationsPskSet(
    WOLFTPM2_DEV * dev,
    const byte * psk,
    word32 pskSz
)
```

Provision PSK on Nations TPM.

```
WOLFTPM_API int wolfTPM2_SpdmNationsPskClear(
    WOLFTPM2_DEV * dev,
    const byte * clearAuth,
    word32 clearAuthSz
)
```

Clear PSK from Nations TPM. Requires ClearAuth from PSK_SET.

```
WOLFTPM_API int wolfTPM2_SpdmSetInfineonMode(
    WOLFTPM2_DEV * dev
)
```

Configure for Infineon SLB9678 SPDM (PSK mode, Algorithm Set B). Must be called before PSK provisioning vendor commands.

Parameters:

- **dev** pointer to a WOLFTPM2_DEV structure

Return: TPM_RC_SUCCESS: successful

```
WOLFTPM_API int wolfTPM2_SpdmConnectInfineonTcg(
    WOLFTPM2_DEV * dev,
    const byte * reqPubKey,
    word32 reqPubKeySz,
    const byte * reqPrivKey,
    word32 reqPrivKeySz
)
```

Establish an SLB9678 asymmetric session using the shared TCG flow. Requires firmware advertising asymmetric SPDM and a responder key pinned with wolfTPM2_SpdmSetResponderPubKey. No PSK fallback is attempted.

Parameters:

- **dev** initialized device with an SPDM context
- **reqPubKey** requester TPMT_PUBLIC, or NULL to generate a key pair
- **reqPubKeySz** public key size, or zero when generating a key pair
- **reqPrivKey** raw P-384 private key, or NULL to generate a key pair
- **reqPrivKeySz** private key size, or zero when generating a key pair

See:

- wolfTPM2_InitWithSpdmKey_ex
- wolfTPM2_SpdmConnectPsk

Return:

- TPM_RC_SUCCESS: session established
- BAD_FUNC_ARG: invalid context or requester key arguments
- WOLFSPDM_E_BAD_STATE: trusted responder key is not configured
- WOLFSPDM_E_PEER_ERROR: unsupported capabilities or peer failure

```
WOLFTPM_API int wolfTPM2_SpdmInfineonGetStatus(
    WOLFTPM2_DEV * dev,
    WOLFSPDM_INFINEON_STATUS * status
)
```

Get SPDM status from Infineon SLB9678 (PSK mode).

```
WOLFTPM_API int wolfTPM2_SpdmInfineonSetOnlyMode(
    WOLFTPM2_DEV * dev,
    int lock
)
```

Lock or unlock SPDM-only mode on Infineon SLB9678.

```
WOLFTPM_API int wolfTPM2_SpdmInfineonEnable(
    WOLFTPM2_DEV * dev,
    int enable,
    int lock
)
```

Enable or disable the SLB9678 SPDM-PSK secure channel via TPM2_SetCapability(TPM_CAP_CONFIGS, TPM_PT_CONFIG_SPDM_PSK). Requires Platform authorization configured in session slot zero. An unused slot uses empty-password authorization. Pass lock=0 to keep it reversible (enable is undone by disabling); lock=1 lifetime-locks the configuration.

Parameters:

- **dev** pointer to a WOLFTPM2_DEV structure
- **enable** 1 to enable SPDM-PSK, 0 to disable
- **lock** 1 to lifetime-lock the configuration (irreversible), 0 normally

Return: TPM_RC_SUCCESS: successful

```
WOLFTPM_API int wolfTPM2_SpdmConnectPsk(
    WOLFTPM2_DEV * dev,
    const byte * psk,
    word32 pskSz,
    const byte * hint,
    word32 hintSz
)
```

Establish SPDM secure session via the DSP0274 PSK handshake (PSK_EXCHANGE / PSK_FINISH). Vendor-neutral spec implementation.

```
WOLFTPM_API int wolfTPM2_UnsetAuth(
    WOLFTPM2_DEV * dev,
    int index
)
```

Clears one of the TPM Authorization slots, pointed by its index number.

Parameters:

- **dev** pointer to a TPM2_DEV struct
- **index** valid TPM authorization slot from 0 through MAX_SESSION_NUM - 1

See:

- wolfTPM2_SetAuth
- wolfTPM2_SetAuthPassword
- wolfTPM2_SetAuthHandle
- wolfTPM2_SetAuthSession

Return:

- TPM_RC_SUCCESS: successful
- TPM_RC_FAILURE: unable to get lock on the TPM2 Context
- BAD_FUNC_ARG: check the provided arguments

```
WOLFTPM_API int wolfTPM2_UnsetAuthSession(
    WOLFTPM2_DEV * dev,
    int index,
    WOLFTPM2_SESSION * session
)
```

Clears one of the TPM Authorization session slots, pointed by its index number and saves the nonce from the TPM so the session can continue to be used again with wolfTPM2_SetAuthSession.

Parameters:

- **dev** pointer to a TPM2_DEV struct
- **index** valid TPM authorization slot from 0 through MAX_SESSION_NUM - 1
- **session** pointer to a WOLFTPM2_SESSION struct used with wolfTPM2_StartSession and wolfTPM2_SetAuthSession

See:

- wolfTPM2_StartSession
- wolfTPM2_SetAuthSession

Return:

- TPM_RC_SUCCESS: successful
- TPM_RC_FAILURE: unable to get lock on the TPM2 Context
- BAD_FUNC_ARG: check the provided arguments

```
WOLFTPM_API int wolfTPM2_SetAuth(
    WOLFTPM2_DEV * dev,
    int index,
    TPM_HANDLE sessionHandle,
    const TPM2B_AUTH * auth,
    TPMA_SESSION sessionAttributes,
    const TPM2B_NAME * name
)
```

Sets a TPM Authorization slot using the provided index, session handle, attributes and auth.

Parameters:

- **dev** pointer to a TPM2_DEV struct
- **index** valid TPM authorization slot from 0 through MAX_SESSION_NUM - 1
- **sessionHandle** integer value of TPM_HANDLE type
- **auth** pointer to a structure of type TPM2B_AUTH containing one TPM Authorization
- **sessionAttributes** integer value of type TPMA_SESSION, selecting one or more attributes for the Session
- **name** pointer to a TPM2B_NAME structure

See:

- wolfTPM2_SetAuthPassword
- wolfTPM2_SetAuthHandle
- wolfTPM2_SetAuthSession

Return:

- TPM_RC_SUCCESS: successful
- BAD_FUNC_ARG: check the provided arguments

Note: It is recommended to use one of the other wolfTPM2 wrappers, like wolfTPM2_SetAuthPassword. Because the wolfTPM2_SetAuth wrapper provides complete control over the TPM Authorization slot for advanced use cases. In most scenarios, wolfTPM2_SetAuthHandle and SetAuthPassword are used.

```
WOLFTPM_API int wolfTPM2_SetAuthPassword(
    WOLFTPM2_DEV * dev,
    int index,
    const TPM2B_AUTH * auth
)
```

Sets a TPM Authorization slot using the provided user auth, typically a password.

Parameters:

- **dev** pointer to a TPM2_DEV struct
- **index** valid TPM authorization slot from 0 through MAX_SESSION_NUM - 1
- **auth** pointer to a structure of type TPM2B_AUTH, typically containing a TPM Key Auth

See:

- wolfTPM2_SetAuthHandle
- wolfTPM2_SetAuthSession
- wolfTPM2_SetAuth

Return:

- TPM_RC_SUCCESS: successful
- BAD_FUNC_ARG: check the provided arguments

Note: Often used for authorizing the loading and use of TPM keys, including Primary Keys

```
WOLFTPM_API int wolfTPM2_SetAuthHandle(
    WOLFTPM2_DEV * dev,
    int index,
    const WOLFTPM2_HANDLE * handle
)
```

Sets a TPM Authorization slot using the user auth associated with a wolfTPM2 Handle.

Parameters:

- **dev** pointer to a TPM2_DEV struct
- **index** valid TPM authorization slot from 0 through MAX_SESSION_NUM - 1
- **handle** pointer to a populated structure of WOLFTPM2_HANDLE type

See:

- wolfTPM2_SetAuth
- wolfTPM2_SetAuthPassword
- wolfTPM2_SetAuthHandle
- wolfTPM2_SetAuthSession

Return:

- TPM_RC_SUCCESS: successful
- BAD_FUNC_ARG: check the provided arguments

Note: This wrapper is especially useful when using a TPM key for multiple operations and TPM Authorization is required again.

```
WOLFTPM_API int wolfTPM2_SetAuthSession(
    WOLFTPM2_DEV * dev,
    int index,
    WOLFTPM2_SESSION * tpmSession,
    TPMA_SESSION sessionAttributes
)
```

Sets a TPM Authorization slot using the provided TPM session handle, index and session attributes.

Parameters:

- **dev** pointer to a TPM2_DEV struct
- **index** valid TPM authorization slot from 0 through MAX_SESSION_NUM - 1
- **tpmSession** pointer to a WOLFTPM2_SESSION struct used with wolfTPM2_StartSession and wolfTPM2_SetAuthSession
- **sessionAttributes** integer value of type TPMA_SESSION, selecting one or more attributes for the Session

See:

- wolfTPM2_SetAuth
- wolfTPM2_SetAuthPassword
- wolfTPM2_SetAuthHandle
- wolfTPM2_SetSessionHandle

Return:

- TPM_RC_SUCCESS: successful
- BAD_FUNC_ARG: check the provided arguments

Note: This wrapper is useful for configuring TPM sessions, e.g. session for parameter encryption

```
WOLFTPM_API int wolfTPM2_SetSessionHandle(
    WOLFTPM2_DEV * dev,
    int index,
    WOLFTPM2_SESSION * tpmSession
)
```

Sets a TPM Authorization slot using the provided wolfTPM2 session object.

Parameters:

- **dev** pointer to a TPM2_DEV struct
- **index** valid TPM authorization slot from 0 through MAX_SESSION_NUM - 1
- **tpmSession** pointer to a WOLFTPM2_SESSION struct used with wolfTPM2_StartSession and wolfTPM2_SetAuthSession

See:

- wolfTPM2_SetAuth
- wolfTPM2_SetAuthPassword
- wolfTPM2_SetAuthHandle
- wolfTPM2_SetAuthSession

Return:

- TPM_RC_SUCCESS: successful
- BAD_FUNC_ARG: check the provided arguments

Note: This wrapper is useful for configuring TPM sessions, e.g. session for parameter encryption

```
WOLFTPM_API int wolfTPM2_SetAuthHandleName(
    WOLFTPM2_DEV * dev,
    int index,
    const WOLFTPM2_HANDLE * handle
)
```

Updates the Name used in a TPM Session with the Name associated with wolfTPM2 Handle.

Parameters:

- **dev** pointer to a TPM2_DEV struct
- **index** valid TPM authorization slot from 0 through MAX_SESSION_NUM - 1
- **handle** pointer to a populated structure of WOLFTPM2_HANDLE type

See:

- wolfTPM2_SetAuth
- wolfTPM2_SetAuthPassword
- wolfTPM2_SetAuthHandle
- wolfTPM2_SetAuthSession

Return:

- TPM_RC_SUCCESS: successful
- BAD_FUNC_ARG: check the provided arguments

Note: Typically, this wrapper is used from another wrappers and in very specific use cases. For example, wolfTPM2_NVWriteAuth

```
WOLFTPM_API int wolfTPM2_StartSession(
    WOLFTPM2_DEV * dev,
    WOLFTPM2_SESSION * session,
    WOLFTPM2_KEY * tpmKey,
    WOLFTPM2_HANDLE * bind,
    TPM_SE sesType,
    int encDecAlg
)
```

Create a TPM session, Policy, HMAC or Trial.

Parameters:

- **dev** pointer to a TPM2_DEV struct
- **session** pointer to an empty WOLFTPM2_SESSION struct
- **tpmKey** pointer to a WOLFTPM2_KEY that will be used as a salt for the session
- **bind** pointer to a WOLFTPM2_HANDLE that will be used to make the session bounded
- **sesType** byte value, the session type (HMAC, Policy or Trial)
- **encDecAlg** integer value, specifying the algorithm in case of parameter encryption (TPM_ALG_CFB or TPM_ALG_XOR). Any value not CFB or XOR is considered NULL and parameter encryption is disabled.

See:

- wolfTPM2_SetAuthSession
- wolfTPM2_StartSession_ex

Return:

- TPM_RC_SUCCESS: successful
- BAD_FUNC_ARG: check the provided arguments

Note: This wrapper can also be used to start TPM session for parameter encryption, see wolfTPM nvram or keygen example

```
WOLFTPM_API int wolfTPM2_StartSession_ex(
    WOLFTPM2_DEV * dev,
    WOLFTPM2_SESSION * session,
    WOLFTPM2_KEY * tpmKey,
    WOLFTPM2_HANDLE * bind,
    TPM_SE sesType,
    int encDecAlg,
    TPMI_ALG_HASH authHash
)
```

Create a TPM session selecting the session auth/hash algorithm.

Parameters:

- **dev** pointer to a TPM2_DEV struct
- **session** pointer to an empty WOLFTPM2_SESSION struct
- **tpmKey** pointer to a WOLFTPM2_KEY that will be used as a salt for the session
- **bind** pointer to a WOLFTPM2_HANDLE that will be used to make the session bounded
- **sesType** byte value, the session type (HMAC, Policy or Trial)
- **encDecAlg** integer value, specifying the algorithm in case of parameter encryption (TPM_ALG_CFB or TPM_ALG_XOR). Any value not CFB or XOR is considered NULL and parameter encryption is disabled.
- **authHash** hash algorithm for the session (for example TPM_ALG_SHA256 or TPM_ALG_SHA512). TPM_ALG_NULL selects the wrapper default.

See:

- wolfTPM2_StartSession
- wolfTPM2_SetAuthSession

Return:

- TPM_RC_SUCCESS: successful
- BAD_FUNC_ARG: check the provided arguments

Note: Use this instead of wolfTPM2_StartSession when the session digest must match a non-default hash, such as SHA-512 policy digests in post-quantum environments

```
WOLFTPM_API int wolfTPM2_CreateAuthSession_EkPolicy(
    WOLFTPM2_DEV * dev,
    WOLFTPM2_SESSION * tpmSession
)
```

Creates a TPM session with Policy Secret to satisfy the default EK policy.

Parameters:

- **dev** pointer to a TPM2_DEV struct
- **tpmSession** pointer to an empty WOLFTPM2_SESSION struct

See:

- wolfTPM2_SetAuthSession
- wolfTPM2_StartSession

Return:

- TPM_RC_SUCCESS: successful
- BAD_FUNC_ARG: check the provided arguments
- TPM_RC_FAILURE: check TPM return code, check available handles, check TPM IO

Note: This wrapper can be used only if the EK authorization is not changed from default

```
WOLFTPM_API int wolfTPM2_CreateAuthSession_EkPolicy_ex(
    WOLFTPM2_DEV * dev,
    WOLFTPM2_SESSION * tpmSession,
    TPMI_ALG_HASH authHash
)
```

Creates a TPM EK policy session selecting the session auth/hash algorithm.

Parameters:

- **dev** pointer to a TPM2_DEV struct
- **tpmSession** pointer to an empty WOLFTPM2_SESSION struct
- **authHash** hash algorithm for the session (for example TPM_ALG_SHA256 or TPM_ALG_SHA512). TPM_ALG_NULL selects the wrapper default.

See:

- wolfTPM2_CreateAuthSession_EkPolicy
- wolfTPM2_StartSession_ex

Return:

- TPM_RC_SUCCESS: successful
- BAD_FUNC_ARG: check the provided arguments
- TPM_RC_FAILURE: check TPM return code, check available handles, check TPM IO

Note: This wrapper can be used only if the EK authorization is not changed from default

```
WOLFTPM_API int wolfTPM2_CreatePrimaryKey(
    WOLFTPM2_DEV * dev,
    WOLFTPM2_KEY * key,
    TPM_HANDLE primaryHandle,
    TPMT_PUBLIC * publicTemplate,
    const byte * auth,
    int authSz
)
```

Single function to prepare and create a TPM 2.0 Primary Key.

Parameters:

- **dev** pointer to a TPM2_DEV struct
- **key** pointer to an empty struct of WOLFTPM2_KEY type
- **primaryHandle** integer value, specifying one of four TPM 2.0 Primary Seeds: TPM_RH_OWNER, TPM_RH_ENDORSEMENT, TPM_RH_PLATFORM or TPM_RH_NULL
- **publicTemplate** pointer to a TPMT_PUBLIC structure populated manually or using one of the wolfTPM2_GetKeyTemplate_... wrappers

- **auth** pointer to a string constant, specifying the password authorization for the Primary Key
- **authSz** integer value, specifying the size of the password authorization, in bytes

See:

- wolfTPM2_CreateKey
- wolfTPM2_CreatePrimaryKey_ex
- wolfTPM2_GetKeyTemplate_RSA
- wolfTPM2_GetKeyTemplate_ECC

Return:

- TPM_RC_SUCCESS: successful
- TPM_RC_FAILURE: generic failure (check TPM IO and TPM return code)
- BAD_FUNC_ARG: check the provided arguments

Note: TPM 2.0 allows only asymmetric RSA or ECC primary keys. Afterwards, both symmetric and asymmetric keys can be created under a TPM 2.0 Primary Key Typically, Primary Keys are used to create Hierarchies of TPM 2.0 Keys. The TPM uses a Primary Key to wrap the other keys, signing or decrypting.

```
WOLFTPM_API int wolfTPM2_CreatePrimaryKey_ex(
    WOLFTPM2_DEV * dev,
    WOLFTPM2_PKEY * pkey,
    TPM_HANDLE primaryHandle,
    TPMT_PUBLIC * publicTemplate,
    const byte * auth,
    int authSz
)
```

Single function to prepare and create a TPM 2.0 Primary Key.

Parameters:

- **dev** pointer to a TPM2_DEV struct
- **pkey** pointer to an empty struct of WOLFTPM2_PKEY type including the creation hash and ticket.
- **primaryHandle** integer value, specifying one of four TPM 2.0 Primary Seeds: TPM_RH_OWNER, TPM_RH_ENDORSEMENT, TPM_RH_PLATFORM or TPM_RH_NULL
- **publicTemplate** pointer to a TPMT_PUBLIC structure populated manually or using one of the wolfTPM2_GetKeyTemplate_... wrappers
- **auth** pointer to a string constant, specifying the password authorization for the Primary Key
- **authSz** integer value, specifying the size of the password authorization, in bytes

See:

- wolfTPM2_CreateKey
- wolfTPM2_CreatePrimaryKey
- wolfTPM2_GetKeyTemplate_RSA
- wolfTPM2_GetKeyTemplate_ECC

Return:

- TPM_RC_SUCCESS: successful
- TPM_RC_FAILURE: generic failure (check TPM IO and TPM return code)
- BAD_FUNC_ARG: check the provided arguments

Note: TPM 2.0 allows only asymmetric RSA or ECC primary keys. Afterwards, both symmetric and asymmetric keys can be created under a TPM 2.0 Primary Key Typically, Primary Keys are used to create Hierarchies of TPM 2.0 Keys. The TPM uses a Primary Key to wrap the other keys, signing or decrypting.

```
WOLFTPM_API int wolfTPM2_ChangeAuthKey(
    WOLFTPM2_DEV * dev,
    WOLFTPM2_KEY * key,
    WOLFTPM2_HANDLE * parent,
    const byte * auth,
    int authSz
)
```

Change the authorization secret of a TPM 2.0 key.

Parameters:

- **dev** pointer to a TPM2_DEV struct
- **key** pointer to an empty struct of WOLFTPM2_KEY type
- **parent** pointer to a struct of WOLFTPM2_HANDLE type, specifying a TPM 2.0 Primary Key to be used as the parent(Storage Key)
- **auth** pointer to a string constant, specifying the password authorization of the TPM 2.0 key
- **authSz** integer value, specifying the size of the password authorization, in bytes

See:

- wolfTPM2_CreatePrimaryKey
- wolfTPM2_SetAuthHandle
- wolfTPM2_UnloadHandle

Return:

- TPM_RC_SUCCESS: successful
- TPM_RC_FAILURE: generic failure (check TPM IO and TPM return code)
- BAD_FUNC_ARG: check the provided arguments

Note: TPM does not allow the authorization secret of a Primary Key to be changed. Instead, use wolfTPM2_CreatePrimary to create the same PrimaryKey with a new auth.

```
WOLFTPM_API int wolfTPM2_CreateKey(
    WOLFTPM2_DEV * dev,
    WOLFTPM2_KEYBLOB * keyBlob,
    WOLFTPM2_HANDLE * parent,
```

```
    TPMT_PUBLIC * publicTemplate,
    const byte * auth,
    int authSz
)
```

Single function to prepare and create a TPM 2.0 Key.

Parameters:

- **dev** pointer to a TPM2_DEV struct
- **keyBlob** pointer to an empty struct of WOLFTPM2_KEYBLOB type
- **parent** pointer to a struct of WOLFTPM2_HANDLE type, specifying the a 2.0 Primary Key to be used as the parent(Storage Key)
- **publicTemplate** pointer to a TPMT_PUBLIC structure populated manually or using one of the wolfTPM2_GetKeyTemplate_... wrappers
- **auth** pointer to a string constant, specifying the password authorization for the TPM 2.0 Key
- **authSz** integer value, specifying the size of the password authorization, in bytes

See:

- wolfTPM2_LoadKey
- wolfTPM2_GetKeyTemplate_RSA
- wolfTPM2_GetKeyTemplate_ECC
- wolfTPM2_CreatePrimaryKey

Return:

- TPM_RC_SUCCESS: successful
- TPM_RC_FAILURE: generic failure (check TPM IO and TPM return code)
- BAD_FUNC_ARG: check the provided arguments

Note: This function only creates the key material and stores it into the keyblob argument. To load the key use wolfTPM2_LoadKey

```
WOLFTPM_API int wolfTPM2_LoadKey(
    WOLFTPM2_DEV * dev,
    WOLFTPM2_KEYBLOB * keyBlob,
    WOLFTPM2_HANDLE * parent
)
```

Single function to load a TPM 2.0 key.

Parameters:

- **dev** pointer to a TPM2_DEV struct
- **keyBlob** pointer to a struct of WOLFTPM2_KEYBLOB type
- **parent** pointer to a struct of WOLFTPM2_HANDLE type, specifying a TPM 2.0 Primary Key to be used as the parent(Storage Key)

See:

- wolfTPM2_CreateKey
- wolfTPM2_CreatePrimaryKey
- wolfTPM2_GetKeyTemplate_RSA
- wolfTPM2_GetKeyTemplate_ECC

Return:

- TPM_RC_SUCCESS: successful
- TPM_RC_FAILURE: generic failure (check TPM IO and TPM return code)
- BAD_FUNC_ARG: check the provided arguments

Note: To load a TPM 2.0 key its parent(Primary Key) should also be loaded prior to this operation. Primary Keys are loaded when they are created.

```
WOLFTPM_API int wolfTPM2_CreateAndLoadKey(
    WOLFTPM2_DEV * dev,
    WOLFTPM2_KEY * key,
    WOLFTPM2_HANDLE * parent,
    TPMT_PUBLIC * publicTemplate,
    const byte * auth,
    int authSz
)
```

Single function to create and load a TPM 2.0 Key in one step.

Parameters:

- **dev** pointer to a TPM2_DEV struct
- **key** pointer to an empty struct of WOLFTPM2_KEY type
- **parent** pointer to a struct of WOLFTPM2_HANDLE type, specifying a TPM 2.0 Primary Key to be used as the parent(Storage Key)
- **publicTemplate** pointer to a TPMT_PUBLIC structure populated manually or using one of the wolfTPM2_GetKeyTemplate_... wrappers
- **auth** pointer to a string constant, specifying the password authorization of the TPM 2.0 key
- **authSz** integer value, specifying the size of the password authorization, in bytes

See:

- wolfTPM2_CreateKey
- wolfTPM2_GetKeyTemplate_RSA
- wolfTPM2_GetKeyTemplate_ECC

Return:

- TPM_RC_SUCCESS: successful
- TPM_RC_FAILURE: generic failure (check TPM IO and TPM return code)
- BAD_FUNC_ARG: check the provided arguments

```
WOLFTPM_API int wolfTPM2_CreateLoadedKey(
    WOLFTPM2_DEV * dev,
    WOLFTPM2_KEYBLOB * keyBlob,
    WOLFTPM2_HANDLE * parent,
    TPMT_PUBLIC * publicTemplate,
    const byte * auth,
    int authSz
)
```

Creates and loads a key using single TPM 2.0 operation, and stores encrypted private key material.

Parameters:

- **dev** pointer to a TPM2_DEV struct
- **keyBlob** pointer to an empty struct of WOLFTPM2_KEYBLOB type, contains private key material as encrypted data
- **parent** pointer to a struct of WOLFTPM2_HANDLE type, specifying a TPM 2.0 Primary Key to be used as the parent(Storage Key)
- **publicTemplate** pointer to a TPMT_PUBLIC structure populated manually or using one of the wolfTPM2_GetKeyTemplate_... wrappers
- **auth** pointer to a string constant, specifying the password authorization of the TPM 2.0 key
- **authSz** integer value, specifying the size of the password authorization, in bytes

See:

- wolfTPM2_CreateAndLoadKey
- wolfTPM2_CreateKey
- wolfTPM2_LoadKey

Return:

- TPM_RC_SUCCESS: successful
- TPM_RC_FAILURE: generic failure (check TPM IO and TPM return code)
- BAD_FUNC_ARG: check the provided arguments

```
WOLFTPM_API int wolfTPM2_LoadPublicKey(
    WOLFTPM2_DEV * dev,
    WOLFTPM2_KEY * key,
    const TPM2B_PUBLIC * pub
)
```

Wrapper to load the public part of an external key.

Parameters:

- **dev** pointer to a TPM2_DEV struct
- **key** pointer to an empty struct of WOLFTPM2_KEY type
- **pub** pointer to a populated structure of TPM2B_PUBLIC type

See:

- wolfTPM2_LoadRsaPublicKey
- wolfTPM2_LoadEccPublicKey
- wolfTPM2_wolfTPM2_LoadPrivateKey

Return:

- TPM_RC_SUCCESS: successful
- TPM_RC_FAILURE: generic failure (check TPM IO and TPM return code)
- BAD_FUNC_ARG: check the provided arguments

Note: The key must be formatted to the format expected by the TPM, see the 'pub' argument and the alternative wrappers.

```
WOLFTPM_API int wolfTPM2_LoadPrivateKey(
    WOLFTPM2_DEV * dev,
    const WOLFTPM2_KEY * parentKey,
    WOLFTPM2_KEY * key,
    const TPM2B_PUBLIC * pub,
    TPM2B_SENSITIVE * sens
)
```

Single function to import an external private key and load it into the TPM in one step.

Parameters:

- **dev** pointer to a TPM2_DEV struct
- **parentKey** pointer to a struct of WOLFTPM2_HANDLE type (required, must be a loaded storage key)
- **key** pointer to an empty struct of WOLFTPM2_KEY type
- **pub** pointer to a populated structure of TPM2B_PUBLIC type
- **sens** pointer to a populated structure of TPM2B_SENSITIVE type

See:

- wolfTPM2_CreateKey
- wolfTPM2_GetKeyTemplate_RSA
- wolfTPM2_GetKeyTemplate_ECC

Return:

- TPM_RC_SUCCESS: successful
- TPM_RC_FAILURE: generic failure (check TPM IO and TPM return code)
- BAD_FUNC_ARG: check the provided arguments

Note: The private key material needs to be prepared in a format that the TPM expects, see the 'sens' argument

```
WOLFTPM_API int wolfTPM2_ImportPrivateKey(
    WOLFTPM2_DEV * dev,
    const WOLFTPM2_KEY * parentKey,
    WOLFTPM2_KEYBLOB * keyBlob,
    const TPM2B_PUBLIC * pub,
    TPM2B_SENSITIVE * sens
)
```

Single function to import an external private key and load it into the TPM in one step.

Parameters:

- **dev** pointer to a TPM2_DEV struct
- **parentKey** pointer to a struct of WOLFTPM2_HANDLE type (required, must be a loaded storage key)
- **keyBlob** pointer to an empty struct of WOLFTPM2_KEYBLOB type
- **pub** pointer to a populated structure of TPM2B_PUBLIC type
- **sens** pointer to a populated structure of TPM2B_SENSITIVE type

See:

- wolfTPM2_ImportRsaPrivateKey
- wolfTPM2_ImportEccPrivateKey

Return:

- TPM_RC_SUCCESS: successful
- TPM_RC_FAILURE: generic failure (check TPM IO and TPM return code)
- BAD_FUNC_ARG: check the provided arguments

Note: The primary key material needs to be prepared in a format that the TPM expects, see the 'sens' argument

```
WOLFTPM_API int wolfTPM2_LoadRsaPublicKey(
    WOLFTPM2_DEV * dev,
    WOLFTPM2_KEY * key,
    const byte * rsaPub,
    word32 rsaPubSz,
    word32 exponent
)
```

Helper function to import the public part of an external RSA key.

Parameters:

- **dev** pointer to a TPM2_DEV struct
- **key** pointer to an empty struct of WOLFTPM2_KEY type
- **rsaPub** pointer to a byte buffer containing the public key material
- **rsaPubSz** integer value of word32 type, specifying the buffer size
- **exponent** integer value of word32 type, specifying the RSA exponent

See:

- wolfTPM2_LoadRsaPublicKey_ex
- wolfTPM2_LoadPublicKey
- wolfTPM2_LoadEccPublicKey
- wolfTPM2_ReadPublicKey

Return:

- TPM_RC_SUCCESS: successful
- TPM_RC_FAILURE: generic failure (check TPM IO and TPM return code)
- BAD_FUNC_ARG: check the provided arguments

Note: Recommended for use, because it does not require TPM format of the public part

```
WOLFTPM_API int wolfTPM2_LoadRsaPublicKey_ex(
    WOLFTPM2_DEV * dev,
    WOLFTPM2_KEY * key,
    const byte * rsaPub,
    word32 rsaPubSz,
    word32 exponent,
    TPMI_ALG_RSA_SCHEME scheme,
    TPMI_ALG_HASH hashAlg
)
```

Advanced helper function to import the public part of an external RSA key.

Parameters:

- **dev** pointer to a TPM2_DEV struct
- **key** pointer to an empty struct of WOLFTPM2_KEY type
- **rsaPub** pointer to a byte buffer containing the public key material
- **rsaPubSz** integer value of word32 type, specifying the buffer size
- **exponent** integer value of word32 type, specifying the RSA exponent
- **scheme** value of TPMI_ALG_RSA_SCHEME type, specifying the RSA scheme
- **hashAlg** value of TPMI_ALG_HASH type, specifying the TPM hashing algorithm

See:

- wolfTPM2_LoadRsaPublicKey
- wolfTPM2_LoadPublicKey
- wolfTPM2_LoadEccPublicKey
- wolfTPM2_ReadPublicKey

Return:

- TPM_RC_SUCCESS: successful
- TPM_RC_FAILURE: generic failure (check TPM IO and TPM return code)
- BAD_FUNC_ARG: check the provided arguments

Note: Allows the developer to specify TPM hashing algorithm and RSA scheme

```
WOLFTPM_API int wolfTPM2_ImportRsaPrivateKey(
    WOLFTPM2_DEV * dev,
    const WOLFTPM2_KEY * parentKey,
    WOLFTPM2_KEYBLOB * keyBlob,
    const byte * rsaPub,
    word32 rsaPubSz,
    word32 exponent,
    const byte * rsaPriv,
    word32 rsaPrivSz,
    TPMI_ALG_RSA_SCHEME scheme,
    TPMI_ALG_HASH hashAlg
)
```

Import an external RSA private key.

Parameters:

- **dev** pointer to a TPM2_DEV struct
- **parentKey** pointer to a struct of WOLFTPM2_HANDLE type (required, must be a loaded storage key)
- **keyBlob** pointer to an empty struct of WOLFTPM2_KEYBLOB type
- **rsaPub** pointer to a byte buffer, containing the public part of the RSA key
- **rsaPubSz** integer value of word32 type, specifying the public part buffer size
- **exponent** integer value of word32 type, specifying the RSA exponent
- **rsaPriv** pointer to a byte buffer, containing the private material of the RSA key
- **rsaPrivSz** integer value of word32 type, specifying the private material buffer size
- **scheme** value of TPMI_ALG_RSA_SCHEME type, specifying the RSA scheme
- **hashAlg** integer value of TPMI_ALG_HASH type, specifying a supported TPM 2.0 hash algorithm

See:

- wolfTPM2_ImportRsaPrivateKeySeed
- wolfTPM2_LoadRsaPrivateKey
- wolfTPM2_LoadRsaPrivateKey_ex
- wolfTPM2_LoadPrivateKey

Return:

- TPM_RC_SUCCESS: successful
- TPM_RC_FAILURE: generic failure (check TPM IO and TPM return code)
- BAD_FUNC_ARG: check the provided arguments
- BUFFER_E: arguments size is larger than what the TPM buffers allow

Note: The imported key is created with both sign and decrypt usage. To import a single-purpose key, call wolfTPM2_ImportRsaPrivateKeySeed with a restricted objectAttributes set.

```
WOLFTPM_API int wolfTPM2_ImportRsaPrivateKeySeed(
    WOLFTPM2_DEV * dev,
    const WOLFTPM2_KEY * parentKey,
    WOLFTPM2_KEYBLOB * keyBlob,
    const byte * rsaPub,
    word32 rsaPubSz,
    word32 exponent,
    const byte * rsaPriv,
    word32 rsaPrivSz,
    TPMI_ALG_RSA_SCHEME scheme,
    TPMI_ALG_HASH hashAlg,
    TPMA_OBJECT attributes,
    byte * seed,
    word32 seedSz
)
```

Import an external RSA private key with custom seed.

Parameters:

- **dev** pointer to a TPM2_DEV struct
- **parentKey** pointer to a struct of WOLFTPM2_HANDLE type (required, must be a loaded storage key)
- **keyBlob** pointer to an empty struct of WOLFTPM2_KEYBLOB type
- **rsaPub** pointer to a byte buffer, containing the public part of the RSA key
- **rsaPubSz** integer value of word32 type, specifying the public part buffer size
- **exponent** integer value of word32 type, specifying the RSA exponent
- **rsaPriv** pointer to a byte buffer, containing the private material of the RSA key
- **rsaPrivSz** integer value of word32 type, specifying the private material buffer size
- **scheme** value of TPMI_ALG_RSA_SCHEME type, specifying the RSA scheme
- **hashAlg** integer value of TPMI_ALG_HASH type, specifying a supported TPM 2.0 hash algorithm
- **attributes** integer value of TPMA_OBJECT type, can contain one or more attributes, e.g. TPMA_OBJECT_fixedTPM (or 0 to automatically populate)
- **seedSz** Optional (use NULL) or supply a custom seed for KDF
- **seed** Size of the seed (use 32 bytes for SHA2-256)

See:

- wolfTPM2_ImportRsaPrivateKey
- wolfTPM2_LoadRsaPrivateKey
- wolfTPM2_LoadRsaPrivateKey_ex
- wolfTPM2_LoadPrivateKey

Return:

- TPM_RC_SUCCESS: successful
- TPM_RC_FAILURE: generic failure (check TPM IO and TPM return code)
- BAD_FUNC_ARG: check the provided arguments
- BUFFER_E: arguments size is larger than what the TPM buffers allow

```
WOLFTPM_API int wolfTPM2_LoadRsaPrivateKey(
    WOLFTPM2_DEV * dev,
    const WOLFTPM2_KEY * parentKey,
    WOLFTPM2_KEY * key,
    const byte * rsaPub,
    word32 rsaPubSz,
    word32 exponent,
    const byte * rsaPriv,
    word32 rsaPrivSz
)
```

Helper function to import and load an external RSA private key in one step.

Parameters:

- **dev** pointer to a TPM2_DEV struct
- **parentKey** pointer to a struct of WOLFTPM2_HANDLE type (required, must be a loaded storage key)
- **key** pointer to an empty struct of WOLFTPM2_KEY type
- **rsaPub** pointer to a byte buffer, containing the public part of the RSA key
- **rsaPubSz** integer value of word32 type, specifying the public part buffer size
- **exponent** integer value of word32 type, specifying the RSA exponent
- **rsaPriv** pointer to a byte buffer, containing the private material of the RSA key
- **rsaPrivSz** integer value of word32 type, specifying the private material buffer size

See:

- wolfTPM2_ImportRsaPrivateKey
- wolfTPM2_LoadRsaPrivateKey_ex
- wolfTPM2_LoadPrivateKey

Return:

- TPM_RC_SUCCESS: successful
- TPM_RC_FAILURE: generic failure (check TPM IO and TPM return code)
- BAD_FUNC_ARG: check the provided arguments

```
WOLFTPM_API int wolfTPM2_LoadRsaPrivateKey_ex(
    WOLFTPM2_DEV * dev,
    const WOLFTPM2_KEY * parentKey,
    WOLFTPM2_KEY * key,
    const byte * rsaPub,
    word32 rsaPubSz,
    word32 exponent,
    const byte * rsaPriv,
    word32 rsaPrivSz,
    TPMI_ALG_RSA_SCHEME scheme,
    TPMI_ALG_HASH hashAlg
)
```

Advanced helper function to import and load an external RSA private key in one step.

Parameters:

- **dev** pointer to a TPM2_DEV struct
- **parentKey** pointer to a struct of WOLFTPM2_HANDLE type (required, must be a loaded storage key)
- **key** pointer to an empty struct of WOLFTPM2_KEY type
- **rsaPub** pointer to a byte buffer, containing the public part of the RSA key
- **rsaPubSz** integer value of word32 type, specifying the public part buffer size
- **exponent** integer value of word32 type, specifying the RSA exponent
- **rsaPriv** pointer to a byte buffer, containing the private material of the RSA key
- **rsaPrivSz** integer value of word32 type, specifying the private material buffer size
- **scheme** value of TPMI_ALG_RSA_SCHEME type, specifying the RSA scheme
- **hashAlg** value of TPMI_ALG_HASH type, specifying the TPM hashing algorithm

See:

- wolfTPM2_LoadRsaPrivateKey
- wolfTPM2_LoadPrivateKey
- wolfTPM2_ImportRsaPrivateKey
- wolfTPM2_LoadEccPrivateKey
- wolfTPM2_SetKeyAuthPassword

Return:

- TPM_RC_SUCCESS: successful
- TPM_RC_FAILURE: generic failure (check TPM IO and TPM return code)
- BAD_FUNC_ARG: check the provided arguments

Warning: If key->handle.auth is empty the key is loaded with empty authorization and any caller holding the handle can use it. Set key->handle.auth (e.g. via wolfTPM2_SetKeyAuthPassword) before loading to require a password.

```
WOLFTPM_API int wolfTPM2_LoadEccPublicKey(
    WOLFTPM2_DEV * dev,
    WOLFTPM2_KEY * key,
    int curveId,
    const byte * eccPubX,
    word32 eccPubXSz,
    const byte * eccPubY,
    word32 eccPubYSz
)
```

Helper function to import the public part of an external ECC key.

Parameters:

- **dev** pointer to a TPM2_DEV struct
- **key** pointer to an empty struct of WOLFTPM2_KEY type
- **curveId** integer value, one of the accepted TPM_ECC_CURVE values
- **eccPubX** pointer to a byte buffer containing the public material of point X

- **eccPubXSz** integer value of word32 type, specifying the point X buffer size
- **eccPubY** pointer to a byte buffer containing the public material of point Y
- **eccPubYSz** integer value of word32 type, specifying the point Y buffer size

See:

- wolfTPM2_LoadPublicKey
- wolfTPM2_LoadRsaPublicKey
- wolfTPM2_ReadPublicKey
- wolfTPM2_LoadEccPrivateKey

Return:

- TPM_RC_SUCCESS: successful
- TPM_RC_FAILURE: generic failure (check TPM IO and TPM return code)
- BAD_FUNC_ARG: check the provided arguments

Note: Recommended for use, because it does not require TPM format of the public part

```
WOLFTPM_API int wolfTPM2_LoadEccPublicKey_ex(
    WOLFTPM2_DEV * dev,
    WOLFTPM2_KEY * key,
    int curveId,
    const byte * eccPubX,
    word32 eccPubXSz,
    const byte * eccPubY,
    word32 eccPubYSz,
    TPMI_ALG_ECC_SCHEME scheme,
    TPMI_ALG_HASH hashAlg,
    TPMA_OBJECT objectAttributes
)
```

Variant of wolfTPM2_LoadEccPublicKey that lets the caller select the signing scheme, hash algorithm, and object attributes - required for loading ECDH peer keys (decrypt attribute) or non-default scheme/hash combinations.

Parameters:

- **dev** pointer to a TPM2_DEV struct
- **key** pointer to an empty struct of WOLFTPM2_KEY type
- **curveId** integer value, one of the accepted TPM_ECC_CURVE values
- **eccPubX** pointer to a byte buffer containing the public material of point X
- **eccPubXSz** integer value of word32 type, specifying the point X buffer size
- **eccPubY** pointer to a byte buffer containing the public material of point Y
- **eccPubYSz** integer value of word32 type, specifying the point Y buffer size
- **scheme** ECC signing scheme (e.g. TPM_ALG_ECDSA, TPM_ALG_ECDH, TPM_ALG_NULL). Use TPM_ALG_ECDH for ECDH peer keys
- **hashAlg** hash algorithm bound to scheme; ignored when scheme is TPM_ALG_NULL
- **objectAttributes** TPMA_OBJECT bitmask. Use TPMA_OBJECT_decrypt for ECDH peer keys, TPMA_OBJECT_sign for verification keys; typically OR'd with TPMA_OBJECT_userWithAuth and TPMA_OBJECT_noDA

See:

- wolfTPM2_LoadEccPublicKey
- wolfTPM2_LoadRsaPublicKey_ex

Return:

- TPM_RC_SUCCESS: successful
- BAD_FUNC_ARG: check the provided arguments, or scheme is TPM_ALG_ECDAA (use wolfTPM2_LoadPublicKey directly for ECDAA)
- BUFFER_E: caller buffer is too small for ECC point coordinates

```
WOLFTPM_API int wolfTPM2_ImportEccPrivateKey(
    WOLFTPM2_DEV * dev,
    const WOLFTPM2_KEY * parentKey,
    WOLFTPM2_KEYBLOB * keyBlob,
    int curveId,
    const byte * eccPubX,
    word32 eccPubXSz,
    const byte * eccPubY,
    word32 eccPubYSz,
    const byte * eccPriv,
    word32 eccPrivSz
)
```

Helper function to import the private material of an external ECC key.

Parameters:

- **dev** pointer to a TPM2_DEV struct
- **parentKey** pointer to a struct of WOLFTPM2_HANDLE type (required, must be a loaded storage key)
- **keyBlob** pointer to an empty struct of WOLFTPM2_KEYBLOB type
- **curveId** integer value, one of the accepted TPM_ECC_CURVE values
- **eccPubX** pointer to a byte buffer containing the public material of point X
- **eccPubXSz** integer value of word32 type, specifying the point X buffer size
- **eccPubY** pointer to a byte buffer containing the public material of point Y
- **eccPubYSz** integer value of word32 type, specifying the point Y buffer size
- **eccPriv** pointer to a byte buffer containing the private material
- **eccPrivSz** integer value of word32 type, specifying the private material size

See:

- wolfTPM2_ImportEccPrivateKeySeed
- wolfTPM2_LoadEccPrivateKey
- wolfTPM2_LoadEccPrivateKey_ex
- wolfTPM2_LoadPrivateKey

Return:

- TPM_RC_SUCCESS: successful
- TPM_RC_FAILURE: generic failure (check TPM IO and TPM return code)
- BAD_FUNC_ARG: check the provided arguments

Note: The imported key is created with both sign and decrypt usage. To import a single-purpose key, call wolfTPM2_ImportEccPrivateKeySeed with a restricted objectAttributes set.

```
WOLFTPM_API int wolfTPM2_ImportEccPrivateKeySeed(
    WOLFTPM2_DEV * dev,
    const WOLFTPM2_KEY * parentKey,
    WOLFTPM2_KEYBLOB * keyBlob,
    int curveId,
    const byte * eccPubX,
    word32 eccPubXSz,
    const byte * eccPubY,
    word32 eccPubYSz,
    const byte * eccPriv,
    word32 eccPrivSz,
    TPMA_OBJECT attributes,
    byte * seed,
    word32 seedSz
)
```

Helper function to import the private material of an external ECC key.

Parameters:

- **dev** pointer to a TPM2_DEV struct
- **parentKey** pointer to a struct of WOLFTPM2_HANDLE type (required, must be a loaded storage key)
- **keyBlob** pointer to an empty struct of WOLFTPM2_KEYBLOB type
- **curveId** integer value, one of the accepted TPM_ECC_CURVE values
- **eccPubX** pointer to a byte buffer containing the public material of point X
- **eccPubXSz** integer value of word32 type, specifying the point X buffer size
- **eccPubY** pointer to a byte buffer containing the public material of point Y
- **eccPubYSz** integer value of word32 type, specifying the point Y buffer size
- **eccPriv** pointer to a byte buffer containing the private material
- **eccPrivSz** integer value of word32 type, specifying the private material size
- **attributes** integer value of TPMA_OBJECT type, can contain one or more attributes, e.g. TPMA_OBJECT_fixedTPM (or 0 to automatically populate)
- **seedSz** Optional (use NULL) or supply a custom seed for KDF
- **seed** Size of the seed (use 32 bytes for SHA2-256)

See:

- wolfTPM2_ImportEccPrivateKey
- wolfTPM2_LoadEccPrivateKey
- wolfTPM2_LoadEccPrivateKey_ex
- wolfTPM2_LoadPrivateKey

Return:

- TPM_RC_SUCCESS: successful
- TPM_RC_FAILURE: generic failure (check TPM IO and TPM return code)
- BAD_FUNC_ARG: check the provided arguments

```
WOLFTPM_API int wolfTPM2_LoadEccPrivateKey(
    WOLFTPM2_DEV * dev,
    const WOLFTPM2_KEY * parentKey,
    WOLFTPM2_KEY * key,
    int curveId,
    const byte * eccPubX,
    word32 eccPubXSz,
    const byte * eccPubY,
    word32 eccPubYSz,
    const byte * eccPriv,
    word32 eccPrivSz
)
```

Helper function to import and load an external ECC private key in one step.

Parameters:

- **dev** pointer to a TPM2_DEV struct
- **parentKey** pointer to a struct of WOLFTPM2_HANDLE type (required, must be a loaded storage key)
- **key** pointer to an empty struct of WOLFTPM2_KEY type
- **curveId** integer value, one of the accepted TPM_ECC_CURVE values
- **eccPubX** pointer to a byte buffer containing the public material of point X
- **eccPubXSz** integer value of word32 type, specifying the point X buffer size
- **eccPubY** pointer to a byte buffer containing the public material of point Y
- **eccPubYSz** integer value of word32 type, specifying the point Y buffer size
- **eccPriv** pointer to a byte buffer containing the private material
- **eccPrivSz** integer value of word32 type, specifying the private material size

See:

- wolfTPM2_ImportEccPrivateKey
- wolfTPM2_LoadEccPublicKey
- wolfTPM2_LoadPrivateKey
- wolfTPM2_SetKeyAuthPassword

Return:

- TPM_RC_SUCCESS: successful
- TPM_RC_FAILURE: generic failure (check TPM IO and TPM return code)
- BAD_FUNC_ARG: check the provided arguments

Warning: If key->handle.auth is empty the key is loaded with empty authorization and any caller holding the handle can use it. Set key->handle.auth (e.g. via wolfTPM2_SetKeyAuthPassword) before loading to require a password.

```
WOLFTPM_API int wolfTPM2_ReadPublicKey(
    WOLFTPM2_DEV * dev,
    WOLFTPM2_KEY * key,
    const TPM_HANDLE handle
)
```

Helper function to receive the public part of a loaded TPM object using its handle.

Parameters:

- **dev** pointer to a TPM2_DEV struct
- **key** pointer to an empty struct of WOLFTPM2_KEY type
- **handle** integer value of TPM_HANDLE type, specifying handle of a loaded TPM object

See:

- wolfTPM2_LoadRsaPublicKey
- wolfTPM2_LoadEccPublicKey
- wolfTPM2_LoadPublicKey

Return:

- TPM_RC_SUCCESS: successful
- TPM_RC_FAILURE: generic failure (check TPM IO and TPM return code)
- BAD_FUNC_ARG: check the provided arguments

Note: The public part of a TPM symmetric keys contains just TPM meta data

```
WOLFTPM_API int wolfTPM2_CreateKeySeal(
    WOLFTPM2_DEV * dev,
    WOLFTPM2_KEYBLOB * keyBlob,
    WOLFTPM2_HANDLE * parent,
    TPMT_PUBLIC * publicTemplate,
    const byte * auth,
    int authSz,
    const byte * sealData,
    int sealSize
)
```

Using this wrapper a secret can be sealed inside a TPM 2.0 Key.

Parameters:

- **dev** pointer to a TPM2_DEV struct
- **keyBlob** pointer to an empty struct of WOLFTPM2_KEYBLOB type
- **parent** pointer to a struct of WOLFTPM2_HANDLE type, specifying the a 2.0 Primary Key to be used as the parent(Storage Key)
- **publicTemplate** pointer to a TPMT_PUBLIC structure populated using one of the wolfTPM2_GetKeyTemplate_Key
- **auth** pointer to a string constant, specifying the password authorization for the TPM 2.0 Key
- **authSz** integer value, specifying the size of the password authorization, in bytes
- **sealData** pointer to a byte buffer, containing the secret(user data) to be sealed

- **sealSize** integer value, specifying the size of the seal buffer, in bytes

See:

- wolfTPM2_GetKeyTemplate_KeySeal
- TPM2_Unseal
- wolfTPM2_CreatePrimary

Return:

- TPM_RC_SUCCESS: successful
- TPM_RC_FAILURE: generic failure (check TPM IO and TPM return code)
- BAD_FUNC_ARG: check the provided arguments

Note: The secret size can not be larger than 128 bytes

```
WOLFTPM_API int wolfTPM2_CreateKeySeal_ex(
    WOLFTPM2_DEV * dev,
    WOLFTPM2_KEYBLOB * keyBlob,
    WOLFTPM2_HANDLE * parent,
    TPMT_PUBLIC * publicTemplate,
    const byte * auth,
    int authSz,
    TPM_ALG_ID pcrAlg,
    byte * pcrArray,
    word32 pcrArraySz,
    const byte * sealData,
    int sealSize
)
```

Using this wrapper a secret can be sealed inside a TPM 2.0 Key with pcr selection.

Parameters:

- **dev** pointer to a WOLFTPM2_DEV struct
- **keyBlob** pointer to an empty struct of WOLFTPM2_KEYBLOB type
- **parent** pointer to a struct of WOLFTPM2_HANDLE type, specifying the a 2.0 Primary Key to be used as the parent(Storage Key)
- **publicTemplate** pointer to a TPMT_PUBLIC structure populated using one of the wolfTPM2_GetKeyTemplate_Key
- **auth** pointer to a string constant, specifying the password authorization for the TPM 2.0 Key
- **authSz** integer value, specifying the size of the password authorization, in bytes
- **pcrAlg** hash algorithm to use when calculating pcr digest
- **pcrArray** optional array of pcrs to be used when creating the tpm object
- **pcrArraySz** length of the pcrArray
- **sealData** pointer to a byte buffer, containing the secret(user data) to be sealed
- **sealSize** integer value, specifying the size of the seal buffer, in bytes

See:

- wolfTPM2_GetKeyTemplate_KeySeal
- TPM2_Unseal

- wolfTPM2_CreatePrimary

Return:

- TPM_RC_SUCCESS: successful
- TPM_RC_FAILURE: generic failure (check TPM IO and TPM return code)
- BAD_FUNC_ARG: check the provided arguments

Note: The secret size can not be larger than 128 bytes

```
WOLFTPM_API int wolfTPM2_ComputeName(
    const TPM2B_PUBLIC * pub,
    TPM2B_NAME * out
)
```

Helper function to generate a hash of the public area of an object in the format expected by the TPM.

Parameters:

- **pub** pointer to a populated structure of TPM2B_PUBLIC type, containing the public area of a TPM object
- **out** pointer to an empty struct of TPM2B_NAME type, to store the computed name

See: wolfTPM2_ImportPrivateKey

Return:

- TPM_RC_SUCCESS: successful
- TPM_RC_FAILURE: generic failure (check TPM IO and TPM return code)
- BAD_FUNC_ARG: check the provided arguments

Note: Computed TPM name includes hash of the TPM_ALG_ID and the public are of the object

```
WOLFTPM_API int wolfTPM2_SensitiveToPrivate(
    TPM2B_SENSITIVE * sens,
    TPM2B_PRIVATE * priv,
    TPMI_ALG_HASH nameAlg,
    TPM2B_NAME * name,
    const WOLFTPM2_KEY * parentKey,
    TPMT_SYM_DEF_OBJECT * sym,
    TPM2B_DATA * symSeed
)
```

Helper function to convert TPM2B_SENSITIVE.

Parameters:

- **sens** pointer to a correctly populated structure of TPM2B_SENSITIVE type
- **priv** pointer to an empty struct of TPM2B_PRIVATE type
- **nameAlg** integer value of TPMI_ALG_HASH type, specifying a valid TPM2 hashing algorithm

- **name** pointer to a TPM2B_NAME structure
- **parentKey** pointer to a WOLFTPM2_KEY structure, specifying a parentKey, if it exists
- **sym** pointer to a structure of TPMT_SYM_DEF_OBJECT type
- **symSeed** pointer to a structure of derived secret (RSA=random, ECC=ECDHE)

See: wolfTPM2_ImportPrivateKey

Return:

- TPM_RC_SUCCESS: successful
- TPM_RC_FAILURE: generic failure (check TPM IO and TPM return code)
- BAD_FUNC_ARG: check the provided arguments

```
WOLFTPM_API word32 wolfTPM2_RsaKey_Exponent(
    const byte * e,
    word32 eSz
)
```

Converts a big-endian byte array to a native word32 value. Used for RSA exponent conversion from ASN.1/DER format.

Parameters:

- **e** pointer to big-endian byte array
- **eSz** size of the byte array (max 4 bytes)

Return: word32 value in native byte order

```
WOLFTPM_API int wolfTPM2_ImportPrivateKeyBuffer(
    WOLFTPM2_DEV * dev,
    const WOLFTPM2_KEY * parentKey,
    int keyType,
    WOLFTPM2_KEYBLOB * keyBlob,
    int encodingType,
    const char * input,
    word32 inSz,
    const char * pass,
    TPMA_OBJECT objectAttributes,
    byte * seed,
    word32 seedSz
)
```

Helper function to import PEM/DER or RSA/ECC private key.

Parameters:

- **dev** pointer to a TPM2_DEV struct
- **keyType** The type of key (TPM_ALG_RSA or TPM_ALG_ECC)
- **parentKey** pointer to a WOLFTPM2_KEY struct, pointing to a Primary Key or TPM Hierarchy
- **keyBlob** pointer to a struct of WOLFTPM2_KEYBLOB type, to import the private key to

- **encodingType** ENCODING_TYPE_PEM or ENCODING_TYPE_ASN1 (DER)
- **input** buffer holding the rsa pem
- **inSz** length of the input pem buffer
- **pass** optional password of the key
- **objectAttributes** integer value of TPMA_OBJECT type, can contain one or more attributes, e.g. TPMA_OBJECT_fixedTPM
- **seedSz** Optional (use NULL) or supply a custom seed for KDF
- **seed** Size of the seed (use 32 bytes for SHA2-256)

Return:

- TPM_RC_SUCCESS: successful
- TPM_RC_FAILURE: generic failure (check TPM IO and TPM return code)
- BAD_FUNC_ARG: check the provided arguments

```
WOLFTPM_API int wolfTPM2_ImportPublicKeyBuffer(
    WOLFTPM2_DEV * dev,
    int keyType,
    WOLFTPM2_KEY * key,
    int encodingType,
    const char * input,
    word32 inSz,
    TPMA_OBJECT objectAttributes
)
```

Helper function to import PEM/DER formatted RSA/ECC public key.

Parameters:

- **dev** pointer to a TPM2_DEV struct
- **keyType** The type of key (TPM_ALG_RSA or TPM_ALG_ECC)
- **key** pointer to a struct of WOLFTPM2_KEY type, to import the public key to
- **encodingType** ENCODING_TYPE_PEM or ENCODING_TYPE_ASN1 (DER)
- **input** buffer holding the rsa pem
- **inSz** length of the input pem buffer
- **objectAttributes** integer value of OR'd TPMA_OBJECT_* types

Return:

- TPM_RC_SUCCESS: successful - populates key->pub
- TPM_RC_FAILURE: generic failure (check TPM IO and TPM return code)
- BAD_FUNC_ARG: check the provided arguments

```
WOLFTPM_API int wolfTPM2_ExportPublicKeyBuffer(
    WOLFTPM2_DEV * dev,
    WOLFTPM2_KEY * tpmKey,
    int encodingType,
    byte * out,
    word32 * outSz
```

)

Helper function to export a TPM RSA/ECC public key with PEM/DER formatting.

Parameters:

- **dev** pointer to a TPM2_DEV struct
- **tpmKey** pointer to a WOLFTPM2_KEY with populated key
- **encodingType** ENCODING_TYPE_PEM or ENCODING_TYPE_ASN1 (DER)
- **out** buffer to export public key
- **outSz** pointer to length of the out buffer

Return:

- TPM_RC_SUCCESS: successful - populates key->pub
- TPM_RC_FAILURE: generic failure (check TPM IO and TPM return code)
- BUFFER_E: insufficient space in provided buffer
- BAD_FUNC_ARG: check the provided arguments

```
WOLFTPM_API int wolfTPM2_RsaPrivateKeyImportDer(
    WOLFTPM2_DEV * dev,
    const WOLFTPM2_KEY * parentKey,
    WOLFTPM2_KEYBLOB * keyBlob,
    const byte * input,
    word32 inSz,
    TPMI_ALG_RSA_SCHEME scheme,
    TPMI_ALG_HASH hashAlg
)
```

Helper function to import Der rsa key directly.

Parameters:

- **dev** pointer to a TPM2_DEV struct
- **parentKey** pointer to a WOLFTPM2_KEY struct, pointing to a Primary Key or TPM Hierarchy
- **keyBlob** pointer to a struct of WOLFTPM2_KEYBLOB type, to import the rsa key to
- **input** buffer holding the rsa der
- **inSz** length of the input der buffer
- **scheme** value of TPMI_ALG_RSA_SCHEME type, specifying the RSA scheme
- **hashAlg** value of TPMI_ALG_HASH type, specifying the TPM hashing algorithm

Return:

- TPM_RC_SUCCESS: successful
- TPM_RC_FAILURE: generic failure (check TPM IO and TPM return code)
- BAD_FUNC_ARG: check the provided arguments

```
WOLFTPM_API int wolfTPM2_RsaPrivateKeyImportPem(
    WOLFTPM2_DEV * dev,
    const WOLFTPM2_KEY * parentKey,
    WOLFTPM2_KEYBLOB * keyBlob,
    const char * input,
    word32 inSz,
    char * pass,
    TPMI_ALG_RSA_SCHEME scheme,
    TPMI_ALG_HASH hashAlg
)
```

Helper function to import Pem rsa key directly.

Parameters:

- **dev** pointer to a TPM2_DEV struct
- **parentKey** pointer to a WOLFTPM2_KEY struct, pointing to a Primary Key or TPM Hierarchy
- **keyBlob** pointer to a struct of WOLFTPM2_KEYBLOB type, to import the rsa key to
- **input** buffer holding the rsa pem
- **inSz** length of the input pem buffer
- **pass** optional password of the key
- **scheme** value of TPMI_ALG_RSA_SCHEME type, specifying the RSA scheme
- **hashAlg** value of TPMI_ALG_HASH type, specifying the TPM hashing algorithm

Return:

- TPM_RC_SUCCESS: successful
- TPM_RC_FAILURE: generic failure (check TPM IO and TPM return code)
- BAD_FUNC_ARG: check the provided arguments

```
WOLFTPM_API int wolfTPM2_RsaKey_TpmToWolf(
    WOLFTPM2_DEV * dev,
    WOLFTPM2_KEY * tpmKey,
    RsaKey * wolfKey
)
```

Extract a RSA TPM key and convert it to a wolfcrypt key.

Parameters:

- **dev** pointer to a TPM2_DEV struct
- **tpmKey** pointer to a struct of WOLFTPM2_KEY type, holding a TPM key
- **wolfKey** pointer to an empty struct of RsaKey type, to store the converted key

See:

- wolfTPM2_RsaKey_WolfToTpm
- wolfTPM2_RsaKey_WolfToTpm_ex

Return:

- TPM_RC_SUCCESS: successful
- TPM_RC_FAILURE: generic failure (check TPM IO and TPM return code)
- BAD_FUNC_ARG: check the provided arguments

```
WOLFTPM_API int wolfTPM2_RsaKey_TpmToPemPub(
    WOLFTPM2_DEV * dev,
    WOLFTPM2_KEY * keyBlob,
    byte * pem,
    word32 * pemSz
)
```

Convert a public RSA TPM key to PEM format public key. Note: This API is a wrapper around wolfTPM2_ExportPublicKeyBuffer.

Parameters:

- **dev** pointer to a TPM2_DEV struct
- **keyBlob** pointer to a struct of WOLFTPM2_KEY type, holding a TPM key
- **pem** pointer to an array of byte type, used as temporary storage for PEM conversation
- **pemSz** pointer to integer variable, to store the used buffer size

See:

- wolfTPM2_ExportPublicKeyBuffer
- wolfTPM2_RsaKey_TpmToWolf
- wolfTPM2_RsaKey_WolfToTpm

Return:

- TPM_RC_SUCCESS: successful
- TPM_RC_FAILURE: generic failure (check TPM IO and TPM return code)
- BAD_FUNC_ARG: check the provided arguments

```
WOLFTPM_API int wolfTPM2_RsaKey_WolfToTpm(
    WOLFTPM2_DEV * dev,
    RsaKey * wolfKey,
    WOLFTPM2_KEY * tpmKey
)
```

Import a RSA wolfcrypt key into the TPM.

Parameters:

- **dev** pointer to a TPM2_DEV struct
- **wolfKey** pointer to a struct of RsaKey type, holding a wolfcrypt key
- **tpmKey** pointer to an empty struct of WOLFTPM2_KEY type, to hold the imported TPM key

See: wolfTPM2_RsaKey_TpmToWolf

Return:

- TPM_RC_SUCCESS: successful
- TPM_RC_FAILURE: generic failure (check TPM IO and TPM return code)
- BAD_FUNC_ARG: check the provided arguments

Note: Allows the use of externally generated keys by wolfcrypt to be used with TPM 2.0

```
WOLFTPM_API int wolfTPM2_RsaKey_WolfToTpm_ex(
    WOLFTPM2_DEV * dev,
    const WOLFTPM2_KEY * parentKey,
    RsaKey * wolfKey,
    WOLFTPM2_KEY * tpmKey
)
```

Import a RSA wolfcrypt key into the TPM under a specific Primary Key or Hierarchy.

Parameters:

- **dev** pointer to a TPM2_DEV struct
- **parentKey** pointer to a WOLFTPM2_KEY struct, pointing to a Primary Key or TPM Hierarchy
- **wolfKey** pointer to a struct of RsaKey type, holding a wolfcrypt key
- **tpmKey** pointer to an empty struct of WOLFTPM2_KEY type, to hold the imported TPM key

See:

- wolfTPM2_RsaKey_WolfToTpm
- wolfTPM2_RsaKey_TpmToWolf

Return:

- TPM_RC_SUCCESS: successful
- TPM_RC_FAILURE: generic failure (check TPM IO and TPM return code)
- BAD_FUNC_ARG: check the provided arguments

Note: Allows the use of wolfcrypt generated keys with wolfTPM

```
WOLFTPM_API int wolfTPM2_CreateRsaKeyBlob(
    WOLFTPM2_DEV * dev,
    const WOLFTPM2_KEY * parentKey,
    RsaKey * wolfKey,
    WOLFTPM2_KEYBLOB * tpmKey
)
```

Create an encrypted RSA key blob from a wolfCrypt key under a specific parent key.

Parameters:

- **dev** pointer to a TPM2_DEV struct
- **parentKey** pointer to a WOLFTPM2_KEY struct, pointing to a Primary Key or TPM Hierarchy
- **wolfKey** pointer to a struct of RsaKey type, holding a wolfcrypt key
- **tpmKey** pointer to an empty struct of WOLFTPM2_KEYBLOB type, to hold the encrypted key blob

See:

- wolfTPM2_LoadKey
- wolfTPM2_RsaKey_WolfToTpm_ex
- wolfTPM2_CreateEccKeyBlob

Return:

- TPM_RC_SUCCESS: successful
- TPM_RC_FAILURE: generic failure (check TPM IO and TPM return code)
- BAD_FUNC_ARG: check the provided arguments

Note: Creates an encrypted version of the key in WOLFTPM2_KEYBLOB to load the key.

```
WOLFTPM_API int wolfTPM2_RsaKey_PubPemToTpm(
    WOLFTPM2_DEV * dev,
    WOLFTPM2_KEY * tpmKey,
    const byte * pem,
    word32 pemSz
)
```

Import a PEM format public key from a file into the TPM.

Parameters:

- **dev** pointer to a TPM2_DEV struct
- **tpmKey** pointer to an empty struct of WOLFTPM2_KEY type, to hold the imported TPM key
- **pem** pointer to an array of byte type, containing a PEM formatted public key material
- **pemSz** pointer to integer variable, specifying the size of PEM key data

See:

- wolfTPM2_RsaKey_WolfToTpm
- wolfTPM2_RsaKey_TpmToPem
- wolfTPM2_RsaKey_TpmToWolf

Return:

- TPM_RC_SUCCESS: successful
- TPM_RC_FAILURE: generic failure (check TPM IO and TPM return code)

```
WOLFTPM_API int wolfTPM2_DecodeRsaDer(
    const byte * der,
    word32 derSz,
    TPM2B_PUBLIC * pub,
    TPM2B_SENSITIVE * sens,
    TPMA_OBJECT attributes
)
```

Import DER RSA private or public key into TPM public and sensitive structures. This does not make any calls to TPM hardware.

Parameters:

- **der** The der encoding of the content of the extension.
- **derSz** The size in bytes of the der encoding.
- **pub** pointer to a populated structure of TPM2B_PUBLIC type
- **sens** pointer to a populated structure of TPM2B_SENSITIVE type
- **attributes** integer value of TPMA_OBJECT type, can contain one or more attributes, e.g. TPMA_OBJECT_fixedTPM (or 0 to automatically populate)

See:

- wolfTPM2_ImportPublicKeyBuffer
- wolfTPM2_ImportPrivateKeyBuffer
- wolfTPM2_DecodeEccDer

Return:

- TPM_RC_SUCCESS: successful
- TPM_RC_FAILURE: generic failure (check TPM IO and TPM return code)

```
WOLFTPM_API int wolfTPM2_EccKey_TpmToWolf(
    WOLFTPM2_DEV * dev,
    WOLFTPM2_KEY * tpmKey,
    ecc_key * wolfKey
)
```

Extract a ECC TPM key and convert to to a wolfcrypt key.

Parameters:

- **dev** pointer to a TPM2_DEV struct
- **tpmKey** pointer to a struct of WOLFTPM2_KEY type, holding a TPM key
- **wolfKey** pointer to an empty struct of ecc_key type, to store the converted key

See:

- wolfTPM2_EccKey_WolfToTpm
- wolfTPM2_EccKey_WolfToTpm_ex

Return:

- TPM_RC_SUCCESS: successful
- TPM_RC_FAILURE: generic failure (check TPM IO and TPM return code)
- BAD_FUNC_ARG: check the provided arguments

```
WOLFTPM_API int wolfTPM2_EccKey_WolfToTpm(
    WOLFTPM2_DEV * dev,
    ecc_key * wolfKey,
    WOLFTPM2_KEY * tpmKey
)
```

Import a ECC wolfcrypt key into the TPM.

Parameters:

- **dev** pointer to a TPM2_DEV struct
- **wolfKey** pointer to a struct of ecc_key type, holding a wolfcrypt key
- **tpmKey** pointer to an empty struct of WOLFTPM2_KEY type, to hold the imported TPM key

See: wolfTPM2_EccKey_TpmToWolf

Return:

- TPM_RC_SUCCESS: successful
- TPM_RC_FAILURE: generic failure (check TPM IO and TPM return code)
- BAD_FUNC_ARG: check the provided arguments

Note: Allows the use of externally generated keys by wolfcrypt to be used with TPM 2.0

```
WOLFTPM_API int wolfTPM2_EccKey_WolfToTpm_ex(
    WOLFTPM2_DEV * dev,
    WOLFTPM2_KEY * parentKey,
    ecc_key * wolfKey,
    WOLFTPM2_KEY * tpmKey
)
```

Import ECC wolfcrypt key into the TPM under a specific Primary Key or Hierarchy.

Parameters:

- **dev** pointer to a TPM2_DEV struct
- **parentKey** pointer to a WOLFTPM2_KEY struct, pointing to a Primary Key or TPM Hierarchy
- **wolfKey** pointer to a struct of ecc_key type, holding a wolfcrypt key
- **tpmKey** pointer to an empty struct of WOLFTPM2_KEY type, to hold the imported TPM key

See:

- wolfTPM2_EccKey_WolfToTPM
- wolfTPM2_EccKey_TpmToWolf

Return:

- TPM_RC_SUCCESS: successful
- TPM_RC_FAILURE: generic failure (check TPM IO and TPM return code)
- BAD_FUNC_ARG: check the provided arguments

Note: Allows the use of wolfcrypt generated keys with wolfTPM

```
WOLFTPM_API int wolfTPM2_CreateEccKeyBlob(
    WOLFTPM2_DEV * dev,
    WOLFTPM2_KEY * parentKey,
    ecc_key * wolfKey,
    WOLFTPM2_KEYBLOB * tpmKey
)
```

Create an encrypted ECC key blob from a wolfCrypt key under a specific parent key.

Parameters:

- **dev** pointer to a TPM2_DEV struct
- **parentKey** pointer to a WOLFTPM2_KEY struct, pointing to a Primary Key or TPM Hierarchy
- **wolfKey** pointer to a struct of ecc_key type, holding a wolfcrypt key
- **tpmKey** pointer to an empty struct of WOLFTPM2_KEYBLOB type, to hold the encrypted key blob

See:

- wolfTPM2_LoadKey
- wolfTPM2_EccKey_WolfToTpm_ex
- wolfTPM2_CreateRsaKeyBlob

Return:

- TPM_RC_SUCCESS: successful
- TPM_RC_FAILURE: generic failure (check TPM IO and TPM return code)
- BAD_FUNC_ARG: check the provided arguments

Note: Creates an encrypted version of the key in WOLFTPM2_KEYBLOB to load the key.

```
WOLFTPM_API int wolfTPM2_EccKey_WolfToPubPoint(
    WOLFTPM2_DEV * dev,
    ecc_key * wolfKey,
    TPM2B_ECC_POINT * pubPoint
)
```

Import a ECC public key generated from wolfcrypt key into the TPM.

Parameters:

- **dev** pointer to a TPM2_DEV struct
- **wolfKey** pointer to a struct of ecc_key type, holding a wolfcrypt public ECC key
- **pubPoint** pointer to an empty struct of TPM2B_ECC_POINT type

See: wolfTPM2_EccKey_TpmToWolf

Return:

- TPM_RC_SUCCESS: successful
- TPM_RC_FAILURE: generic failure (check TPM IO and TPM return code)
- BAD_FUNC_ARG: check the provided arguments

Note: Allows the use of externally generated public ECC key by wolfcrypt to be used with TPM 2.0

```
WOLFTPM_API int wolfTPM2_DecodeEccDer(
    const byte * der,
    word32 derSz,
    TPM2B_PUBLIC * pub,
    TPM2B_SENSITIVE * sens,
    TPMA_OBJECT attributes
)
```

Import DER ECC private or public key into TPM public and sensitive structures. This does not make any calls to TPM hardware.

Parameters:

- **der** The der encoding of the content of the extension.
- **derSz** The size in bytes of the der encoding.
- **pub** pointer to a populated structure of TPM2B_PUBLIC type
- **sens** pointer to a populated structure of TPM2B_SENSITIVE type
- **attributes** integer value of TPMA_OBJECT type, can contain one or more attributes, e.g. TPMA_OBJECT_fixedTPM (or 0 to automatically populate)

See:

- wolfTPM2_ImportPublicKeyBuffer
- wolfTPM2_ImportPrivateKeyBuffer
- wolfTPM2_DecodeRsaDer

Return:

- TPM_RC_SUCCESS: successful
- TPM_RC_FAILURE: generic failure (check TPM IO and TPM return code)

```
WOLFTPM_API int wolfTPM2_SignHash(
    WOLFTPM2_DEV * dev,
    WOLFTPM2_KEY * key,
    const byte * digest,
    int digestSz,
    byte * sig,
    int * sigSz
)
```

Helper function to sign arbitrary data using a TPM key.

Parameters:

- **dev** pointer to a TPM2_DEV struct
- **key** pointer to a struct of WOLFTPM2_KEY type, holding a TPM key material
- **digest** pointer to a byte buffer, containing the arbitrary data
- **digestSz** integer value, specifying the size of the digest buffer, in bytes
- **sig** pointer to a byte buffer, containing the generated signature
- **sigSz** integer value, specifying the size of the signature buffer, in bytes

See:

- wolfTPM2_VerifyHash
- wolfTPM2_SignHashScheme
- wolfTPM2_VerifyHashScheme

Return:

- TPM_RC_SUCCESS: successful
- TPM_RC_FAILURE: generic failure (check TPM IO and TPM return code)
- BAD_FUNC_ARG: check the provided arguments

```
WOLFTPM_API int wolfTPM2_SignHashScheme(
    WOLFTPM2_DEV * dev,
    WOLFTPM2_KEY * key,
    const byte * digest,
    int digestSz,
    byte * sig,
    int * sigSz,
    TPMI_ALG_SIG_SCHEME sigAlg,
    TPMI_ALG_HASH hashAlg
)
```

Advanced helper function to sign arbitrary data using a TPM key, and specify the signature scheme and hashing algorithm.

Parameters:

- **dev** pointer to a TPM2_DEV struct
- **key** pointer to a struct of WOLFTPM2_KEY type, holding a TPM key material

- **digest** pointer to a byte buffer, containing the arbitrary data
- **digestSz** integer value, specifying the size of the digest buffer, in bytes
- **sig** pointer to a byte buffer, containing the generated signature
- **sigSz** integer value, specifying the size of the signature buffer, in bytes
- **sigAlg** integer value of TPMI_ALG_SIG_SCHEME type, specifying a supported TPM 2.0 signature scheme
- **hashAlg** integer value of TPMI_ALG_HASH type, specifying a supported TPM 2.0 hash algorithm

See:

- wolfTPM2_SignHash
- wolfTPM2_VerifyHash
- wolfTPM2_VerifyHashScheme

Return:

- TPM_RC_SUCCESS: successful
- TPM_RC_FAILURE: generic failure (check TPM IO and TPM return code)
- BAD_FUNC_ARG: check the provided arguments

Note: ECC signatures contain fixed-width R followed by S, each padded to the signing key curve size.

```
WOLFTPM_API int wolfTPM2_VerifyHash(
    WOLFTPM2_DEV * dev,
    WOLFTPM2_KEY * key,
    const byte * sig,
    int sigSz,
    const byte * digest,
    int digestSz
)
```

Helper function to verify a TPM generated signature.

Parameters:

- **dev** pointer to a TPM2_DEV struct
- **key** pointer to a struct of WOLFTPM2_KEY type, holding a TPM 2.0 key material
- **sig** pointer to a byte buffer, containing the generated signature
- **sigSz** integer value, specifying the size of the signature buffer, in bytes
- **digest** pointer to a byte buffer, containing the signed data
- **digestSz** integer value, specifying the size of the digest buffer, in bytes

See:

- wolfTPM2_SignHash
- wolfTPM2_SignHashScheme
- wolfTPM2_VerifyHashScheme
- wolfTPM2_VerifyHash_ex

Return:

- TPM_RC_SUCCESS: successful
- TPM_RC_FAILURE: generic failure (check TPM IO and TPM return code)
- BAD_FUNC_ARG: check the provided arguments

```
WOLFTPM_API int wolfTPM2_VerifyHash_ex(
    WOLFTPM2_DEV * dev,
    WOLFTPM2_KEY * key,
    const byte * sig,
    int sigSz,
    const byte * digest,
    int digestSz,
    int hashAlg
)
```

Helper function to verify a TPM generated signature.

Parameters:

- **dev** pointer to a TPM2_DEV struct
- **key** pointer to a struct of WOLFTPM2_KEY type, holding a TPM 2.0 key material
- **sig** pointer to a byte buffer, containing the generated signature
- **sigSz** integer value, specifying the size of the signature buffer, in bytes
- **digest** pointer to a byte buffer, containing the signed data
- **digestSz** integer value, specifying the size of the digest buffer, in bytes
- **hashAlg** hash algorithm used to sign

See:

- wolfTPM2_SignHash
- wolfTPM2_SignHashScheme
- wolfTPM2_VerifyHashScheme

Return:

- TPM_RC_SUCCESS: successful
- TPM_RC_FAILURE: generic failure (check TPM IO and TPM return code)
- BAD_FUNC_ARG: check the provided arguments

```
WOLFTPM_API int wolfTPM2_VerifyHashScheme(
    WOLFTPM2_DEV * dev,
    WOLFTPM2_KEY * key,
    const byte * sig,
    int sigSz,
    const byte * digest,
    int digestSz,
    TPMI_ALG_SIG_SCHEME sigAlg,
    TPMI_ALG_HASH hashAlg
)
```

Advanced helper function to verify a TPM generated signature.

Parameters:

- **dev** pointer to a TPM2_DEV struct
- **key** pointer to a struct of WOLFTPM2_KEY type, holding a TPM 2.0 key material
- **sig** pointer to a byte buffer, containing the generated signature
- **sigSz** integer value, specifying the size of the signature buffer, in bytes
- **digest** pointer to a byte buffer, containing the signed data
- **digestSz** integer value, specifying the size of the digest buffer, in bytes
- **sigAlg** integer value of TPMI_ALG_SIG_SCHEME type, specifying a supported TPM 2.0 signature scheme
- **hashAlg** integer value of TPMI_ALG_HASH type, specifying a supported TPM 2.0 hash algorithm

See:

- wolfTPM2_SignHash
- wolfTPM2_SignHashScheme
- wolfTPM2_VerifyHash

Return:

- TPM_RC_SUCCESS: successful
- TPM_RC_FAILURE: generic failure (check TPM IO and TPM return code)
- BAD_FUNC_ARG: check the provided arguments

```
WOLFTPM_API int wolfTPM2_VerifyHashTicket(
    WOLFTPM2_DEV * dev,
    WOLFTPM2_KEY * key,
    const byte * sig,
    int sigSz,
    const byte * digest,
    int digestSz,
    TPMI_ALG_SIG_SCHEME sigAlg,
    TPMI_ALG_HASH hashAlg,
    TPMT_TK_VERIFIED * checkTicket
)
```

Advanced helper function to verify a TPM generated signature and return ticket.

Parameters:

- **dev** pointer to a TPM2_DEV struct
- **key** pointer to a struct of WOLFTPM2_KEY type, holding a TPM 2.0 key material
- **sig** pointer to a byte buffer, containing the generated signature
- **sigSz** integer value, specifying the size of the signature buffer, in bytes
- **digest** pointer to a byte buffer, containing the signed data
- **digestSz** integer value, specifying the size of the digest buffer, in bytes
- **sigAlg** integer value of TPMI_ALG_SIG_SCHEME type, specifying a supported TPM 2.0 signature scheme

- **hashAlg** integer value of TPMI_ALG_HASH type, specifying a supported TPM 2.0 hash algorithm
- **checkTicket** returns the validation ticket proving the signature for digest was checked

See:

- wolfTPM2_VerifyHashScheme
- wolfTPM2_VerifyHashTicket
- wolfTPM2_VerifyHash

Return:

- TPM_RC_SUCCESS: successful
- TPM_RC_FAILURE: generic failure (check TPM IO and TPM return code)
- BAD_FUNC_ARG: check the provided arguments

```
WOLFTPM_API int wolfTPM2_SignSequenceStart(
    WOLFTPM2_DEV * dev,
    WOLFTPM2_KEY * key,
    const byte * context,
    int contextSz,
    TPM_HANDLE * sequenceHandle
)
```

Post-Quantum Cryptography: Start a signing sequence with context support.

Parameters:

- **dev** Device structure
- **key** Signing key
- **context** Signature context (for PQ algorithms like ML-DSA)
- **contextSz** Size of context buffer
- **sequenceHandle** Output sequence handle

See:

- wolfTPM2_SignSequenceUpdate
- wolfTPM2_SignSequenceComplete
- wolfTPM2_VerifySequenceStart

Return: 0 on success, negative on error

Example

```
WOLFTPM2_DEV dev;
WOLFTPM2_KEY key;
TPM_HANDLE sequenceHandle;
byte context[1024];
int contextSz = sizeof(context);
```

```
// Initialize and set up key...
rc = wolfTPM2_SignSequenceStart(&dev, &key, context, contextSz,
 ↪  &sequenceHandle);
```

```
WOLFTPM_API int wolfTPM2_SignSequenceUpdate(
    WOLFTPM2_DEV * dev,
    TPM_HANDLE sequenceHandle,
    const byte * data,
    int dataSz
)
```

Post-Quantum Cryptography: Update a signing sequence with data.

Parameters:

- **dev** Device structure
- **sequenceHandle** Sequence handle from SignSequenceStart
- **data** Data to add to sequence
- **dataSz** Size of data buffer

See:

- wolfTPM2_SignSequenceStart
- wolfTPM2_SignSequenceComplete

Return: 0 on success, negative on error

Example

```
byte data[256];
rc = wolfTPM2_SignSequenceUpdate(&dev, sequenceHandle, data, sizeof(data));
```

```
WOLFTPM_API int wolfTPM2_SignSequenceComplete(
    WOLFTPM2_DEV * dev,
    TPM_HANDLE sequenceHandle,
    WOLFTPM2_KEY * key,
    const byte * data,
    int dataSz,
    byte * sig,
    int * sigSz
)
```

Post-Quantum Cryptography: Complete a signing sequence.

Parameters:

- **dev** Device structure
- **sequenceHandle** Sequence handle from SignSequenceStart
- **key** Signing key

- **data** Final data to add to sequence
- **dataSz** Size of data buffer
- **sig** Output signature buffer
- **sigSz** Input/output signature size

See:

- wolfTPM2_SignSequenceStart
- wolfTPM2_SignSequenceUpdate

Return: 0 on success, negative on error

Note: ECC signatures contain fixed-width R followed by S, each padded to the signing key curve size.

Example

```
byte sig[2048];
int sigSz = sizeof(sig);
rc = wolfTPM2_SignSequenceComplete(&dev, sequenceHandle, &key, data, dataSz,
 ↪  sig, &sigSz);
```

```
WOLFTPM_API int wolfTPM2_VerifySequenceStart(
    WOLFTPM2_DEV * dev,
    WOLFTPM2_KEY * key,
    const byte * context,
    int contextSz,
    TPM_HANDLE * sequenceHandle
)
```

Post-Quantum Cryptography: Start a verification sequence with context support.

Parameters:

- **dev** Device structure
- **key** Verification key
- **context** Signature context (for PQ algorithms like ML-DSA)
- **contextSz** Size of context buffer
- **sequenceHandle** Output sequence handle

See:

- wolfTPM2_VerifySequenceUpdate
- wolfTPM2_VerifySequenceComplete
- wolfTPM2_SignSequenceStart

Return: 0 on success, negative on error

Example

```
TPM_HANDLE sequenceHandle;
byte context[1024];
rc = wolfTPM2_VerifySequenceStart(&dev, &key, context, sizeof(context),
 ↪  &sequenceHandle);
```

```
WOLFTPM_API int wolfTPM2_VerifySequenceUpdate(
    WOLFTPM2_DEV * dev,
    TPM_HANDLE sequenceHandle,
    const byte * data,
    int dataSz
)
```

Post-Quantum Cryptography: Update a verification sequence with data.

Parameters:

- **dev** Device structure
- **sequenceHandle** Sequence handle from VerifySequenceStart
- **data** Data to add to sequence
- **dataSz** Size of data buffer

See:

- wolfTPM2_VerifySequenceStart
- wolfTPM2_VerifySequenceComplete

Return: 0 on success, negative on error

Example

```
rc = wolfTPM2_VerifySequenceUpdate(&dev, sequenceHandle, data, sizeof(data));
```

```
WOLFTPM_API int wolfTPM2_VerifySequenceComplete(
    WOLFTPM2_DEV * dev,
    TPM_HANDLE sequenceHandle,
    WOLFTPM2_KEY * key,
    const byte * data,
    int dataSz,
    const byte * sig,
    int sigSz,
    TPMT_TK_VERIFIED * validation
)
```

Post-Quantum Cryptography: Complete a verification sequence.

Parameters:

- **dev** Device structure
- **sequenceHandle** Sequence handle from VerifySequenceStart

- **key** Verification key
- **data** Optional final chunk of message data; if non-NULL it is folded into the sequence via an internal TPM2_SequenceUpdate before the Complete is sent (Part 3 Sec.20.3 — the wire command itself only carries the signature; the message must already be accumulated in the sequence object on the TPM).
- **dataSz** Size of data buffer; pass 0 to skip the internal update.
- **sig** Signature to verify
- **sigSz** Size of signature buffer
- **validation** Optional output validation ticket

See:

- wolfTPM2_VerifySequenceStart
- wolfTPM2_VerifySequenceUpdate

Return: 0 on success, negative on error

Example

```
TPMT_TK_VERIFIED validation;
rc = wolfTPM2_VerifySequenceComplete(&dev, sequenceHandle, &key, data, dataSz,
 ↪  sig, sigSz, &validation);
```

```
WOLFTPM_API int wolfTPM2_SignDigest(
    WOLFTPM2_DEV * dev,
    WOLFTPM2_KEY * key,
    const byte * digest,
    int digestSz,
    const byte * context,
    int contextSz,
    byte * sig,
    int * sigSz
)
```

Post-Quantum Cryptography: Sign a digest with context support.

Parameters:

- **dev** Device structure
- **key** Signing key
- **digest** Digest to sign
- **digestSz** Size of digest buffer
- **context** Signature context (for PQ algorithms like ML-DSA)
- **contextSz** Size of context buffer
- **sig** Output signature buffer
- **sigSz** Input/output signature size

See:

- wolfTPM2_VerifyDigestSignature
- wolfTPM2_SignHash

Return: 0 on success, negative on error

Note: ECC signatures contain fixed-width R followed by S, each padded to the signing key curve size.

Example

```
byte digest[64], sig[2048];
int sigSz = sizeof(sig);
byte context[1024];
rc = wolfTPM2_SignDigest(&dev, &key, digest, sizeof(digest), context,
 ↪  sizeof(context), sig, &sigSz);
```

```
WOLFTPM_API int wolfTPM2_VerifyDigestSignature(
    WOLFTPM2_DEV * dev,
    WOLFTPM2_KEY * key,
    const byte * digest,
    int digestSz,
    const byte * sig,
    int sigSz,
    const byte * context,
    int contextSz,
    TPMT_TK_VERIFIED * validation
)
```

Post-Quantum Cryptography: Verify a digest signature with context support.

Parameters:

- **dev** Device structure
- **key** Verification key
- **digest** Digest that was signed
- **digestSz** Size of digest buffer
- **sig** Signature to verify
- **sigSz** Size of signature buffer
- **context** Signature context (for PQ algorithms like ML-DSA)
- **contextSz** Size of context buffer
- **validation** Optional output validation ticket

See:

- wolfTPM2_SignDigest
- wolfTPM2_VerifyHash

Return: 0 on success, negative on error

Example

```
TPMT_TK_VERIFIED validation;
rc = wolfTPM2_VerifyDigestSignature(&dev, &key, digest, sizeof(digest), sig,
 ↪  sigSz, context, sizeof(context), &validation);
```

```
WOLFTPM_API int wolfTPM2_Encapsulate(
    WOLFTPM2_DEV * dev,
    WOLFTPM2_KEY * key,
    byte * ciphertext,
    int * ciphertextSz,
    byte * sharedSecret,
    int * sharedSecretSz
)
```

Post-Quantum Cryptography: KEM Encapsulate (public key operation)

Parameters:

- **dev** Device structure
- **key** Public key for encapsulation
- **ciphertext** Output ciphertext buffer
- **ciphertextSz** Input/output ciphertext size
- **sharedSecret** Output shared secret buffer
- **sharedSecretSz** Input/output shared secret size

See: wolfTPM2_Decapsulate

Return: 0 on success, negative on error

Example

```
byte ciphertext[2048], sharedSecret[64];
int ciphertextSz = sizeof(ciphertext);
int sharedSecretSz = sizeof(sharedSecret);
rc = wolfTPM2_Encapsulate(&dev, &key, ciphertext, &ciphertextSz, sharedSecret,
 ↪  &sharedSecretSz);
```

```
WOLFTPM_API int wolfTPM2_Decapsulate(
    WOLFTPM2_DEV * dev,
    WOLFTPM2_KEY * key,
    const byte * ciphertext,
    int ciphertextSz,
    byte * sharedSecret,
    int * sharedSecretSz
)
```

Post-Quantum Cryptography: KEM Decapsulate (private key operation)

Parameters:

- **dev** Device structure
- **key** Private key for decapsulation
- **ciphertext** Ciphertext to decapsulate
- **ciphertextSz** Size of ciphertext buffer
- **sharedSecret** Output shared secret buffer
- **sharedSecretSz** Input/output shared secret size

See: wolfTPM2_Encapsulate

Return: 0 on success, negative on error

Example

```
byte sharedSecret[64];
int sharedSecretSz = sizeof(sharedSecret);
rc = wolfTPM2_Decapsulate(&dev, &key, ciphertext, ciphertextSz, sharedSecret,
 ↪  &sharedSecretSz);
```

```
WOLFTPM_API int wolfTPM2_GetKeyTemplate_MLDSA(
    TPMT_PUBLIC * publicTemplate,
    TPMA_OBJECT objectAttributes,
    TPMI_MLDSA_PARAMETER_SET parameterSet,
    int allowExternalMu
)
```

Create a key template for ML-DSA signing keys (TCG v185)

Parameters:

- **publicTemplate** Pointer to TPMT_PUBLIC structure to populate
- **objectAttributes** Key attributes (sign flag will be enforced, decrypt cleared)
- **parameterSet** ML-DSA parameter set (TPM_MLDSA_44, TPM_MLDSA_65, or TPM_MLDSA_87)
- **allowExternalMu** Non-zero to allow TPM2_SignDigest/VerifyDigestSignature

See:

- wolfTPM2_GetKeyTemplate_HASH_MLDSA
- wolfTPM2_GetKeyTemplate_MLKEM

Return: TPM_RC_SUCCESS on success, BAD_FUNC_ARG if publicTemplate is NULL

```
WOLFTPM_API int wolfTPM2_GetKeyTemplate_MLDSA_ex(
    TPMT_PUBLIC * publicTemplate,
    TPMA_OBJECT objectAttributes,
    TPMI_MLDSA_PARAMETER_SET parameterSet,
    int allowExternalMu,
    TPMI_ALG_HASH nameAlg
)
```

Create an ML-DSA key template selecting the name algorithm (TCG v185)

Parameters:

- **publicTemplate** Pointer to TPMT_PUBLIC structure to populate
- **objectAttributes** Key attributes (sign flag will be enforced, decrypt cleared)
- **parameterSet** ML-DSA parameter set (TPM_MLDSA_44, TPM_MLDSA_65, or TPM_MLDSA_87)
- **allowExternalMu** Non-zero to allow TPM2_SignDigest/VerifyDigestSignature

- **nameAlg** Object name algorithm (e.g. TPM_ALG_SHA512). TPM_ALG_NULL selects the wrapper default.

See: wolfTPM2_GetKeyTemplate_MLDSA

Return: TPM_RC_SUCCESS on success, BAD_FUNC_ARG if publicTemplate is NULL

```
WOLFTPM_API int wolfTPM2_GetKeyTemplate_HASH_MLDSA(
    TPMT_PUBLIC * publicTemplate,
    TPMA_OBJECT objectAttributes,
    TPMI_MLDSA_PARAMETER_SET parameterSet,
    TPMI_ALG_HASH hashAlg
)
```

Create a key template for Pre-Hash ML-DSA signing keys (TCG v185)

Parameters:

- **publicTemplate** Pointer to TPMT_PUBLIC structure to populate
- **objectAttributes** Key attributes (sign flag will be enforced, decrypt cleared)
- **parameterSet** ML-DSA parameter set (TPM_MLDSA_44, TPM_MLDSA_65, or TPM_MLDSA_87)
- **hashAlg** Pre-hash algorithm (e.g., TPM_ALG_SHA256)

See:

- wolfTPM2_GetKeyTemplate_MLDSA
- wolfTPM2_GetKeyTemplate_MLKEM

Return: TPM_RC_SUCCESS on success, BAD_FUNC_ARG if publicTemplate is NULL

```
WOLFTPM_API int wolfTPM2_GetKeyTemplate_HASH_MLDSA_ex(
    TPMT_PUBLIC * publicTemplate,
    TPMA_OBJECT objectAttributes,
    TPMI_MLDSA_PARAMETER_SET parameterSet,
    TPMI_ALG_HASH hashAlg,
    TPMI_ALG_HASH nameAlg
)
```

Create a Pre-Hash ML-DSA key template selecting the name algorithm (TCG v185)

Parameters:

- **publicTemplate** Pointer to TPMT_PUBLIC structure to populate
- **objectAttributes** Key attributes (sign flag will be enforced, decrypt cleared)
- **parameterSet** ML-DSA parameter set (TPM_MLDSA_44, TPM_MLDSA_65, or TPM_MLDSA_87)
- **hashAlg** Pre-hash algorithm (e.g., TPM_ALG_SHA256)
- **nameAlg** Object name algorithm (e.g. TPM_ALG_SHA512). TPM_ALG_NULL selects the wrapper default.

See: wolfTPM2_GetKeyTemplate_HASH_MLDSA

Return: TPM_RC_SUCCESS on success, BAD_FUNC_ARG if publicTemplate is NULL

```
WOLFTPM_API int wolfTPM2_GetKeyTemplate_MLKEM(
    TPMT_PUBLIC * publicTemplate,
    TPMA_OBJECT objectAttributes,
    TPMI_MLKEM_PARAMETER_SET parameterSet
)
```

Create a key template for ML-KEM decryption keys (TCG v185)

Parameters:

- **publicTemplate** Pointer to TPMT_PUBLIC structure to populate
- **objectAttributes** Key attributes (decrypt flag will be enforced, sign cleared)
- **parameterSet** ML-KEM parameter set (TPM_MLKEM_512, TPM_MLKEM_768, or TPM_MLKEM_1024)

See:

- wolfTPM2_GetKeyTemplate_MLDSA
- wolfTPM2_GetKeyTemplate_HASH_MLDSA

Return: TPM_RC_SUCCESS on success, BAD_FUNC_ARG if publicTemplate is NULL

```
WOLFTPM_API int wolfTPM2_GetKeyTemplate_MLKEM_ex(
    TPMT_PUBLIC * publicTemplate,
    TPMA_OBJECT objectAttributes,
    TPMI_MLKEM_PARAMETER_SET parameterSet,
    TPMI_ALG_HASH nameAlg
)
```

Create an ML-KEM key template selecting the name algorithm (TCG v185)

Parameters:

- **publicTemplate** Pointer to TPMT_PUBLIC structure to populate
- **objectAttributes** Key attributes (decrypt flag will be enforced, sign cleared)
- **parameterSet** ML-KEM parameter set (TPM_MLKEM_512, TPM_MLKEM_768, or TPM_MLKEM_1024)
- **nameAlg** Object name algorithm (e.g. TPM_ALG_SHA512). TPM_ALG_NULL selects the wrapper default.

See: wolfTPM2_GetKeyTemplate_MLKEM

Return: TPM_RC_SUCCESS on success, BAD_FUNC_ARG if publicTemplate is NULL

```
WOLFTPM_API int wolfTPM2_ECDHGenKey(
    WOLFTPM2_DEV * dev,
    WOLFTPM2_KEY * ecdhKey,
    int curve_id,
    const byte * auth,
    int authSz
)
```

Generates and then loads a ECC key-pair with NULL hierarchy for Diffie-Hellman exchange.

Parameters:

- **dev** pointer to a TPM2_DEV struct
- **ecdhKey** pointer to an empty structure of WOLFTPM2_KEY type
- **curve_id** integer value, specifying a valid TPM_ECC_CURVE value
- **auth** pointer to a string constant, specifying the password authorization for the TPM 2.0 Key
- **authSz** integer value, specifying the size of the password authorization, in bytes

See:

- wolfTPM2_ECDHGen
- wolfTPM2_ECDHGenZ
- wolfTPM2_ECDHEGenKey
- wolfTPM2_ECDHEGenZ

Return:

- TPM_RC_SUCCESS: successful
- TPM_RC_FAILURE: generic failure (check TPM IO and TPM return code)
- BAD_FUNC_ARG: check the provided arguments

```
WOLFTPM_API int wolfTPM2_ECDHGen(
    WOLFTPM2_DEV * dev,
    WOLFTPM2_KEY * privKey,
    TPM2B_ECC_POINT * pubPoint,
    byte * out,
    int * outSz
)
```

Generates ephemeral key and computes Z (shared secret)

Parameters:

- **dev** pointer to a TPM2_DEV struct
- **privKey** pointer to a structure of WOLFTPM2_KEY type
- **pubPoint** pointer to an empty structure of TPM2B_ECC_POINT type
- **out** pointer to a byte buffer, to store the generated shared secret
- **outSz** integer value, specifying the size of the shared secret, in bytes

See:

- wolfTPM2_ECDHGenZ
- wolfTPM2_ECDHGenKey
- wolfTPM2_ECDHEGenKey
- wolfTPM2_ECDHEGenZ

Return:

- TPM_RC_SUCCESS: successful
- TPM_RC_FAILURE: generic failure (check TPM IO and TPM return code)
- BAD_FUNC_ARG: check the provided arguments

Note: One shot API using private key handle to generate key-pair and return public point and shared secret

```
WOLFTPM_API int wolfTPM2_ECDHGenZ(
    WOLFTPM2_DEV * dev,
    WOLFTPM2_KEY * privKey,
    const TPM2B_ECC_POINT * pubPoint,
    byte * out,
    int * outSz
)
```

Computes Z (shared secret) using pubPoint and loaded private ECC key.

Parameters:

- **dev** pointer to a TPM2_DEV struct
- **privKey** pointer to a structure of WOLFTPM2_KEY type, containing a valid TPM handle
- **pubPoint** pointer to a populated structure of TPM2B_ECC_POINT type
- **out** pointer to a byte buffer, to store the computed shared secret
- **outSz** integer value, specifying the size of the shared secret, in bytes

See:

- wolfTPM2_ECDHGen
- wolfTPM2_ECDHGenKey
- wolfTPM2_ECDHEGenKey
- wolfTPM2_ECDHEGenZ

Return:

- TPM_RC_SUCCESS: successful
- TPM_RC_FAILURE: generic failure (check TPM IO and TPM return code)
- BAD_FUNC_ARG: check the provided arguments

```
WOLFTPM_API int wolfTPM2_ECDHEGenKey(
    WOLFTPM2_DEV * dev,
    WOLFTPM2_KEY * ecdhKey,
    int curve_id
```

)

Generates ephemeral ECC key and returns array index (2 phase method)

Parameters:

- **dev** pointer to a TPM2_DEV struct
- **ecdhKey** pointer to an empty structure of WOLFTPM2_KEY type
- **curve_id** integer value, specifying a valid TPM_ECC_CURVE value

See:

- wolfTPM2_ECDHEGenZ
- wolfTPM2_ECDHGen
- wolfTPM2_ECDHGenKey
- wolfTPM2_ECDHGenZ

Return:

- TPM_RC_SUCCESS: successful
- TPM_RC_FAILURE: generic failure (check TPM IO and TPM return code)
- BAD_FUNC_ARG: check the provided arguments

Note: One time use key

```
WOLFTPM_API int wolfTPM2_ECDHEGenZ(
    WOLFTPM2_DEV * dev,
    WOLFTPM2_KEY * parentKey,
    WOLFTPM2_KEY * ecdhKey,
    const TPM2B_ECC_POINT * pubPoint,
    byte * out,
    int * outSz
)
```

Computes Z (shared secret) using pubPoint and counter (2 phase method)

Parameters:

- **dev** pointer to a TPM2_DEV struct
- **parentKey** pointer to a structure of WOLFTPM2_KEY type, containing a valid TPM handle of a primary key
- **ecdhKey** pointer to a structure of WOLFTPM2_KEY type, containing a valid TPM handle
- **pubPoint** pointer to an empty struct of TPM2B_ECC_POINT type
- **out** pointer to a byte buffer, to store the computed shared secret
- **outSz** integer value, specifying the size of the shared secret, in bytes

See:

- wolfTPM2_ECDHEGenKey
- wolfTPM2_ECDHGen

- wolfTPM2_ECDHGenKey
- wolfTPM2_ECDHGenZ

Return:

- TPM_RC_SUCCESS: successful
- TPM_RC_FAILURE: generic failure (check TPM IO and TPM return code)
- BAD_FUNC_ARG: check the provided arguments

Note: The counter, array ID, can only be used one time

```
WOLFTPM_API int wolfTPM2_RsaEncrypt(
    WOLFTPM2_DEV * dev,
    WOLFTPM2_KEY * key,
    TPM_ALG_ID padScheme,
    const byte * msg,
    int msgSz,
    byte * out,
    int * outSz
)
```

Perform RSA encryption using a TPM 2.0 key.

Parameters:

- **dev** pointer to a TPM2_DEV struct
- **key** pointer to a struct of WOLFTPM2_KEY type, holding a TPM key material
- **padScheme** integer value of TPM_ALG_ID type, specifying the padding scheme
- **msg** pointer to a byte buffer, containing the arbitrary data for encryption
- **msgSz** integer value, specifying the size of the arbitrary data buffer
- **out** pointer to a byte buffer, where the encrypted data will be stored
- **outSz** integer value, specifying the size of the encrypted data buffer

See: wolfTPM2_RsaDecrypt

Return:

- TPM_RC_SUCCESS: successful
- TPM_RC_FAILURE: generic failure (check TPM IO and TPM return code)
- BAD_FUNC_ARG: check the provided arguments

```
WOLFTPM_API int wolfTPM2_RsaEncrypt_ex(
    WOLFTPM2_DEV * dev,
    WOLFTPM2_KEY * key,
    TPM_ALG_ID padScheme,
    const byte * msg,
    int msgSz,
    byte * out,
    int * outSz,
    TPMI_ALG_HASH hashAlg
```

)

Perform RSA encryption selecting the padding scheme hash algorithm.

Parameters:

- **dev** pointer to a TPM2_DEV struct
- **key** pointer to a struct of WOLFTPM2_KEY type, holding a TPM key material
- **padScheme** integer value of TPM_ALG_ID type, specifying the padding scheme
- **msg** pointer to a byte buffer, containing the arbitrary data for encryption
- **msgSz** integer value, specifying the size of the arbitrary data buffer
- **out** pointer to a byte buffer, where the encrypted data will be stored
- **outSz** integer value, specifying the size of the encrypted data buffer
- **hashAlg** hash for the padding scheme (e.g. TPM_ALG_SHA512 for OAEP). TPM_ALG_NULL selects the wrapper default.

See:

- wolfTPM2_RsaEncrypt
- wolfTPM2_RsaDecrypt_ex

Return:

- TPM_RC_SUCCESS: successful
- TPM_RC_FAILURE: generic failure (check TPM IO and TPM return code)
- BAD_FUNC_ARG: check the provided arguments

```
WOLFTPM_API int wolfTPM2_RsaDecrypt(
    WOLFTPM2_DEV * dev,
    WOLFTPM2_KEY * key,
    TPM_ALG_ID padScheme,
    const byte * in,
    int inSz,
    byte * msg,
    int * msgSz
)
```

Perform RSA decryption using a TPM 2.0 key.

Parameters:

- **dev** pointer to a TPM2_DEV struct
- **key** pointer to a struct of WOLFTPM2_KEY type, holding a TPM key material
- **padScheme** integer value of TPM_ALG_ID type, specifying the padding scheme
- **in** pointer to a byte buffer, containing the encrypted data
- **inSz** integer value, specifying the size of the encrypted data buffer
- **msg** pointer to a byte buffer, containing the decrypted data
- **msgSz** pointer to size of the encrypted data buffer, on return set actual size

See: wolfTPM2_RsaEncrypt

Return:

- TPM_RC_SUCCESS: successful
- TPM_RC_FAILURE: generic failure (check TPM IO and TPM return code)
- BAD_FUNC_ARG: check the provided arguments

```
WOLFTPM_API int wolfTPM2_RsaDecrypt_ex(
    WOLFTPM2_DEV * dev,
    WOLFTPM2_KEY * key,
    TPM_ALG_ID padScheme,
    const byte * in,
    int inSz,
    byte * msg,
    int * msgSz,
    TPMI_ALG_HASH hashAlg
)
```

Perform RSA decryption selecting the padding scheme hash algorithm.

Parameters:

- **dev** pointer to a TPM2_DEV struct
- **key** pointer to a struct of WOLFTPM2_KEY type, holding a TPM key material
- **padScheme** integer value of TPM_ALG_ID type, specifying the padding scheme
- **in** pointer to a byte buffer, containing the encrypted data
- **inSz** integer value, specifying the size of the encrypted data buffer
- **msg** pointer to a byte buffer, containing the decrypted data
- **msgSz** pointer to size of the encrypted data buffer, on return set actual size
- **hashAlg** hash for the padding scheme (e.g. TPM_ALG_SHA512 for OAEP). TPM_ALG_NULL selects the wrapper default.

See:

- wolfTPM2_RsaDecrypt
- wolfTPM2_RsaEncrypt_ex

Return:

- TPM_RC_SUCCESS: successful
- TPM_RC_FAILURE: generic failure (check TPM IO and TPM return code)
- BAD_FUNC_ARG: check the provided arguments

```
WOLFTPM_API int wolfTPM2_ReadPCR(
    WOLFTPM2_DEV * dev,
    int pcrIndex,
    int hashAlg,
```

```
    byte * digest,
    int * pDigestLen
)
```

Read the values of a specified TPM 2.0 Platform Configuration Registers(PCR)

Parameters:

- **dev** pointer to a TPM2_DEV struct
- **pcrIndex** integer value, specifying a valid PCR index, between 0 and 23 (TPM locality could have an impact on successful access)
- **hashAlg** integer value, specifying a TPM_ALG_SHA256 or TPM_ALG_SHA1 registers to be accessed
- **digest** pointer to a byte buffer, where the PCR values will be stored
- **pDigestLen** pointer to an integer variable, where the size of the digest buffer will be stored

See: wolfTPM2_ExtendPCR

Return:

- TPM_RC_SUCCESS: successful
- TPM_RC_FAILURE: generic failure (check TPM IO and TPM return code)
- BAD_FUNC_ARG: check the provided arguments

Note: Make sure to specify the correct hashing algorithm, because there are two sets of PCR registers, one for SHA256 and the other for SHA1(deprecated, but still possible to be read)

```
WOLFTPM_API int wolfTPM2_ResetPCR(
    WOLFTPM2_DEV * dev,
    int pcrIndex
)
```

Reset a PCR register to its default value.

Parameters:

- **dev** pointer to a TPM2_DEV struct
- **pcrIndex** integer value from PCR_FIRST through PCR_LAST

See:

- wolfTPM2_ReadPCR
- wolfTPM2_ExtendPCR

Return:

- TPM_RC_SUCCESS: successful
- TPM_RC_FAILURE: generic failure (check TPM IO and TPM return code)
- BAD_FUNC_ARG: check the provided arguments

Note: Reset permission depends on the selected PCR's TPM attributes and locality.

```
WOLFTPM_API int wolfTPM2_SetLocality(
    WOLFTPM2_DEV * dev,
    int locality
)
```

Change the active TPM locality at runtime.

Parameters:

- **dev** pointer to a TPM2_DEV struct
- **locality** integer value, specifying the TPM locality (0 to 4)

See: wolfTPM2_ResetPCR

Return:

- TPM_RC_SUCCESS: locality granted and now active
- other: on the built-in TIS driver any non-success TPM_RC_* code (for example TPM_RC_TIMEOUT) may be returned when the locality could not be granted, so test rc != TPM_RC_SUCCESS rather than a specific code
- NOT_COMPILED_IN: locality switching is not supported for this backend
- BAD_FUNC_ARG: check the provided arguments

Note: Startup uses WOLFTPM_LOCALITY_DEFAULT (0); this switches so subsequent commands run at the given locality (e.g. loc 2/3 to reset PCRs not resettable at loc 0). For the built-in TIS driver it performs the ACCESS handshake (request, then release-and-request if the TPM does not preempt, restoring the old locality on failure); for the SWTPM/mssim transport it records the locality (sent with each command). The Linux kernel driver and Windows TBS own the locality and return NOT_COMPILED_IN. Return to locality 0 when done.

```
WOLFTPM_API int wolfTPM2_ExtendPCR(
    WOLFTPM2_DEV * dev,
    int pcrIndex,
    int hashAlg,
    const byte * digest,
    int digestLen
)
```

Extend a PCR register with a user provided digest.

Parameters:

- **dev** pointer to a TPM2_DEV struct
- **pcrIndex** integer value, specifying a valid PCR index, between 0 and 23 (TPM locality could have an impact on successful access)
- **hashAlg** integer value, specifying a TPM_ALG_SHA256 or TPM_ALG_SHA1 registers to be accessed
- **digest** pointer to a byte buffer, containing the digest value to be extended into the PCR

- **digestLen** the size of the digest buffer

See: wolfTPM2_ReadPCR

Return:

- TPM_RC_SUCCESS: successful
- TPM_RC_FAILURE: generic failure (check TPM IO and TPM return code)
- BAD_FUNC_ARG: check the provided arguments

Note: Make sure to specify the correct hashing algorithm

```
WOLFTPM_API int wolfTPM2_AllocatePCRBanks(
    WOLFTPM2_DEV * dev,
    const TPM_ALG_ID * hashAlgs,
    int hashAlgCount,
    PCR_Allocate_Out * allocOut
)
```

Re-provision which PCR banks the TPM has allocated.

Parameters:

- **dev** pointer to a TPM2_DEV struct
- **hashAlgs** array of TPM_ALG_ID hash algorithms naming the complete new bank set
- **hashAlgCount** number of entries in hashAlgs, 1 to HASH_COUNT; zero is rejected because it would leave the TPM with no PCR banks
- **allocOut** optional, receives the TPM's allocationSuccess, maxPCR, sizeNeeded and sizeAvailable; zeroed on entry

See:

- wolfTPM2_AllocatePCRBanks_ex
- TPM2_IsPcrBankAllocated
- wolfTPM2_ExtendPCR

Return:

- TPM_RC_SUCCESS: the new allocation is staged for the next reset
- BUFFER_E: the TPM has no room for the requested bank set; see allocOut->sizeNeeded and allocOut->sizeAvailable
- TPM_RC_HASH: the TPM does not implement one of the requested algs; nothing was sent and the allocation is unchanged
- TPM_RC_PCR: the TPM refused the selection, typically because it keeps a single bank active and more than one was requested
- TPM_RC_HIERARCHY: the platform hierarchy is disabled
- TPM_RC_BAD_AUTH: platform auth is not the empty password; use wolfTPM2_AllocatePCRBanks_ex with a session
- BAD_FUNC_ARG: check the provided arguments

Note:

- This REPLACES the allocation: hashAlgs is the complete new set and every bank not named in it is deallocated. Allocating SHA-384 alone on a SHA-256 TPM removes the SHA-256 bank. Many parts (Infineon SLB9672 and later) support only one active bank at a time and reject a multi-bank request outright with TPM_RC_PCR. A TPM that answers but lacks the space reports it in allocOut instead, which maps to BUFFER_E.
- Changing banks invalidates every PolicyPCR digest and makes every blob sealed to PCR values unsealable. The PCR contents are zeroed at the Startup(CLEAR) and re-allocating the old bank does not bring them back.
- Success means the change is STAGED. The TPM applies it at the next Startup(CLEAR) following a _TPM_Init, so the caller must power cycle the TPM (or restart the simulator process) and re-read the banks with TPM2_IsPcrBankAllocated to confirm. No TPM command performs that reset.
- Requires the platform hierarchy. Under an OS the platform firmware has usually cleared phEnable, in which case the TPM answers TPM_RC_HIERARCHY no matter what auth is supplied.

```
WOLFTPM_API int wolfTPM2_AllocatePCRBanks_ex(
    WOLFTPM2_DEV * dev,
    WOLFTPM2_SESSION * session,
    const TPM_ALG_ID * hashAlgs,
    int hashAlgCount,
    PCR_Allocate_Out * allocOut
)
```

Re-provision the TPM's PCR banks using a caller supplied session.

Parameters:

- **dev** pointer to a TPM2_DEV struct
- **session** pointer to a WOLFTPM2_SESSION for platform auth, or NULL
- **hashAlgs** array of TPM_ALG_ID hash algorithms naming the complete new bank set
- **hashAlgCount** number of entries in hashAlgs, 1 to HASH_COUNT
- **allocOut** optional, receives the TPM's allocation result

See: wolfTPM2_AllocatePCRBanks

Return:

- TPM_RC_SUCCESS: the new allocation is staged for the next reset
- BAD_FUNC_ARG: check the provided arguments

Note:

- Same as wolfTPM2_AllocatePCRBanks, but takes a session for a platform hierarchy that carries an authPolicy. This must be a POLICY session: slot 0 authorization is a password or a policy session, so an HMAC session there is not an authorization session and the TPM answers TPM_RC_AUTH_MISSING. Passing NULL is identical to wolfTPM2_AllocatePCRBanks.
- Session slot 0 is saved and restored around the command, and the response nonce is copied back into the caller's session so it stays usable for a following command.

```
WOLFTPM_API int wolfTPM2_NVCreateAuth(
    WOLFTPM2_DEV * dev,
    WOLFTPM2_HANDLE * parent,
    WOLFTPM2_NV * nv,
    word32 nvIndex,
    word32 nvAttributes,
    word32 maxSize,
    const byte * auth,
    int authSz
)
```

Creates a new NV Index to be later used for storing data into the TPM's NVRAM.

Parameters:

- **dev** pointer to a TPM2_DEV struct
- **parent** pointer to a WOLFTPM2_HANDLE, specifying the TPM hierarchy for the new NV Index
- **nv** pointer to an empty structure of WOLFTPM2_NV type, to hold the new NV Index
- **nvIndex** integer value, holding the NV Index Handle given by the TPM upon success
- **nvAttributes** integer value, use wolfTPM2_GetNvAttributesTemplate to create correct value
- **maxSize** integer value, specifying the maximum number of bytes written at this NV Index
- **auth** pointer to a string constant, specifying the password authorization for this NV Index
- **authSz** integer value, specifying the size of the password authorization, in bytes

See:

- wolfTPM2_NVCreateAuthPolicy
- wolfTPM2_NVWriteAuth
- wolfTPM2_NVReadAuth
- wolfTPM2_NVDeleteAuth
- wolfTPM2_NVOpen

Return:

- TPM_RC_SUCCESS: successful
- TPM_RC_FAILURE: generic failure (check TPM IO and TPM return code)
- BAD_FUNC_ARG: check the provided arguments

Note: This is a wolfTPM2 wrapper around TPM2_NV_DefineSpace

```
WOLFTPM_API int wolfTPM2_NVCreateAuthPolicy(
    WOLFTPM2_DEV * dev,
    WOLFTPM2_HANDLE * parent,
    WOLFTPM2_NV * nv,
    word32 nvIndex,
    word32 nvAttributes,
    word32 maxSize,
    const byte * auth,
    int authSz,
    const byte * authPolicy,
    int authPolicySz
```

)

Creates a new NV Index to be later used for storing data into the TPM's NVRAM.

Parameters:

- **dev** pointer to a TPM2_DEV struct
- **parent** pointer to a WOLFTPM2_HANDLE, specifying the TPM hierarchy for the new NV Index
- **nv** pointer to an empty structure of WOLFTPM2_NV type, to hold the new NV Index
- **nvIndex** integer value, holding the NV Index Handle given by the TPM upon success
- **nvAttributes** integer value, use wolfTPM2_GetNvAttributesTemplate to create correct value
- **maxSize** integer value, specifying the maximum number of bytes written at this NV Index
- **auth** pointer to a string constant, specifying the password authorization for this NV Index
- **authSz** integer value, specifying the size of the password authorization, in bytes
- **authPolicy** optional policy for using this key (The policy is computed using the nameAlg of the object)
- **authPolicySz** size of the authPolicy

See:

- wolfTPM2_NVCreateAuth
- wolfTPM2_NVWriteAuth
- wolfTPM2_NVReadAuth
- wolfTPM2_NVDeleteAuth
- wolfTPM2_NVOpen

Return:

- TPM_RC_SUCCESS: successful
- TPM_RC_FAILURE: generic failure (check TPM IO and TPM return code)
- BAD_FUNC_ARG: check the provided arguments

Note: This is a wolfTPM2 wrapper around TPM2_NV_DefineSpace

```
WOLFTPM_API int wolfTPM2_NVCreateAuthPolicy_ex(
    WOLFTPM2_DEV * dev,
    WOLFTPM2_HANDLE * parent,
    WOLFTPM2_NV * nv,
    word32 nvIndex,
    word32 nvAttributes,
    word32 maxSize,
    const byte * auth,
    int authSz,
    const byte * authPolicy,
    int authPolicySz,
    TPMI_ALG_HASH nameAlg
)
```

Variant of wolfTPM2_NVCreateAuthPolicy that lets the caller pass an explicit nameAlg. Required when authPolicy was computed with SM3-256, SHA3-256, or any other non-SHA hash that the size-based legacy inference cannot distinguish from SHA-256.

Parameters:

- **dev** pointer to a TPM2_DEV struct
- **parent** pointer to a WOLFTPM2_HANDLE for the hierarchy
- **nv** pointer to an empty WOLFTPM2_NV that receives the created index
- **nvIndex** NV index handle (e.g. 0x01400001)
- **nvAttributes** TPMA_NV bitmask
- **maxSize** NV index data size in bytes
- **auth** optional NV authValue (NULL/0 for none)
- **authSz** optional NV authValue size
- **authPolicy** optional policy digest (NULL/0 for none)
- **authPolicySz** optional policy digest size
- **nameAlg** index name hash algorithm (e.g. TPM_ALG_SHA384). Pass TPM_ALG_NULL to fall back to size-based inference for SHA-only policies, or to use WOLFTPM2_WRAP_DIGEST for the no-policy case

Return:

- TPM_RC_SUCCESS: successful
- BAD_FUNC_ARG: nameAlg digest size disagrees with authPolicySz, or other invalid arguments
- BUFFER_E: auth or authPolicy buffer too large

```
WOLFTPM_API int wolfTPM2_NVWriteAuth(
    WOLFTPM2_DEV * dev,
    WOLFTPM2_NV * nv,
    word32 nvIndex,
    byte * dataBuf,
    word32 dataSz,
    word32 offset
)
```

Stores user data to a NV Index, at a given offset.

Parameters:

- **dev** pointer to a TPM2_DEV struct
- **nv** pointer to a populated structure of WOLFTPM2_NV type
- **nvIndex** integer value, holding an existing NV Index Handle value
- **dataBuf** pointer to a byte buffer, containing the user data to be written to the TPM's NVRAM
- **dataSz** integer value, specifying the size of the user data buffer, in bytes
- **offset** integer value of word32 type, specifying the offset from the NV Index memory start, can be zero

See:

- wolfTPM2_NVReadAuth
- wolfTPM2_NVCreateAuth
- wolfTPM2_NVDeleteAuth
- wolfTPM2_NVWriteAuthPolicy

Return:

- TPM_RC_SUCCESS: successful
- TPM_RC_FAILURE: generic failure (check TPM IO and TPM return code)
- BAD_FUNC_ARG: check the provided arguments

Note:

- User data size should be less or equal to the NV Index maxSize specified using wolfTPM2_CreateAuth
- For an index with TPMA_NV_OWNERWRITE or TPMA_NV_PPWRITE set nv->handle.hndl to TPM_RH_OWNER or TPM_RH_PLATFORM and nv->handle.auth to that hierarchy auth (after wolfTPM2_NVOpen)

```
WOLFTPM_API int wolfTPM2_NVWriteAuthPolicy(
    WOLFTPM2_DEV * dev,
    WOLFTPM2_SESSION * tpmSession,
    TPM_ALG_ID pcrAlg,
    byte * pcrArray,
    word32 pcrArraySz,
    WOLFTPM2_NV * nv,
    word32 nvIndex,
    byte * dataBuf,
    word32 dataSz,
    word32 offset
)
```

Stores user data to a NV Index, at a given offset. Allows using a policy session and PCR's for authentication.

Parameters:

- **dev** pointer to a TPM2_DEV struct
- **tpmSession** pointer to a WOLFTPM2_SESSION struct used with wolfTPM2_StartSession and wolfTPM2_SetAuthSession
- **pcrAlg** the hash algorithm to use with PCR policy
- **pcrArray** array of PCR Indexes to use when creating the policy
- **pcrArraySz** the number of PCR Indexes in the pcrArray
- **nv** pointer to a populated structure of WOLFTPM2_NV type
- **nvIndex** integer value, holding an existing NV Index Handle value
- **dataBuf** pointer to a byte buffer, containing the user data to be written to the TPM's NVRAM
- **dataSz** integer value, specifying the size of the user data buffer, in bytes
- **offset** integer value of word32 type, specifying the offset from the NV Index memory start, can be zero

See:

- wolfTPM2_NVReadAuth
- wolfTPM2_NVCreateAuth
- wolfTPM2_NVDeleteAuth
- wolfTPM2_NVWriteAuth

Return:

- TPM_RC_SUCCESS: successful
- TPM_RC_FAILURE: generic failure (check TPM IO and TPM return code)
- BAD_FUNC_ARG: check the provided arguments
- BUFFER_E: data too large for a policy without PCR's

Note:

- User data size should be less or equal to the NV Index maxSize specified using wolfTPM2_CreateAuth
- For an index with TPMA_NV_OWNERWRITE or TPMA_NV_PPWRITE set nv->handle.hndl to TPM_RH_OWNER or TPM_RH_PLATFORM and nv->handle.auth to that hierarchy auth (after wolfTPM2_NVOpen)
- pcrArray may be NULL with pcrArraySz 0 when the policy session was already satisfied by the caller (e.g. PolicySecret or PolicyCommandCode). The data must then fit in one MAX_NV_BUFFER_SIZE chunk, otherwise BUFFER_E is returned

```
WOLFTPM_API int wolfTPM2_NVExtend(
    WOLFTPM2_DEV * dev,
    WOLFTPM2_NV * nv,
    word32 nvIndex,
    byte * dataBuf,
    word32 dataSz
)
```

Extend data to an NV index.

Parameters:

- **dev** pointer to a TPM2_DEV struct
- **nv** pointer to a populated structure of WOLFTPM2_NV type
- **nvIndex** integer value, holding an existing NV Index Handle value
- **dataBuf** pointer to a byte buffer, containing the user data to be written to the TPM's NVRAM
- **dataSz** integer value, specifying the size of the user data buffer, in bytes

See:

- wolfTPM2_NVReadAuth
- wolfTPM2_NVCreateAuth
- wolfTPM2_NVOpen
- wolfTPM2_NVDeleteAuth

Return:

- TPM_RC_SUCCESS: successful
- TPM_RC_FAILURE: generic failure (check TPM IO and TPM return code)
- BAD_FUNC_ARG: check the provided arguments

Note:

- When NV index is read it will return the digest
- For an index with TPMA_NV_OWNERWRITE or TPMA_NV_PPWRITE set nv->handle.hndl to TPM_RH_OWNER or TPM_RH_PLATFORM and nv->handle.auth to that hierarchy auth (after wolfTPM2_NVOpen)

```
WOLFTPM_API int wolfTPM2_NVReadAuth(
    WOLFTPM2_DEV * dev,
    WOLFTPM2_NV * nv,
    word32 nvIndex,
    byte * dataBuf,
    word32 * pDataSz,
    word32 offset
)
```

Reads user data from a NV Index, starting at the given offset.

Parameters:

- **dev** pointer to a TPM2_DEV struct
- **nv** pointer to a populated structure of WOLFTPM2_NV type
- **nvIndex** integer value, holding an existing NV Index Handle value
- **dataBuf** pointer to an empty byte buffer, used to store the read data from the TPM's NVRAM
- **pDataSz** pointer to an integer variable, used to store the size of the data read from NVRAM, in bytes
- **offset** integer value of word32 type, specifying the offset from the NV Index memory start, can be zero

See:

- wolfTPM2_NVWriteAuth
- wolfTPM2_NVCreateAuth
- wolfTPM2_NVDeleteAuth
- wolfTPM2_NVReadAuthPolicy

Return:

- TPM_RC_SUCCESS: successful
- TPM_RC_FAILURE: generic failure (check TPM IO and TPM return code)
- BAD_FUNC_ARG: check the provided arguments

Note:

- User data size should be less or equal to the NV Index maxSize specified using wolfTPM2_CreateAuth
- For an index with TPMA_NV_OWNERREAD or TPMA_NV_PPREAD set nv->handle.hndl to TPM_RH_OWNER or TPM_RH_PLATFORM and nv->handle.auth to that hierarchy auth (after wolfTPM2_NVOpen)

```
WOLFTPM_API int wolfTPM2_NVReadAuthPolicy(
    WOLFTPM2_DEV * dev,
    WOLFTPM2_SESSION * tpmSession,
    TPM_ALG_ID pcrAlg,
    byte * pcrArray,
    word32 pcrArraySz,
    WOLFTPM2_NV * nv,
    word32 nvIndex,
    byte * dataBuf,
    word32 * pDataSz,
    word32 offset
)
```

Reads user data from a NV Index, starting at the given offset. Allows using a policy session and PCR's for authentication.

Parameters:

- **dev** pointer to a TPM2_DEV struct
- **tpmSession** pointer to a WOLFTPM2_SESSION struct used with wolfTPM2_StartSession and wolfTPM2_SetAuthSession
- **pcrAlg** the hash algorithm to use with PCR policy
- **pcrArray** array of PCR Indexes to use when creating the policy
- **pcrArraySz** the number of PCR Indexes in the pcrArray
- **nv** pointer to a populated structure of WOLFTPM2_NV type
- **nvIndex** integer value, holding an existing NV Index Handle value
- **dataBuf** pointer to an empty byte buffer, used to store the read data from the TPM's NVRAM
- **pDataSz** pointer to an integer variable, used to store the size of the data read from NVRAM, in bytes
- **offset** integer value of word32 type, specifying the offset from the NV Index memory start, can be zero

See:

- wolfTPM2_NVWriteAuth
- wolfTPM2_NVCreateAuth
- wolfTPM2_NVDeleteAuth
- wolfTPM2_NVReadAuth

Return:

- TPM_RC_SUCCESS: successful
- TPM_RC_FAILURE: generic failure (check TPM IO and TPM return code)
- BAD_FUNC_ARG: check the provided arguments
- BUFFER_E: data too large for a policy without PCR's

Note:

- User data size should be less or equal to the NV Index maxSize specified using wolfTPM2_CreateAuth
- For an index with TPMA_NV_OWNERREAD or TPMA_NV_PPREAD set nv->handle.hndl to TPM_RH_OWNER or TPM_RH_PLATFORM and nv->handle.auth to that hierarchy auth (after wolfTPM2_NVOpen)

- pcrArray may be NULL with pcrArraySz 0 when the policy session was already satisfied by the caller (e.g. PolicySecret or PolicyCommandCode). The data must then fit in one MAX_NV_BUFFER_SIZE chunk, otherwise BUFFER_E is returned

```
WOLFTPM_API int wolfTPM2_NVReadCert(
    WOLFTPM2_DEV * dev,
    TPM_HANDLE handle,
    uint8_t * buffer,
    uint32_t * len
)
```

Helper to get size of NV and read buffer without authentication. Typically used for reading a certificate from an NV.

Parameters:

- **dev** pointer to a TPM2_DEV struct
- **handle** integer value, holding an existing NV Index Handle value
- **buffer** pointer to an empty byte buffer, used to store the read data from the TPM's NVRAM
- **len** pointer to an integer variable, used to store the size of the data read from NVRAM, in bytes

See:

- wolfTPM2_NVWriteAuth
- wolfTPM2_NVCreateAuth
- wolfTPM2_NVDeleteAuth

Return:

- TPM_RC_SUCCESS: successful
- TPM_RC_FAILURE: generic failure (check TPM IO and TPM return code)
- BAD_FUNC_ARG: check the provided arguments

```
WOLFTPM_API int wolfTPM2_NVIncrement(
    WOLFTPM2_DEV * dev,
    WOLFTPM2_NV * nv
)
```

Increments an NV one-way counter.

Parameters:

- **dev** pointer to a TPM2_DEV struct
- **nv** pointer to a populated structure of WOLFTPM2_NV type

See:

- wolfTPM2_NVOpen

- wolfTPM2_NVCreateAuth

Return:

- TPM_RC_SUCCESS: successful
- TPM_RC_FAILURE: generic failure (check TPM IO and TPM return code)
- BAD_FUNC_ARG: check the provided arguments

```
WOLFTPM_API int wolfTPM2_NVOpen(
    WOLFTPM2_DEV * dev,
    WOLFTPM2_NV * nv,
    word32 nvIndex,
    const byte * auth,
    word32 authSz
)
```

Open an NV and populate the required authentication and name hash.

Parameters:

- **dev** pointer to a TPM2_DEV struct
- **nv** pointer to an empty structure of WOLFTPM2_NV type, to hold the new NV Index
- **nvIndex** integer value, holding the NV Index Handle given by the TPM upon success
- **auth** pointer to a string constant, specifying the password authorization for this NV Index
- **authSz** integer value, specifying the size of the password authorization, in bytes

See:

- wolfTPM2_NVCreateAuth
- wolfTPM2_UnloadHandle

Return:

- TPM_RC_SUCCESS: successful
- TPM_RC_FAILURE: generic failure (check TPM IO and TPM return code)
- BAD_FUNC_ARG: check the provided arguments

```
WOLFTPM_API int wolfTPM2_NVWriteLock(
    WOLFTPM2_DEV * dev,
    WOLFTPM2_NV * nv
)
```

Lock writes on the specified NV Index.

Parameters:

- **dev** pointer to a TPM2_DEV struct
- **nv** pointer to an structure of WOLFTPM2_NV type loaded using wolfTPM2_NVOpen

See:

- wolfTPM2_NVOpen
- wolfTPM2_NVCreateAuth
- wolfTPM2_NVWriteAuth
- wolfTPM2_NVReadAuth

Return:

- TPM_RC_SUCCESS: successful
- TPM_RC_FAILURE: generic failure (check TPM IO and TPM return code)
- BAD_FUNC_ARG: check the provided arguments

```
WOLFTPM_API int wolfTPM2_NVDeleteAuth(
    WOLFTPM2_DEV * dev,
    WOLFTPM2_HANDLE * parent,
    word32 nvIndex
)
```

Destroys an existing NV Index.

Parameters:

- **dev** pointer to a TPM2_DEV struct
- **parent** pointer to a WOLFTPM2_HANDLE, specifying the TPM hierarchy for the new NV Index
- **nvIndex** integer value, holding the NV Index Handle given by the TPM upon success

See:

- wolfTPM2_NVCreateAuth
- wolfTPM2_NVWriteAuth
- wolfTPM2_NVReadAuth

Return:

- TPM_RC_SUCCESS: successful
- TPM_RC_FAILURE: generic failure (check TPM IO and TPM return code)
- BAD_FUNC_ARG: check the provided arguments

```
WOLFTPM_API int wolfTPM2_NVCreate(
    WOLFTPM2_DEV * dev,
    TPM_HANDLE authHandle,
    word32 nvIndex,
    word32 nvAttributes,
    word32 maxSize,
    const byte * auth,
    int authSz
)
```

Deprecated, use newer API.

See: wolfTPM2_NVCreateAuth

```
WOLFTPM_API int wolfTPM2_NVWrite(
    WOLFTPM2_DEV * dev,
    TPM_HANDLE authHandle,
    word32 nvIndex,
    byte * dataBuf,
    word32 dataSz,
    word32 offset
)
```

Deprecated, use newer API.

See: wolfTPM2_NVWriteAuth

```
WOLFTPM_API int wolfTPM2_NVRead(
    WOLFTPM2_DEV * dev,
    TPM_HANDLE authHandle,
    word32 nvIndex,
    byte * dataBuf,
    word32 * dataSz,
    word32 offset
)
```

Deprecated, use newer API.

See: wolfTPM2_NVReadAuth

```
WOLFTPM_API int wolfTPM2_NVDelete(
    WOLFTPM2_DEV * dev,
    TPM_HANDLE authHandle,
    word32 nvIndex
)
```

Deprecated, use newer API.

See: wolfTPM2_NVDeleteAuth

```
WOLFTPM_API int wolfTPM2_NVReadPublic(
    WOLFTPM2_DEV * dev,
    word32 nvIndex,
    TPMS_NV_PUBLIC * nvPublic
)
```

Extracts the public information about an nvIndex, such as maximum size.

Parameters:

- **dev** pointer to a TPM2_DEV struct
- **nvIndex** integer value, holding the NV Index Handle given by the TPM upon success
- **nvPublic** pointer to a TPMS_NV_PUBLIC, used to store the extracted nvIndex public information

See:

- wolfTPM2_NVCreateAuth
- wolfTPM2_NVDeleteAuth
- wolfTPM2_NVWriteAuth
- wolfTPM2_NVReadAuth

Return:

- TPM_RC_SUCCESS: successful
- TPM_RC_FAILURE: generic failure (check TPM IO and TPM return code)
- BAD_FUNC_ARG: check the provided arguments

```
WOLFTPM_API int wolfTPM2_NVStoreKey(
    WOLFTPM2_DEV * dev,
    TPM_HANDLE primaryHandle,
    WOLFTPM2_KEY * key,
    TPM_HANDLE persistentHandle
)
```

Helper function to store a TPM 2.0 Key into the TPM's NVRAM.

Parameters:

- **dev** pointer to a TPM2_DEV struct
- **primaryHandle** integer value, specifying a TPM 2.0 Hierarchy. typically TPM_RH_OWNER
- **key** pointer to a structure of WOLFTPM2_KEY type, containing the TPM 2.0 key for storing
- **persistentHandle** integer value, specifying an existing nvIndex

See:

- wolfTPM2_NVDeleteKey
- wolfTPM2_NVCreateAuth
- wolfTPM2_NVDeleteAuth

Return:

- TPM_RC_SUCCESS: successful
- TPM_RC_FAILURE: generic failure (check TPM IO and TPM return code)
- BAD_FUNC_ARG: check the provided arguments

```
WOLFTPM_API int wolfTPM2_NVDeleteKey(
    WOLFTPM2_DEV * dev,
    TPM_HANDLE primaryHandle,
    WOLFTPM2_KEY * key
)
```

Helper function to delete a TPM 2.0 Key from the TPM's NVRAM.

Parameters:

- **dev** pointer to a TPM2_DEV struct
- **primaryHandle** integer value, specifying a TPM 2.0 Hierarchy. typically TPM_RH_OWNER
- **key** pointer to a structure of WOLFTPM2_KEY type, containing the nvIndex handle value

See:

- wolfTPM2_NVDeleteKey
- wolfTPM2_NVCreateAuth
- wolfTPM2_NVDeleteAuth

Return:

- TPM_RC_SUCCESS: successful
- TPM_RC_FAILURE: generic failure (check TPM IO and TPM return code)
- BAD_FUNC_ARG: check the provided arguments

```
WOLFTPM_API struct WC_RNG * wolfTPM2_GetRng(
    WOLFTPM2_DEV * dev
)
```

Get the wolfcrypt RNG instance used for wolfTPM.

Parameters:

- **dev** pointer to a TPM2_DEV struct

See: wolfTPM2_GetRandom

Return:

- TPM_RC_SUCCESS: successful
- TPM_RC_FAILURE: generic failure (check TPM IO and TPM return code)
- BAD_FUNC_ARG: check the provided arguments

Note: Only if wolfcrypt is enabled and configured for use instead of the TPM RNG

```
WOLFTPM_API int wolfTPM2_GetRandom(
    WOLFTPM2_DEV * dev,
    byte * buf,
    word32 len
)
```

Get a set of random number, generated with the TPM RNG or wolfcrypt RNG.

Parameters:

- **dev** pointer to a TPM2_DEV struct
- **buf** pointer to a byte buffer, used to store the generated random numbers
- **len** integer value of word32 type, used to store the size of the buffer, in bytes

See: wolfTPM2_GetRandom

Return:

- TPM_RC_SUCCESS: successful
- TPM_RC_FAILURE: generic failure (check TPM IO and TPM return code)
- BAD_FUNC_ARG: check the provided arguments

Note: Define WOLFTPM2_USE_HW_RNG to use the TPM RNG source.

Warning: When using WOLFTPM2_USE_HW_RNG, random data travels over the SPI/I2C bus unprotected. If this data is used for session nonces, an attacker with physical bus access can force known values. Use default local RNG unless the bus is physically secure.

```
WOLFTPM_API int wolfTPM2_UnloadHandle(
    WOLFTPM2_DEV * dev,
    WOLFTPM2_HANDLE * handle
)
```

Use to discard any TPM loaded object.

Parameters:

- **dev** pointer to a TPM2_DEV struct
- **handle** pointer to a structure of WOLFTPM2_HANDLE type, with a valid TPM 2.0 handle value

See: wolfTPM2_UnloadHandles

Return:

- TPM_RC_SUCCESS: successful
- TPM_RC_FAILURE: generic failure (check TPM IO and TPM return code)
- BAD_FUNC_ARG: check the provided arguments

```
WOLFTPM_API int wolfTPM2_Clear(
    WOLFTPM2_DEV * dev
)
```

Clears the TPM with TPM2_Clear, authorized by the lockout hierarchy. This removes the owner and endorsement hierarchy objects and the NV indices created by the owner, and resets their authorizations. NV indices created by the platform (TPMA_NV_PLATFORMCREATE) are preserved. It does not deinitialize wolfTPM; use wolfTPM2_Cleanup for that.

Parameters:

- **dev** pointer to a TPM2_DEV struct

See: wolfTPM2_Cleanup

Return:

- TPM_RC_SUCCESS: successful
- TPM_RC_FAILURE: generic failure (check TPM IO and TPM return code)
- BAD_FUNC_ARG: check the provided arguments

```
WOLFTPM_API int wolfTPM2_DictionaryAttackLockReset(
    WOLFTPM2_DEV * dev
)
```

Reset the dictionary attack lockout failed-tries counter.

Parameters:

- **dev** pointer to a TPM2_DEV struct

See: wolfTPM2_DictionaryAttackParameters

Return:

- TPM_RC_SUCCESS: successful
- TPM_RC_LOCKOUT: lockoutAuth is in its recovery interval
- BAD_FUNC_ARG: check the provided arguments

```
WOLFTPM_API int wolfTPM2_DictionaryAttackParameters(
    WOLFTPM2_DEV * dev,
    word32 newMaxTries,
    word32 newRecoveryTime,
    word32 lockoutRecovery
)
```

Set the dictionary attack lockout parameters.

Parameters:

- **dev** pointer to a TPM2_DEV struct
- **newMaxTries** failed auths before lockout (must be non-zero)
- **newRecoveryTime** seconds to forgive one failed try (0 = reboot only)
- **lockoutRecovery** seconds before a failed lockoutAuth recovers

See: wolfTPM2_DictionaryAttackLockReset

Return:

- TPM_RC_SUCCESS: successful
- TPM_RC_LOCKOUT: lockoutAuth is in its recovery interval
- BAD_FUNC_ARG: check the provided arguments

```
WOLFTPM_API int wolfTPM2_HashStart(
    WOLFTPM2_DEV * dev,
    WOLFTPM2_HASH * hash,
    TPMI_ALG_HASH hashAlg,
    const byte * usageAuth,
    word32 usageAuthSz
)
```

Helper function to start a TPM generated hash.

Parameters:

- **dev** pointer to a TPM2_DEV struct
- **hash** pointer to a WOLFTPM2_HASH structure
- **hashAlg** integer value, specifying a valid TPM 2.0 hash algorithm
- **usageAuth** pointer to a string constant, specifying the authorization for subsequent use of the hash
- **usageAuthSz** integer value, specifying the size of the authorization, in bytes

See:

- wolfTPM2_HashUpdate
- wolfTPM2_HashFinish

Return:

- TPM_RC_SUCCESS: successful
- TPM_RC_FAILURE: generic failure (check TPM IO and TPM return code)
- BAD_FUNC_ARG: check the provided arguments

```
WOLFTPM_API int wolfTPM2_HashUpdate(
    WOLFTPM2_DEV * dev,
    WOLFTPM2_HASH * hash,
    const byte * data,
    word32 dataSz
)
```

Update a TPM generated hash with new user data.

Parameters:

- **dev** pointer to a TPM2_DEV struct
- **hash** pointer to a WOLFTPM2_HASH structure
- **data** pointer to a byte buffer, containing the user data to be added to the hash
- **dataSz** integer value of word32 type, specifying the size of the user data, in bytes

See:

- wolfTPM2_HashStart
- wolfTPM2_HashFinish

Return:

- TPM_RC_SUCCESS: successful
- TPM_RC_FAILURE: generic failure (check TPM IO and TPM return code)
- BAD_FUNC_ARG: check the provided arguments

Note: Make sure the auth is correctly set

```
WOLFTPM_API int wolfTPM2_HashFinish(
    WOLFTPM2_DEV * dev,
    WOLFTPM2_HASH * hash,
    byte * digest,
    word32 * digestSz
)
```

Finalize a TPM generated hash and get the digest output in a user buffer.

Parameters:

- **dev** pointer to a TPM2_DEV struct
- **hash** pointer to a WOLFTPM2_HASH structure
- **digest** pointer to a byte buffer, used to store the resulting digest
- **digestSz** pointer to size of digest buffer, on return set to bytes stored in digest buffer

See:

- wolfTPM2_HashStart
- wolfTPM2_HashUpdate

Return:

- TPM_RC_SUCCESS: successful
- TPM_RC_FAILURE: generic failure (check TPM IO and TPM return code)
- BAD_FUNC_ARG: check the provided arguments

Note: Make sure the auth is correctly set

```
WOLFTPM_API int wolfTPM2_LoadKeyedHashKey(
    WOLFTPM2_DEV * dev,
    WOLFTPM2_KEY * key,
    WOLFTPM2_HANDLE * parent,
    int hashAlg,
    const byte * keyBuf,
    word32 keySz,
    const byte * usageAuth,
    word32 usageAuthSz
)
```

Creates and loads a new TPM key of KeyedHash type, typically used for HMAC operations.

Parameters:

- **dev** pointer to a TPM2_DEV struct
- **key** pointer to an empty structure of WOLFTPM2_KEY type, to store the generated key
- **parent** pointer to a structure of WOLFTPM2_KEY type, containing a valid TPM handle of a primary key
- **hashAlg** integer value, specifying a valid TPM 2.0 hash algorithm
- **keyBuf** pointer to a byte array, containing derivation values for the new KeyedHash key
- **keySz** integer value, specifying the size of the derivation values stored in keyBuf, in bytes
- **usageAuth** pointer to a string constant, specifying the authorization of the new key
- **usageAuthSz** integer value, specifying the size of the authorization, in bytes

See:

- wolfTPM2_HmacStart
- wolfTPM2_HmacUpdate
- wolfTPM2_HmacFinish

Return:

- TPM_RC_SUCCESS: successful
- TPM_RC_FAILURE: generic failure (check TPM IO and TPM return code)
- BAD_FUNC_ARG: check the provided arguments

Note: To generate HMAC using the TPM it is recommended to use the wolfTPM2_Hmac wrappers

```
WOLFTPM_API int wolfTPM2_HmacStart(
    WOLFTPM2_DEV * dev,
    WOLFTPM2_HMAC * hmac,
    WOLFTPM2_HANDLE * parent,
    TPMI_ALG_HASH hashAlg,
    const byte * keyBuf,
    word32 keySz,
    const byte * usageAuth,
```

```
    word32 usageAuthSz
)
```

Helper function to start a TPM generated hmac.

Parameters:

- **dev** pointer to a TPM2_DEV struct
- **hmac** pointer to a WOLFTPM2_HMAC structure
- **parent** pointer to a structure of WOLFTPM2_KEY type, containing a valid TPM handle of a primary key
- **hashAlg** integer value, specifying a valid TPM 2.0 hash algorithm
- **keyBuf** pointer to a byte array, containing derivation values for the new KeyedHash key
- **keySz** integer value, specifying the size of the derivation values stored in keyBuf, in bytes
- **usageAuth** pointer to a string constant, specifying the authorization for subsequent use of the hmac
- **usageAuthSz** integer value, specifying the size of the authorization, in bytes

See:

- wolfTPM2_HmacUpdate
- wolfTPM2_HmacFinish
- wolfTPM2_LoadKeyedHashKey

Return:

- TPM_RC_SUCCESS: successful
- TPM_RC_FAILURE: generic failure (check TPM IO and TPM return code)
- BAD_FUNC_ARG: check the provided arguments

```
WOLFTPM_API int wolfTPM2_HmacUpdate(
    WOLFTPM2_DEV * dev,
    WOLFTPM2_HMAC * hmac,
    const byte * data,
    word32 dataSz
)
```

Update a TPM generated hmac with new user data.

Parameters:

- **dev** pointer to a TPM2_DEV struct
- **hmac** pointer to a WOLFTPM2_HMAC structure
- **data** pointer to a byte buffer, containing the user data to be added to the hmac
- **dataSz** integer value of word32 type, specifying the size of the user data, in bytes
- **dev** pointer to a TPM2_DEV struct
- **hmac** pointer to an active WOLFTPM2_HMAC structure
- **data** pointer to data to add to HMAC
- **dataSz** size of data in bytes

See:

- wolfTPM2_HmacStart
- wolfTPM2_HMACFinish
- wolfTPM2_HmacStart
- wolfTPM2_HmacFinish

Return:

- TPM_RC_SUCCESS: successful
- TPM_RC_FAILURE: generic failure (check TPM IO and TPM return code)
- BAD_FUNC_ARG: check the provided arguments
- TPM_RC_SUCCESS: successful
- TPM_RC_FAILURE: generic failure (check TPM IO and TPM return code)
- BAD_FUNC_ARG: check the provided arguments

Note:

- Make sure the TPM authorization is correctly set
- Adds data to an active HMAC sequence

Update an HMAC operation with data

```
WOLFTPM_API int wolfTPM2_HmacFinish(
    WOLFTPM2_DEV * dev,
    WOLFTPM2_HMAC * hmac,
    byte * digest,
    word32 * digestSz
)
```

Finalize a TPM generated hmac and get the digest output in a user buffer.

Parameters:

- **dev** pointer to a TPM2_DEV struct
- **hmac** pointer to a WOLFTPM2_HMAC structure
- **digest** pointer to a byte buffer, used to store the resulting hmac digest
- **digestSz** integer value of word32 type, specifying the size of the digest, in bytes

See:

- wolfTPM2_HmacStart
- wolfTPM2_HmacUpdate

Return:

- TPM_RC_SUCCESS: successful
- TPM_RC_FAILURE: generic failure (check TPM IO and TPM return code)
- BAD_FUNC_ARG: check the provided arguments

Note: Make sure the TPM authorization is correctly set

```
WOLFTPM_API int wolfTPM2_LoadSymmetricKey(
    WOLFTPM2_DEV * dev,
    WOLFTPM2_KEY * key,
    int alg,
    const byte * keyBuf,
    word32 keySz
)
```

Loads an external symmetric key into the TPM.

Parameters:

- **dev** pointer to a TPM2_DEV struct
- **key** pointer to an empty structure of WOLFTPM2_KEY type, to store the TPM handle and key information
- **alg** integer value, specifying a valid TPM 2.0 symmetric key algorithm, e.g. TPM_ALG_CFB for AES CFB
- **keyBuf** pointer to a byte array, containing private material of the symmetric key
- **keySz** integer value, specifying the size of the key material stored in keyBuf, in bytes
- **dev** pointer to a TPM2_DEV struct
- **key** pointer to an empty WOLFTPM2_KEY structure to store loaded key
- **alg** algorithm type (TPM_ALG_AES, etc)
- **keyBuf** pointer to key material
- **keySz** size of key material in bytes

See:

- wolfTPM2_EncryptDecryptBlock
- wolfTPM2_EncryptDecrypt
- TPM2_EncryptDecrypt2
- wolfTPM2_EncryptDecryptBlock
- wolfTPM2_EncryptDecrypt
- wolfTPM2_SetKeyAuthPassword

Return:

- TPM_RC_SUCCESS: successful
- TPM_RC_FAILURE: generic failure (check TPM IO and TPM return code)
- BAD_FUNC_ARG: check the provided arguments
- TPM_RC_SUCCESS: successful
- TPM_RC_FAILURE: generic failure (check TPM IO and TPM return code)
- BAD_FUNC_ARG: check the provided arguments

Note: Creates and loads a symmetric key for encryption/decryption operations

Warning: If key->handle.auth is empty the key is loaded with empty authorization and any caller holding the handle can use it. Set key->handle.auth (e.g. via wolfTPM2_SetKeyAuthPassword) before loading to require a password.

Load a symmetric key into the TPM

```
WOLFTPM_API int wolfTPM2_EncryptDecryptBlock(
    WOLFTPM2_DEV * dev,
    WOLFTPM2_KEY * key,
    const byte * in,
    byte * out,
    word32 inOutSz,
    byte * iv,
    word32 ivSz,
    int isDecrypt
)
```

Encrypt or decrypt a single block of data with the symmetric key.

Parameters:

- **dev** pointer to a TPM2_DEV (must not be NULL)
- **key** handle/template of the loaded symmetric key
- **in** input buffer
- **out** output buffer (in-place is allowed)
- **inOutSz** length of `in` / `out` in bytes
- **iv** input/output IV buffer
- **ivSz** length of `iv` in bytes
- **isDecrypt** WOLFTPM2_ENCRYPT (0) for encrypt, WOLFTPM2_DECRYPT (1) for decrypt

Return:

- TPM_RC_SUCCESS on success.
- BAD_FUNC_ARG if dev/key/in/out is NULL, inOutSz is zero, the mode needs an IV but none was supplied, or a CBC/ECB length is not block-aligned.
- BUFFER_E if `inOutSz` exceeds the TPM input buffer size or `ivSz` exceeds the TPM IV buffer size.
- other TPM_RC_* values surfaced from TPM2_EncryptDecrypt2.

Wraps TPM2_EncryptDecrypt2 (TPM 2.0 Part 3 Section 15). The TPM symmetric mode is taken from `key->pub.publicArea.parameters.symDetail.sym.mode.aes`.

Length contract:

- TPM_ALG_CBC and TPM_ALG_ECB require `inOutSz` to be a whole multiple of MAX_AES_BLOCK_SIZE_BYTES; otherwise BAD_FUNC_ARG is returned.
- TPM_ALG_CFB, TPM_ALG_OFB, TPM_ALG_CTR accept any non-zero length and the TPM-returned IV reflects exactly the bytes consumed.
- TPM_ALG_NULL is forwarded to the TPM, which rejects it.

IV requirements: modes other than ECB and NULL require a non-NULL IV. On success the IV in `iv` is updated with the TPM's running IV so the next call can chain.

```
WOLFTPM_API int wolfTPM2_SetCommand(
    WOLFTPM2_DEV * dev,
    TPM_CC commandCode,
```

```
    int enableFlag
)
```

Vendor specific TPM command, used to enable other restricted TPM commands.

Parameters:

- **dev** pointer to a TPM2_DEV struct
- **commandCode** integer value, representing a valid vendor command
- **enableFlag** integer value, non-zero values represent “to enable”

See: TPM2_GPIO_Config

Return:

- TPM_RC_SUCCESS: successful
- TPM_RC_FAILURE: generic failure (check TPM IO and TPM return code)
- BAD_FUNC_ARG: check the provided arguments

```
WOLFTPM_API int wolfTPM2_Reset(
    WOLFTPM2_DEV * dev,
    int doShutdown,
    int doStartup
)
```

Helper function to shutdown, startup or reset the TPM.

Parameters:

- **dev** pointer to a TPM2_DEV struct
- **doStartup** integer value, non-zero values represent “perform Startup after Shutdown”
- **doShutdown** integer value, non-zero values represent “perform Shutdown”

See:

- wolfTPM2_Init
- wolfTPM2_Reset

Return:

- TPM_RC_SUCCESS: successful
- TPM_RC_FAILURE: generic failure (check TPM IO and TPM return code)
- NOT_COMPILED_IN: transport where the OS owns TPM startup state
- BAD_FUNC_ARG: check the provided arguments

Note:

- The behavior depends on the doStartup and doShutdown flags:
-

  - Both flags set to 1: Performs a full TPM restart (shutdown then startup)
-
  - Only doStartup=1: Just starts up the TPM
-
  - Only doShutdown=1: Just shuts down the TPM
- On the Linux kernel driver (/dev/tpmX) and Windows TBS the OS owns TPM startup state, so no TPM command is sent and NOT_COMPILED_IN is returned. A call requesting neither a shutdown nor a startup still returns TPM_RC_SUCCESS, since nothing was declined. See docs/DEVTPM.md.

```
WOLFTPM_API int wolfTPM2_Shutdown(
    WOLFTPM2_DEV * dev,
    int doStartup
)
```

Helper function to shutdown or reset the TPM.

Parameters:

- **dev** pointer to a TPM2_DEV struct
- **doStartup** integer value, non-zero values represent “perform Startup after Shutdown”

See:

- wolfTPM2_Init
- wolfTPM2_Shutdown

Return:

- TPM_RC_SUCCESS: successful
- TPM_RC_FAILURE: generic failure (check TPM IO and TPM return code)
- NOT_COMPILED_IN: transport where the OS owns TPM startup state
- BAD_FUNC_ARG: check the provided arguments

Note:

- If doStartup is set, then TPM2_Startup is performed right after TPM2_Shutdown
- On the Linux kernel driver (/dev/tpmX) and Windows TBS the OS owns TPM startup state, so no TPM command is sent and NOT_COMPILED_IN is returned. A call requesting neither a shutdown nor a startup still returns TPM_RC_SUCCESS, since nothing was declined. See docs/DEVTPM.md.

```
WOLFTPM_API int wolfTPM2_UnloadHandles(
    WOLFTPM2_DEV * dev,
    word32 handleStart,
    word32 handleCount
)
```

One-shot API to unload subsequent TPM handles.

Parameters:

- **dev** pointer to a TPM2_DEV struct
- **handleStart** integer value of word32 type, specifying the value of the first TPM handle
- **handleCount** integer value of word32 type, specifying the number of handles

See: wolfTPM2_Init

Return:

- TPM_RC_SUCCESS: successful
- TPM_RC_FAILURE: generic failure (check TPM IO and TPM return code)
- BAD_FUNC_ARG: check the provided arguments

```
WOLFTPM_API int wolfTPM2_UnloadHandles_AllTransient(
    WOLFTPM2_DEV * dev
)
```

One-shot API to unload all transient TPM handles.

Parameters:

- **dev** pointer to a TPM2_DEV struct

See:

- wolfTPM2_UnloadHandles
- wolfTPM2_CreatePrimary

Return:

- TPM_RC_SUCCESS: successful
- TPM_RC_FAILURE: generic failure (check TPM IO and TPM return code)
- BAD_FUNC_ARG: check the provided arguments

Note:

- Flushes the transient handles the TPM reports as loaded, following TPM_CAP_HANDLES continuation pages, or the first MAX_HANDLE_NUM if the first TPM_CAP_HANDLES query fails. A failed query or flush is returned; a failed flush does not stop the remaining handles from being flushed
- If there are Primary Keys as transient objects, they need to be recreated before TPM keys can be used

```
WOLFTPM_API int wolfTPM2_GetKeyTemplate_RSA(
    TPMT_PUBLIC * publicTemplate,
    TPMA_OBJECT objectAttributes
)
```

Prepares a TPM public template for new RSA key based on user selected object attributes.

Parameters:

- **publicTemplate** pointer to an empty structure of TPMT_PUBLIC type, to store the new RSA template
- **objectAttributes** integer value of TPMA_OBJECT type, can contain one or more attributes, e.g. TPMA_OBJECT_fixedTPM

See:

- wolfTPM2_GetKeyTemplate_RSA_ex
- wolfTPM2_GetKeyTemplate_ECC
- wolfTPM2_GetKeyTemplate_Symmetric
- wolfTPM2_GetKeyTemplate_KeyedHash
- wolfTPM2_GetKeyTemplate_KeySeal

Return:

- TPM_RC_SUCCESS: successful
- BAD_FUNC_ARG: check the provided arguments

```
WOLFTPM_API int wolfTPM2_GetKeyTemplate_RSA_ex(
    TPMT_PUBLIC * publicTemplate,
    TPM_ALG_ID nameAlg,
    TPMA_OBJECT objectAttributes,
    int keyBits,
    long exponent,
    TPM_ALG_ID sigScheme,
    TPM_ALG_ID sigHash
)
```

Prepares a TPM public template for new RSA key based on user selected object attributes.

Parameters:

- **publicTemplate** pointer to an empty structure of TPMT_PUBLIC type, to store the new RSA template
- **nameAlg** integer value of TPM_ALG_ID type, specifying a TPM supported hashing algorithm, typically TPM_ALG_SHA256 for SHA 256
- **objectAttributes** integer value of TPMA_OBJECT type, can contain one or more attributes, e.g. TPMA_OBJECT_fixedTPM
- **keyBits** integer value, specifying the size of the symmetric key, typically 128 or 256 bits
- **exponent** integer value of word32 type, specifying the RSA exponent
- **sigScheme** integer value of TPM_ALG_ID type, specifying a TPM supported signature scheme

- **sigHash** integer value of TPM_ALG_ID type, specifying a TPM supported signature hash scheme

See:

- wolfTPM2_GetKeyTemplate_RSA
- wolfTPM2_GetKeyTemplate_ECC
- wolfTPM2_GetKeyTemplate_ECC_ex
- wolfTPM2_GetKeyTemplate_Symmetric
- wolfTPM2_GetKeyTemplate_KeyedHash
- wolfTPM2_GetKeyTemplate_KeySeal

Return:

- TPM_RC_SUCCESS: successful
- BAD_FUNC_ARG: check the provided arguments

```
WOLFTPM_API int wolfTPM2_GetKeyTemplate_ECC(
    TPMT_PUBLIC * publicTemplate,
    TPMA_OBJECT objectAttributes,
    TPM_ECC_CURVE curve,
    TPM_ALG_ID sigScheme
)
```

Prepares a TPM public template for new ECC key based on user selected object attributes.

Parameters:

- **publicTemplate** pointer to an empty structure of TPMT_PUBLIC type, to store the new ECC key template
- **objectAttributes** integer value of TPMA_OBJECT type, can contain one or more attributes, e.g. TPMA_OBJECT_fixedTPM
- **curve** integer value of TPM_ECC_CURVE type, specifying a TPM supported ECC curve ID; the requested curve is used exactly as given (the configurable WOLFTPM2_ECC_DEFAULT_CURVE applies only to the no-curve SRK/AIK templates, not to this explicit-curve API)
- **sigScheme** integer value of TPM_ALG_ID type, specifying a TPM supported signature scheme

See:

- wolfTPM2_GetKeyTemplate_ECC_ex
- wolfTPM2_GetKeyTemplate_RSA
- wolfTPM2_GetKeyTemplate_Symmetric
- wolfTPM2_GetKeyTemplate_KeyedHash
- wolfTPM2_GetKeyTemplate_KeySeal

Return:

- TPM_RC_SUCCESS: successful
- BAD_FUNC_ARG: check the provided arguments

```
WOLFTPM_API int wolfTPM2_GetKeyTemplate_ECC_ex(
    TPMT_PUBLIC * publicTemplate,
    TPM_ALG_ID nameAlg,
    TPMA_OBJECT objectAttributes,
    TPM_ECC_CURVE curve,
    TPM_ALG_ID sigScheme,
    TPM_ALG_ID sigHash
)
```

Prepares a TPM public template for new ECC key based on user selected object attributes.

Parameters:

- **publicTemplate** pointer to an empty structure of TPMT_PUBLIC type, to store the new ECC key template
- **nameAlg** integer value of TPM_ALG_ID type, specifying a TPM supported hashing algorithm, typically TPM_ALG_SHA256 for SHA 256
- **objectAttributes** integer value of TPMA_OBJECT type, can contain one or more attributes, e.g. TPMA_OBJECT_fixedTPM
- **curve** integer value of TPM_ECC_CURVE type, specifying a TPM supported ECC curve ID; the requested curve is used exactly as given (the configurable WOLFTPM2_ECC_DEFAULT_CURVE applies only to the no-curve SRK/AIK templates, not to this explicit-curve API)
- **sigScheme** integer value of TPM_ALG_ID type, specifying a TPM supported signature scheme
- **sigHash** integer value of TPM_ALG_ID type, specifying a TPM supported signature hash scheme

See:

- wolfTPM2_GetKeyTemplate_ECC
- wolfTPM2_GetKeyTemplate_RSA
- wolfTPM2_GetKeyTemplate_Symmetric
- wolfTPM2_GetKeyTemplate_KeyedHash
- wolfTPM2_GetKeyTemplate_KeySeal

Return:

- TPM_RC_SUCCESS: successful
- BAD_FUNC_ARG: check the provided arguments

```
WOLFTPM_API int wolfTPM2_GetKeyTemplate_Symmetric(
    TPMT_PUBLIC * publicTemplate,
    int keyBits,
    TPM_ALG_ID algMode,
    int isSign,
    int isDecrypt
)
```

Prepares a TPM public template for new Symmetric key.

Parameters:

- **publicTemplate** pointer to an empty structure of TPMT_PUBLIC type, to store the new Symmetric key template
- **keyBits** integer value, specifying the size of the symmetric key, typically 128 or 256 bits
- **algMode** integer value of TPM_ALG_ID type, specifying a TPM supported symmetric algorithm, e.g. TPM_ALG_CFB for AES CFB
- **isSign** integer value, non-zero values represent "a signing key"
- **isDecrypt** integer value, non-zero values represent "a decryption key"

See:

- wolfTPM2_GetKeyTemplate_RSA
- wolfTPM2_GetKeyTemplate_ECC
- wolfTPM2_GetKeyTemplate_KeyedHash
- wolfTPM2_GetKeyTemplate_KeySeal

Return:

- TPM_RC_SUCCESS: successful
- BAD_FUNC_ARG: check the provided arguments

```
WOLFTPM_API int wolfTPM2_GetKeyTemplate_KeyedHash(
    TPMT_PUBLIC * publicTemplate,
    TPM_ALG_ID hashAlg,
    int isSign,
    int isDecrypt
)
```

Prepares a TPM public template for new KeyedHash key.

Parameters:

- **publicTemplate** pointer to an empty structure of TPMT_PUBLIC type, to store the new template
- **hashAlg** integer value of TPM_ALG_ID type, specifying a TPM supported hashing algorithm, e.g. TPM_ALG_SHA256 for SHA 256
- **isSign** integer value, non-zero values represent "a signing key"
- **isDecrypt** integer value, non-zero values represent "a decryption key"

See:

- wolfTPM2_GetKeyTemplate_RSA
- wolfTPM2_GetKeyTemplate_ECC
- wolfTPM2_GetKeyTemplate_Symmetric
- wolfTPM2_GetKeyTemplate_KeySeal

Return:

- TPM_RC_SUCCESS: successful
- BAD_FUNC_ARG: check the provided arguments

```
WOLFTPM_API int wolfTPM2_GetKeyTemplate_KeySeal(
    TPMT_PUBLIC * publicTemplate,
    TPM_ALG_ID nameAlg
)
```

Prepares a TPM public template for new key for sealing secrets.

Parameters:

- **publicTemplate** pointer to an empty structure of TPMT_PUBLIC type, to store the new template
- **nameAlg** integer value of TPM_ALG_ID type, specifying a TPM supported hashing algorithm, typically TPM_ALG_SHA256 for SHA 256

See:

- wolfTPM2_GetKeyTemplate_ECC
- wolfTPM2_GetKeyTemplate_Symmetric
- wolfTPM2_GetKeyTemplate_KeyedHash
- wolfTPM2_GetKeyTemplate_KeySeal

Return:

- TPM_RC_SUCCESS: successful
- BAD_FUNC_ARG: check the provided arguments

Note: There are strict requirements for a Key Seal, therefore most of the key parameters are predetermined by the wrapper

```
WOLFTPM_API int wolfTPM2_GetKeyTemplate_EK(
    TPMT_PUBLIC * publicTemplate,
    TPM_ALG_ID alg,
    int keyBits,
    TPM_ECC_CURVE curveID,
    TPM_ALG_ID nameAlg,
    int highRange
)
```

Prepares a TPM public template for generating the TPM Endorsement Key.

Parameters:

- **publicTemplate** pointer to an empty structure of TPMT_PUBLIC type, to store the new template
- **alg** can be only TPM_ALG_RSA or TPM_ALG_ECC, see Note above
- **keyBits** integer value, specifying bits for the key, typically 2048 (RSA) or 256 (ECC)
- **curveID** use one of the accepted TPM_ECC_CURVE values like TPM_ECC_NIST_P256 (only used when alg=TPM_ALG_ECC)
- **nameAlg** integer value of TPMI_ALG_HASH type, specifying a valid TPM2 hashing algorithm (typically TPM_ALG_SHA256)
- **highRange** integer value: 0=low range, 1=high range

See:

- wolfTPM2_GetKeyTemplate_ECC_EK
- wolfTPM2_GetKeyTemplate_RSA_SRK
- wolfTPM2_GetKeyTemplate_RSA_AIK
- wolfTPM2_GetKeyTemplate_EKIndex

Return:

- TPM_RC_SUCCESS: successful
- BAD_FUNC_ARG: check the provided arguments

```
WOLFTPM_API int wolfTPM2_GetKeyTemplate_EKIndex(
    word32 nvIndex,
    TPMT_PUBLIC * publicTemplate
)
```

Helper to get the Endorsement public key template by NV index.

Parameters:

- **nvIndex** handle for NV index. Typically starting from TPM_20_TCG_NV_SPACE
- **publicTemplate** pointer to an empty structure of TPMT_PUBLIC type, to store the new template

See:

- wolfTPM2_GetKeyTemplate_EK
- wolfTPM2_GetKeyTemplate_ECC_EK
- wolfTPM2_GetKeyTemplate_RSA_SRK
- wolfTPM2_GetKeyTemplate_RSA_AIK

Return:

- TPM_RC_SUCCESS: successful
- BAD_FUNC_ARG: check the provided arguments

```
WOLFTPM_API int wolfTPM2_GetKeyTemplate_RSA_EK(
    TPMT_PUBLIC * publicTemplate
)
```

Prepares a TPM public template for generating the TPM Endorsement Key of RSA type.

Parameters:

- **publicTemplate** pointer to an empty structure of TPMT_PUBLIC type, to store the new template

See:

- wolfTPM2_GetKeyTemplate_EK
- wolfTPM2_GetKeyTemplate_ECC_EK
- wolfTPM2_GetKeyTemplate_RSA_SRK
- wolfTPM2_GetKeyTemplate_RSA_AIK

Return:

- TPM_RC_SUCCESS: successful
- BAD_FUNC_ARG: check the provided arguments

```
WOLFTPM_API int wolfTPM2_GetKeyTemplate_ECC_EK(
    TPMT_PUBLIC * publicTemplate
)
```

Prepares a TPM public template for generating the TPM Endorsement Key of ECC type.

Parameters:

- **publicTemplate** pointer to an empty structure of TPMT_PUBLIC type, to store the new template

See:

- wolfTPM2_GetKeyTemplate_EK
- wolfTPM2_GetKeyTemplate_RSA_EK
- wolfTPM2_GetKeyTemplate_ECC_SRK
- wolfTPM2_GetKeyTemplate_ECC_AIK

Return:

- TPM_RC_SUCCESS: successful
- BAD_FUNC_ARG: check the provided arguments

```
WOLFTPM_API int wolfTPM2_GetKeyTemplate_RSA_SRK(
    TPMT_PUBLIC * publicTemplate
)
```

Prepares a TPM public template for generating a new TPM Storage Key of RSA type.

Parameters:

- **publicTemplate** pointer to an empty structure of TPMT_PUBLIC type, to store the new template

See:

- wolfTPM2_GetKeyTemplate_ECC_SRK
- wolfTPM2_GetKeyTemplate_RSA_AIK
- wolfTPM2_GetKeyTemplate_RSA_EK

Return:

- TPM_RC_SUCCESS: successful
- BAD_FUNC_ARG: check the provided arguments

```
WOLFTPM_API int wolfTPM2_GetKeyTemplate_ECC_SRK(
    TPMT_PUBLIC * publicTemplate
)
```

Prepares a TPM public template for generating a new TPM Storage Key of ECC type.

Parameters:

- **publicTemplate** pointer to an empty structure of TPMT_PUBLIC type, to store the new template

See:

- wolfTPM2_GetKeyTemplate_RSA_SRK
- wolfTPM2_GetKeyTemplate_ECC_AIK
- wolfTPM2_GetKeyTemplate_ECC_EK

Return:

- TPM_RC_SUCCESS: successful
- BAD_FUNC_ARG: check the provided arguments

```
WOLFTPM_API int wolfTPM2_GetKeyTemplate_RSA_AIK(
    TPMT_PUBLIC * publicTemplate
)
```

Prepares a TPM public template for generating a new TPM Attestation Key of RSA type.

Parameters:

- **publicTemplate** pointer to an empty structure of TPMT_PUBLIC type, to store the new template

See:

- wolfTPM2_GetKeyTemplate_ECC_AIK
- wolfTPM2_GetKeyTemplate_RSA_SRK

Return:

- TPM_RC_SUCCESS: successful
- BAD_FUNC_ARG: check the provided arguments

```
WOLFTPM_API int wolfTPM2_GetKeyTemplate_ECC_AIK(
    TPMT_PUBLIC * publicTemplate
)
```

Prepares a TPM public template for generating a new TPM Attestation Key of ECC type.

Parameters:

- **publicTemplate** pointer to an empty structure of TPMT_PUBLIC type, to store the new template

See:

- wolfTPM2_GetKeyTemplate_RSA_AIK
- wolfTPM2_GetKeyTemplate_ECC_SRK

Return:

- TPM_RC_SUCCESS: successful
- BAD_FUNC_ARG: check the provided arguments

```
WOLFTPM_API int wolfTPM2_SetKeyTemplate_Unique(
    TPMT_PUBLIC * publicTemplate,
    const byte * unique,
    int uniqueSz
)
```

Sets the unique area of a public template used by Create or CreatePrimary.

Parameters:

- **publicTemplate** pointer to an empty structure of TPMT_PUBLIC type, to store the new template
- **unique** optional pointer to buffer to populate unique area of public template. If NULL, the buffer will be zeroized.
- **uniqueSz** size to fill the unique field. If zero the key size is used.

See:

- wolfTPM2_CreateKey
- wolfTPM2_CreatePrimaryKey

Return:

- TPM_RC_SUCCESS: successful
- BAD_FUNC_ARG: check the provided arguments

```
WOLFTPM_API int wolfTPM2_GetNvAttributesTemplate(
    TPM_HANDLE auth,
    word32 * nvAttributes
)
```

Prepares a TPM NV Index template.

Parameters:

- **auth** integer value, representing the TPM Hierarchy under which the new TPM NV index will be created
- **nvAttributes** pointer to an empty integer variable, to store the NV Attributes

See:

- wolfTPM2_CreateAuth
- wolfTPM2_WriteAuth
- wolfTPM2_ReadAuth
- wolfTPM2_DeleteAuth

Return:

- TPM_RC_SUCCESS: successful
- BAD_FUNC_ARG: check the provided arguments

```
WOLFTPM_API int wolfTPM2_CreateEK(
    WOLFTPM2_DEV * dev,
    WOLFTPM2_KEY * ekKey,
    TPM_ALG_ID alg
)
```

Generates a new TPM Endorsement key, based on the user selected algorithm, RSA or ECC.

Parameters:

- **dev** pointer to a TPM2_DEV struct
- **ekKey** pointer to an empty WOLFTPM2_KEY structure, to store information about the new EK
- **alg** can be only TPM_ALG_RSA or TPM_ALG_ECC, see Note above

See:

- wolfTPM2_CreateSRK
- wolfTPM2_GetKeyTemplate_RSA_EK
- wolfTPM2_GetKeyTemplate_ECC_EK

Return:

- TPM_RC_SUCCESS: successful
- BAD_FUNC_ARG: check the provided arguments

Note: Although only RSA and ECC can be used for EK, symmetric keys can be created and used by the TPM

```
WOLFTPM_API int wolfTPM2_CreateSRK(
    WOLFTPM2_DEV * dev,
    WOLFTPM2_KEY * srkKey,
    TPM_ALG_ID alg,
    const byte * auth,
    int authSz
)
```

Generates a new TPM Primary Key that will be used as a Storage Key for other TPM keys.

Parameters:

- **dev** pointer to a TPM2_DEV struct
- **srkKey** pointer to an empty WOLFTPM2_KEY structure, to store information about the new EK
- **alg** can be only TPM_ALG_RSA or TPM_ALG_ECC, see Note above
- **auth** pointer to a string constant, specifying the password authorization for the TPM 2.0 Key
- **authSz** integer value, specifying the size of the password authorization, in bytes

See:

- wolfTPM2_CreateEK
- wolfTPM2_CreateAndLoadAIK
- wolfTPM2_GetKeyTemplate_RSA_SRK
- wolfTPM2_GetKeyTemplate_ECC_SRK

Return:

- TPM_RC_SUCCESS: successful
- TPM_RC_FAILURE: generic failure (check TPM IO and TPM return code)
- BAD_FUNC_ARG: check the provided arguments

Note: Although only RSA and ECC can be used for EK, symmetric keys can be created and used by the TPM

```
WOLFTPM_API int wolfTPM2_CreateAndLoadAIK(
    WOLFTPM2_DEV * dev,
    WOLFTPM2_KEY * aikKey,
    TPM_ALG_ID alg,
    WOLFTPM2_KEY * srkKey,
    const byte * auth,
    int authSz
)
```

Generates a new TPM Attestation Key under the provided Storage Key.

Parameters:

- **dev** pointer to a TPM2_DEV struct
- **aikKey** pointer to an empty WOLFTPM2_KEY structure, to store the newly generated TPM key
- **alg** can be only TPM_ALG_RSA or TPM_ALG_ECC
- **srkKey** pointer to a WOLFTPM2_KEY structure, pointing to valid TPM handle of a loaded Storage Key
- **auth** pointer to a string constant, specifying the password authorization for the TPM 2.0 Key
- **authSz** integer value, specifying the size of the password authorization, in bytes

See:

- wolfTPM2_CreateSRK
- wolfTPM2_GetKeyTemplate_RSA_AIK
- wolfTPM2_GetKeyTemplate_ECC_AIK

Return:

- TPM_RC_SUCCESS: successful
- TPM_RC_FAILURE: generic failure (check TPM IO and TPM return code)
- BAD_FUNC_ARG: check the provided arguments

```
WOLFTPM_API int wolfTPM2_GetTime(
    WOLFTPM2_KEY * aikKey,
    GetTime_Out * getTimeOut
)
```

One-shot API to generate a TPM signed timestamp.

Parameters:

- **aikKey** pointer to a WOLFTPM2_KEY structure, containing valid TPM handle of a loaded attestation key
- **getTimeOut** pointer to an empty structure of GetTime_Out type, to store the output of the command

See:

- wolfTPM2_CreateSRK
- wolfTPM2_GetKeyTemplate_RSA_EK
- wolfTPM2_GetKeyTemplate_ECC_EK

Return:

- TPM_RC_SUCCESS: successful
- TPM_RC_FAILURE: generic failure (check TPM IO and TPM return code)
- BAD_FUNC_ARG: check the provided arguments

Note: The attestation key must be generated and loaded prior to this call

```
WOLFTPM_API int wolfTPM2_CSR_SetCustomExt(
    WOLFTPM2_DEV * dev,
    WOLFTPM2_CSR * csr,
    int critical,
    const char * oid,
    const byte * der,
    word32 derSz
)
```

Helper for Certificate Signing Request (CSR) generation to set a custom request extension oid and value usage for a WOLFTPM2_CSR structure.

Parameters:

- **dev** pointer to a TPM2_DEV struct (not used)
- **csr** pointer to a WOLFTPM2_CSR structure
- **critical** If 0, the extension will not be marked critical, otherwise it will be marked critical.
- **oid** Dot separated oid as a string. For example "1.2.840.10045.3.1.7"
- **der** The der encoding of the content of the extension.
- **derSz** The size in bytes of the der encoding.

See:

- wolfTPM2_CSR_SetSubject
- wolfTPM2_CSR_SetKeyUsage
- wolfTPM2_CSR_MakeAndSign
- wolfTPM2_CSR_MakeAndSign_ex

Return:

- TPM_RC_SUCCESS: successful
- BAD_FUNC_ARG: check the provided arguments

```
WOLFTPM_API int wolfTPM2_CSR_SetKeyUsage(
    WOLFTPM2_DEV * dev,
    WOLFTPM2_CSR * csr,
    const char * keyUsage
)
```

Helper for Certificate Signing Request (CSR) generation to set a extended key usage or key usage for a WOLFTPM2_CSR structure. Pass either extended key usage or key usage values. Mixed string types are not supported, however you can call `wolfTPM2_CSR_SetKeyUsage` twice (once for extended key usage strings and once for standard key usage strings).

Parameters:

- **dev** pointer to a TPM2_DEV struct (not used)
- **csr** pointer to a WOLFTPM2_CSR structure

- **keyUsage** string list of comma separated key usage attributes. Possible Extended Key Usage values: any, serverAuth, clientAuth, codeSigning, emailProtection, timeStamping and OCSPSigning Possible Key Usage values: digitalSignature, nonRepudiation, contentCommitment, keyEncipherment, dataEncipherment, keyAgreement, keyCertSign, cRLSign, encipherOnly, decipherOnly Default: "serverAuth,clientAuth,codeSigning"

See:

- wolfTPM2_CSR_SetSubject
- wolfTPM2_CSR_SetCustomExt
- wolfTPM2_CSR_MakeAndSign
- wolfTPM2_CSR_MakeAndSign_ex

Return:

- TPM_RC_SUCCESS: successful
- BAD_FUNC_ARG: check the provided arguments

```
WOLFTPM_API int wolfTPM2_CSR_SetSubject(
    WOLFTPM2_DEV * dev,
    WOLFTPM2_CSR * csr,
    const char * subject
)
```

Helper for Certificate Signing Request (CSR) generation to set a subject for a WOLFTPM2_CSR structure.

Parameters:

- **dev** pointer to a TPM2_DEV struct (not used)
- **csr** pointer to a WOLFTPM2_CSR structure
- **subject** distinguished name string using /CN= syntax. Example: "/C=US/ST=Washington/L=Seattle/O=wolfSSL/O

See:

- wolfTPM2_CSR_SetKeyUsage
- wolfTPM2_CSR_SetCustomExt
- wolfTPM2_CSR_MakeAndSign
- wolfTPM2_CSR_MakeAndSign_ex

Return:

- TPM_RC_SUCCESS: successful
- BAD_FUNC_ARG: check the provided arguments

```
WOLFTPM_API int wolfTPM2_CSR_MakeAndSign_ex(
    WOLFTPM2_DEV * dev,
    WOLFTPM2_CSR * csr,
    WOLFTPM2_KEY * key,
```

```
    int outFormat,
    byte * out,
    int outSz,
    int sigType,
    int selfSignCert,
    int devId
)
```

Helper for Certificate Signing Request (CSR) generation using a TPM based key (WOLFTPM2_KEY structure with subject and key usage already set.

Parameters:

- **dev** pointer to a TPM2_DEV struct
- **csr** pointer to a WOLFTPM2_CSR structure
- **key** WOLFTPM2_KEY structure
- **outFormat** CTC_FILETYPE_ASN1 or CTC_FILETYPE_PEM
- **out** destination buffer for CSR as ASN.1/DER or PEM
- **outSz** destination buffer maximum size
- **sigType** Use 0 to automatically select SHA2-256 based on keyType (CTC_SHA256wRSA or CTC_SHA256wECDSA). See wolfCrypt "enum Ctc_SigType" for list of possible values.
- **selfSignCert** If set to 1 (non-zero) then result will be a self signed certificate. Zero (0) will generate a CSR (Certificate Signing Request) to be used by a CA.
- **devId** The device identifier used when registering the crypto callback. Use INVALID_DEVID (-2) to automatically register the required crypto callback.

See:

- wolfTPM2_CSR_SetSubject
- wolfTPM2_CSR_SetKeyUsage
- wolfTPM2_CSR_SetCustomExt
- wolfTPM2_CSR_MakeAndSign

Return:

- Success: Positive integer (size of the output)
- TPM_RC_FAILURE: generic failure (check TPM IO and TPM return code)
- BAD_FUNC_ARG: check the provided arguments

```
WOLFTPM_API int wolfTPM2_CSR_MakeAndSign(
    WOLFTPM2_DEV * dev,
    WOLFTPM2_CSR * csr,
    WOLFTPM2_KEY * key,
    int outFormat,
    byte * out,
    int outSz
)
```

Helper for Certificate Signing Request (CSR) generation using a TPM based key (WOLFTPM2_KEY structure with subject and key usage already set.

Parameters:

- **dev** pointer to a TPM2_DEV struct
- **csr** pointer to a WOLFTPM2_CSR structure
- **key** WOLFTPM2_KEY structure
- **outFormat** CTC_FILETYPE_ASN1 or CTC_FILETYPE_PEM
- **out** destination buffer for CSR as ASN.1/DER or PEM
- **outSz** destination buffer maximum size

See:

- wolfTPM2_CSR_SetSubject
- wolfTPM2_CSR_SetKeyUsage
- wolfTPM2_CSR_SetCustomExt
- wolfTPM2_CSR_MakeAndSign_ex

Return:

- Success: Positive integer (size of the output)
- TPM_RC_FAILURE: generic failure (check TPM IO and TPM return code)
- BAD_FUNC_ARG: check the provided arguments

```
WOLFTPM_API int wolfTPM2_CSR_Generate_ex(
    WOLFTPM2_DEV * dev,
    WOLFTPM2_KEY * key,
    const char * subject,
    const char * keyUsage,
    int outFormat,
    byte * out,
    int outSz,
    int sigType,
    int selfSignCert,
    int devId
)
```

Helper for Certificate Signing Request (CSR) generation using a TPM based key (WOLFTPM2_KEY). Single shot API for outputting a CSR or self-signed cert based on TPM key.

Parameters:

- **dev** pointer to a TPM2_DEV struct
- **key** pointer to a loaded WOLFTPM2_KEY structure
- **subject** distinguished name string using /CN= syntax. Example: "/C=US/ST=Washington/L=Seattle/O=wolfSSL/O
- **keyUsage** string list of comma separated key usage attributes. Possible values: any, serverAuth, clientAuth, codeSigning, emailProtection, timeStamping and OCSPSigning Default: "serverAuth,clientAuth,codeSigning"
- **outFormat** CTC_FILETYPE_ASN1 or CTC_FILETYPE_PEM
- **out** destination buffer for CSR as ASN.1/DER or PEM
- **outSz** destination buffer maximum size

- **sigType** Use 0 to automatically select SHA2-256 based on keyType (CTC_SHA256wRSA or CTC_SHA256wECDSA). See wolfCrypt “enum Ctc_SigType” for list of possible values.
- **selfSignCert** If set to 1 (non-zero) then result will be a self signed certificate. Zero (0) will generate a CSR (Certificate Signing Request) to be used by a CA.
- **devId** The device identifier used when registering the crypto callback. Use INVALID_DEVID (-2) to automatically register the required crypto callback.
- **dev** pointer to a TPM2_DEV struct
- **key** pointer to a loaded WOLFTPM2_KEY structure
- **subject** distinguished name string using /CN= syntax
- **keyUsage** string list of comma separated key usage attributes
- **outFormat** output format (CTC_FILETYPE_ASN1 or CTC_FILETYPE_PEM)
- **out** pointer to buffer for CSR/cert output
- **outSz** size of output buffer
- **sigType** signature algorithm (0 for default SHA2-256)
- **selfSignCert** If 1, generate self-signed cert; if 0, generate CSR
- **devId** device ID for crypto callback (-2 for auto-register)

See:

- wolfTPM2_SetCryptoDevCb
- wolfTPM2_CSR_Generate
- wolfTPM2_CSR_Generate
- wolfTPM2_CSR_MakeAndSign_ex

Return:

- Success: Positive integer (size of the output)
- TPM_RC_FAILURE: generic failure (check TPM IO and TPM return code)
- BAD_FUNC_ARG: check the provided arguments
- Success: Positive integer (size of the output)
- TPM_RC_FAILURE: generic failure (check TPM IO and TPM return code)
- BAD_FUNC_ARG: check the provided arguments

Note: Single shot API for outputting a CSR or self-signed cert based on TPM key

Generate a Certificate Signing Request (CSR) or self-signed certificate with extended options

```
WOLFTPM_API int wolfTPM2_CSR_Generate(
    WOLFTPM2_DEV * dev,
    WOLFTPM2_KEY * key,
    const char * subject,
    const char * keyUsage,
    int outFormat,
    byte * out,
    int outSz
)
```

Helper for Certificate Signing Request (CSR) generation using a TPM based key (WOLFTPM2_KEY). Single shot API for outputting a CSR or self-signed cert based on TPM key.

Parameters:

- **dev** pointer to a TPM2_DEV struct
- **key** pointer to a loaded WOLFTPM2_KEY structure
- **subject** distinguished name string using /CN= syntax. Example: "/C=US/ST=Washington/L=Seattle/O=wolfSSL/O
- **keyUsage** string list of comma separated key usage attributes. Possible values: any, serverAuth, clientAuth, codeSigning, emailProtection, timeStamping and OCSPSigning Default: "serverAuth,clientAuth,codeSigning"
- **outFormat** CTC_FILETYPE_ASN1 or CTC_FILETYPE_PEM
- **out** destination buffer for CSR as ASN.1/DER or PEM
- **outSz** destination buffer maximum size

See:

- wolfTPM2_SetCryptoDevCb
- wolfTPM2_CSR_Generate_ex

Return:

- Success: Positive integer (size of the output)
- TPM_RC_FAILURE: generic failure (check TPM IO and TPM return code)
- BAD_FUNC_ARG: check the provided arguments

```
WOLFTPM_API int wolfTPM2_ChangePlatformAuth(
    WOLFTPM2_DEV * dev,
    WOLFTPM2_SESSION * session
)
```

Helper to set the platform hierarchy authentication value to random. Setting the platform auth to random value is used to prevent application from being able to use platform hierarchy. This is defined in section 10 of the TCG PC Client Platform specification.

Parameters:

- **dev** pointer to a TPM2_DEV struct
- **session** the current session, a session is required to protect the new platform auth

See:

- TPM2_HierarchyChangeAuth
- wolfTPM2_ChangeHierarchyAuth

Return:

- TPM_RC_SUCCESS (0) on success
- TPM_RC_FAILURE: generic failure (check TPM IO and TPM return code)
- BAD_FUNC_ARG: check the provided arguments

```
WOLFTPM_API int wolfTPM2_ChangeHierarchyAuth(
    WOLFTPM2_DEV * dev,
    WOLFTPM2_SESSION * session,
    TPMI_RH_HIERARCHY_AUTH authHandle
)
```

Helper to set the hierarchy authentication value to random. Setting the platform auth to random value is used to prevent application from being able to use platform hierarchy. This is defined in section 10 of the TCG PC Client Platform specification.

Parameters:

- **dev** pointer to a TPM2_DEV struct
- **session** the current session, a session is required to protect the new platform auth
- **authHandle** the auth hierarchy (example: TPM_RH_PLATFORM or TPM_RH_LOCKOUT)

See:

- TPM2_HierarchyChangeAuth
- wolfTPM2_ChangePlatformAuth

Return:

- TPM_RC_SUCCESS (0) on success
- TPM_RC_FAILURE: generic failure (check TPM IO and TPM return code)
- BAD_FUNC_ARG: check the provided arguments

```
WOLFTPM_API int wolfTPM2_CryptoDevCb(
    int devId,
    wc_CryptoInfo * info,
    void * ctx
)
```

A reference crypto callback API for using the TPM for crypto offload. This callback function is registered using wolfTPM2_SetCryptoDevCb or wc_CryptoDev_RegisterDevice.

Parameters:

- **devId** The devId used when registering the callback. Any signed integer value besides INVALID_DEVID
- **info** point to wc_CryptoInfo structure with detailed information about crypto type and parameters
- **ctx** The user context supplied when callback was registered with wolfTPM2_SetCryptoDevCb

See:

- wolfTPM2_SetCryptoDevCb
- wolfTPM2_ClearCryptoDevCb

Return:

- TPM_RC_SUCCESS: successful
- CRYPTOCB_UNAVAILABLE: Do not use TPM hardware, fall-back to default software crypto.
- WC_HW_E: generic hardware failure

Note: Key generation, signing and RSA operations on an ECC, RSA or ML-DSA key whose key id is a 4-byte big-endian persistent or transient handle use the TPM key at that handle with an empty authorization value. RSA public operations on an RsaKey that has a modulus use that modulus instead. The TPM key must be unrestricted and must fit the operation: RSA with the NULL scheme and the decrypt attribute, ECC with an ECDSA scheme or the NULL scheme, which signs with the curve's hash, where the hash is no shorter than the input and no longer than the curve, or pure ML-DSA with the parameter set given to wc_MlDsaKey_SetParams. The operation also fails when the wolfCrypt key holds a different public key or only a private key; without either, a transient id names whatever key is loaded at that handle. Key generation creates the key under storageKey with an empty auth and noDA. A persistent handle stores it there, which needs an empty owner authorization. WOLFTPM2_KEYID_TRANSIENT leaves it loaded at a handle the TPM picks and writes that handle to the key id; the caller unloads it with wolfTPM2_UnloadHandle. Key generation fails for any other key id, for a wolfCrypt key that already holds a key and for a curve shorter than its hash (P-192, P-224).

```
WOLFTPM_API int wolfTPM2_SetCryptoDevCb(
    WOLFTPM2_DEV * dev,
    CryptoDevCallbackFunc cb,
    TpmCryptoDevCtx * tpmCtx,
    int * pDevId
)
```

Register a crypto callback function and return assigned devId.

Parameters:

- **dev** pointer to a TPM2_DEV struct
- **cb** The wolfTPM2_CryptoDevCb API is a template, but you can also provide your own
- **tpmCtx** The user supplied context. For wolfTPM2_CryptoDevCb use TpmCryptoDevCtx, but can also be your own.
- **pDevId** Pointer to automatically assigned device ID.

See:

- wolfTPM2_CryptoDevCb
- wolfTPM2_ClearCryptoDevCb

Return:

- TPM_RC_SUCCESS: successful
- TPM_RC_FAILURE: generic failure (check TPM IO and TPM return code)
- BAD_FUNC_ARG: check the provided arguments

```
WOLFTPM_API int wolfTPM2_ClearCryptoDevCb(
    WOLFTPM2_DEV * dev,
    int devId
)
```

Clears the registered crypto callback.

Parameters:

- **dev** pointer to a TPM2_DEV struct
- **devId** The devId used when registering the callback

See:

- wolfTPM2_CryptoDevCb
- wolfTPM2_SetCryptoDevCb

Return:

- TPM_RC_SUCCESS: successful
- TPM_RC_FAILURE: generic failure (check TPM IO and TPM return code)
- BAD_FUNC_ARG: check the provided arguments

```
WOLFTPM_API WOLFTPM2_DEV * wolfTPM2_New(
    void
)
```

Allocate and initialize a WOLFTPM2_DEV.

See:

- wolfTPM2_Free
- wolfTPM2_InitWithSpdmKey
- wolfTPM2_InitWithSpdmPsk

Return:

- pointer to new device struct
- NULL: on any error

Note: This convenience constructor uses uncredentialed wolfTPM2_Init. An SPDM-only TPM requires a caller-allocated WOLFTPM2_DEV initialized with wolfTPM2_InitWithSpdmKey, wolfTPM2_InitWithSpdmKey_ex, or wolfTPM2_InitWithSpdmPsk instead.

```
WOLFTPM_API int wolfTPM2_Free(
    WOLFTPM2_DEV * dev
)
```

Cleanup and Free a WOLFTPM2_DEV that was allocated by wolfTPM2_New.

Parameters:

- **dev** pointer to a TPM2_DEV struct

See: wolfTPM2_New

Return: TPM_RC_SUCCESS: successful

```
WOLFTPM_API WOLFTPM2_KEYBLOB * wolfTPM2_NewKeyBlob(
    void
)
```

Allocate and initialize a WOLFTPM2_KEYBLOB.

See: wolfTPM2_FreeKeyBlob

Return:

- pointer to newly initialized WOLFTPM2_KEYBLOB
- NULL on any error

```
WOLFTPM_API int wolfTPM2_FreeKeyBlob(
    WOLFTPM2_KEYBLOB * blob
)
```

Free a WOLFTPM2_KEYBLOB that was allocated with wolfTPM2_NewKeyBlob.

Parameters:

- **blob** pointer to a WOLFTPM2_KEYBLOB that was allocated by wolfTPM2_NewKeyBlob

See: wolfTPM2_NewKeyBlob

Return: TPM_RC_SUCCESS: successful

```
WOLFTPM_API TPMT_PUBLIC * wolfTPM2_NewPublicTemplate(
    void
)
```

Allocate and initialize a TPMT_PUBLIC.

See: wolfTPM2_FreePublicTemplate

Return:

- pointer to newly initialized
- NULL on any error

```
WOLFTPM_API int wolfTPM2_FreePublicTemplate(
    TPMT_PUBLIC * PublicTemplate
)
```

Free a TPMT_PUBLIC that was allocated with wolfTPM2_NewPublicTemplate.

Parameters:

- **PublicTemplate** pointer to a TPMT_PUBLIC that was allocated with wolfTPM2_NewPublicTemplate

See: wolfTPM2_NewPublicTemplate

Return: TPM_RC_SUCCESS: successful

```
WOLFTPM_API WOLFTPM2_KEY * wolfTPM2_NewKey(
    void
)
```

Allocate and initialize a WOLFTPM2_KEY.

See: wolfTPM2_FreeKey

Return:

- pointer to newly initialized WOLFTPM2_KEY
- NULL on any error

```
WOLFTPM_API int wolfTPM2_FreeKey(
    WOLFTPM2_KEY * key
)
```

Free a WOLFTPM2_KEY that was allocated with wolfTPM2_NewKey.

Parameters:

- **key** pointer to a WOLFTPM2_KEY that was allocated by wolfTPM2_NewKey

See: wolfTPM2_NewKey

Return: TPM_RC_SUCCESS: successful

```
WOLFTPM_API WOLFTPM2_SESSION * wolfTPM2_NewSession(
    void
)
```

Allocate and initialize a WOLFTPM2_SESSION.

See: wolfTPM2_FreeSession

Return:

- pointer to newly initialized WOLFTPM2_SESSION
- NULL on any error

```
WOLFTPM_API int wolfTPM2_FreeSession(
    WOLFTPM2_SESSION * session
)
```

Free a WOLFTPM2_SESSION that was allocated with wolfTPM2_NewSession.

Parameters:

- **session** pointer to a WOLFTPM2_SESSION struct

See: wolfTPM2_NewSession

Return: TPM_RC_SUCCESS: successful

```
WOLFTPM_API WOLFTPM2_CSR * wolfTPM2_NewCSR(
    void
)
```

Allocate and initialize a WOLFTPM2_CSR.

See: wolfTPM2_FreeCSR

Return:

- pointer to newly initialized WOLFTPM2_CSR
- NULL on any error

```
WOLFTPM_API int wolfTPM2_FreeCSR(
    WOLFTPM2_CSR * csr
)
```

Free a WOLFTPM2_CSR that was allocated with wolfTPM2_NewCSR.

Parameters:

- **csr** pointer to a WOLFTPM2_CSR that was allocated by wolfTPM2_NewCSR

See: wolfTPM2_NewCSR

Return: TPM_RC_SUCCESS: successful

```
WOLFTPM_API WOLFTPM2_HANDLE * wolfTPM2_GetHandleRefFromKey(
    WOLFTPM2_KEY * key
)
```

Retrieve the WOLFTPM2_HANDLE from a WOLFTPM2_KEY.

Parameters:

- **key** pointer to a WOLFTPM2_KEY struct

Return:

- pointer to handle in the key structure
- NULL if key pointer is NULL

```
WOLFTPM_API WOLFTPM2_HANDLE * wolfTPM2_GetHandleRefFromKeyBlob(
    WOLFTPM2_KEYBLOB * keyBlob
)
```

Retrieve the WOLFTPM2_HANDLE from a WOLFTPM2_KEYBLOB.

Parameters:

- **keyBlob** pointer to a WOLFTPM2_KEYBLOB struct

Return:

- pointer to handle in the key blob structure
- NULL if key pointer is NULL

```
WOLFTPM_API WOLFTPM2_HANDLE * wolfTPM2_GetHandleRefFromSession(
    WOLFTPM2_SESSION * session
)
```

Retrieve the WOLFTPM2_HANDLE from a WOLFTPM2_SESSION.

Parameters:

- **session** pointer to a WOLFTPM2_SESSION struct

Return:

- pointer to handle in the session structure
- NULL if key pointer is NULL

```
WOLFTPM_API TPM_HANDLE wolfTPM2_GetHandleValue(
    WOLFTPM2_HANDLE * handle
)
```

Get the 32-bit handle value from the WOLFTPM2_HANDLE.

Parameters:

- **handle** pointer to WOLFTPM2_HANDLE structure

Return: TPM_HANDLE value from TPM

```
WOLFTPM_API int wolfTPM2_SetKeyAuthPassword(
    WOLFTPM2_KEY * key,
    const byte * auth,
    int authSz
)
```

Set the authentication data for a key.

Parameters:

- **key** pointer to wrapper key struct
- **auth** pointer to auth data
- **authSz** length in bytes of auth data

Return:

- TPM_RC_SUCCESS: successful
- BAD_FUNC_ARG: check the provided arguments

```
WOLFTPM_API int wolfTPM2_GetKeyBlobAsBuffer(
    byte * buffer,
    word32 bufferSz,
    WOLFTPM2_KEYBLOB * key
)
```

Marshal data from a keyblob to a binary buffer. This can be stored to disk for loading in a separate process or after power cycling. If buffer is not provided then size only will be returned.

Parameters:

- **buffer** pointer to buffer in which to store marshaled keyblob
- **bufferSz** size of the above buffer
- **key** pointer to keyblob to marshal

See: wolfTPM2_SetKeyBlobFromBuffer

Return:

- Positive integer (size of the output)
- BUFFER_E: insufficient space in provided buffer
- BAD_FUNC_ARG: check the provided arguments

```
WOLFTPM_API int wolfTPM2_GetKeyBlobAsSeparateBuffers(
    byte * pubBuffer,
    word32 * pubBufferSz,
    byte * privBuffer,
    word32 * privBufferSz,
    WOLFTPM2_KEYBLOB * key
)
```

Marshal data from a keyblob to a binary buffer. This can be stored to disk for loading in a separate process or after power cycling. If either buffer is NULL then the size will be returned for each part.

Parameters:

- **pubBuffer** pointer to buffer in which to store the public part of the marshaled keyblob
- **pubBufferSz** pointer to the size of the above buffer
- **privBuffer** pointer to buffer in which to store the private part of the marshaled keyblob
- **privBufferSz** pointer to the size of the above buffer
- **key** pointer to keyblob to marshal

See: wolfTPM2_GetKeyBlobAsSeparateBuffers

Return:

- TPM_RC_SUCCESS: successful
- BUFFER_E: insufficient space in provided buffer
- BAD_FUNC_ARG: check the provided arguments
- LENGTH_ONLY_E: Returning length only (when either of the buffers is NULL)

```
WOLFTPM_API int wolfTPM2_SetKeyBlobFromBuffer(
    WOLFTPM2_KEYBLOB * key,
    byte * buffer,
    word32 bufferSz
)
```

Unmarshal data into a WOLFTPM2_KEYBLOB struct. This can be used to load a keyblob that was previously marshaled by wolfTPM2_GetKeyBlobAsBuffer.

Parameters:

- **key** pointer to keyblob to load and unmarshall data into
- **buffer** pointer to buffer containing marshalled keyblob to load from
- **bufferSz** size of the above buffer

See: wolfTPM2_GetKeyBlobAsBuffer

Return:

- TPM_RC_SUCCESS: successful
- BUFFER_E: buffer is too small or there is extra data remaining and not unmarshalled
- BAD_FUNC_ARG: check the provided arguments

```
WOLFTPM_API int wolfTPM2_PolicyRestart(
    WOLFTPM2_DEV * dev,
    TPM_HANDLE sessionHandle
)
```

Restart the policy digest for a policy session.

Parameters:

- **dev** pointer to a TPM2_DEV struct
- **sessionHandle** the handle of the current session, a session is required to use policy pcr

See:

- wolfTPM2_GetPolicyDigest
- wolfTPM2_PolicyPCR
- wolfTPM2_PolicyAuthorize

Return:

- TPM_RC_SUCCESS: successful
- INPUT_SIZE_E: policyDigestSz is too small to hold the returned digest
- BAD_FUNC_ARG: check the provided arguments

```
WOLFTPM_API int wolfTPM2_GetPolicyDigest(
    WOLFTPM2_DEV * dev,
    TPM_HANDLE sessionHandle,
    byte * policyDigest,
    word32 * policyDigestSz
)
```

Get the policy digest of the session that was passed in wolfTPM2_GetPolicyDigest.

Parameters:

- **dev** pointer to a TPM2_DEV struct
- **sessionHandle** the handle of the current session, a session is required to use policy pcr
- **policyDigest** output digest of the policy
- **policyDigestSz** pointer to the size of the policyDigest

See:

- wolfTPM2_PolicyPCR
- wolfTPM2_PolicyAuthorize
- wolfTPM2_PolicyRestart

Return:

- TPM_RC_SUCCESS: successful
- INPUT_SIZE_E: policyDigestSz is too small to hold the returned digest
- BAD_FUNC_ARG: check the provided arguments

```
WOLFTPM_API int wolfTPM2_PolicyPCR(
    WOLFTPM2_DEV * dev,
    TPM_HANDLE sessionHandle,
    TPM_ALG_ID pcrAlg,
    byte * pcrArray,
    word32 pcrArraySz
)
```

Apply the PCR's to the policy digest for the policy session.

Parameters:

- **dev** pointer to a TPM2_DEV struct
- **sessionHandle** the handle of the current policy session, a session is required to use policy PCR
- **pcrAlg** the hash algorithm to use with PCR policy
- **pcrArray** array of PCR Indexes to use when creating the policy
- **pcrArraySz** the number of PCR Indexes in the pcrArray

See:

- wolfTPM2_GetPolicyDigest
- wolfTPM2_PolicyPCR
- wolfTPM2_PolicyAuthorize
- wolfTPM2_PolicyRestart

Return:

- TPM_RC_SUCCESS: successful
- INPUT_SIZE_E: policyDigestSz is too small to hold the returned digest
- BAD_FUNC_ARG: check the provided arguments

```
WOLFTPM_API int wolfTPM2_PolicyTransportSPDM(
    WOLFTPM2_DEV * dev,
    TPM_HANDLE sessionHandle,
    const TPM2B_NAME * reqKeyName,
    const TPM2B_NAME * tpmKeyName
)
```

Add TPM2_PolicyTransportSPDM (TPM 2.0 Library v1.84) to a policy session. An entity whose authPolicy contains this term can only be authorized by a command that arrives inside an SPDM session. Optional key names further restrict which SPDM session qualifies.

Parameters:

- **dev** pointer to a TPM2_DEV struct
- **sessionHandle** the handle of the current policy session
- **reqKeyName** requester SPDM key name to bind, or NULL for any
- **tpmKeyName** TPM SPDM key name to bind, or NULL for any

See:

- wolfTPM2_PolicyTransportSPDMMake
- wolfTPM2_GetCapability_SPDMSessionInfo
- wolfTPM2_GetPolicyDigest

Return:

- TPM_RC_SUCCESS: successful
- TPM_RC_VALUE: PolicyTransportSPDM already applied to this session
- TPM_RC_HASH: a name uses an unsupported hash algorithm
- TPM_RC_SIZE: a name is not the size its hash algorithm requires
- BUFFER_E: a name is larger than TPM2B_NAME can hold
- BAD_FUNC_ARG: check the provided arguments

```
WOLFTPM_API int wolfTPM2_PolicyAuthorize(
    WOLFTPM2_DEV * dev,
    TPM_HANDLE sessionHandle,
    const TPM2B_PUBLIC * pub,
    const TPMT_TK_VERIFIED * checkTicket,
    const byte * pcrDigest,
    word32 pcrDigestSz,
    const byte * policyRef,
    word32 policyRefSz
)
```

Apply the PCR's to the policy digest for the policy session.

Parameters:

- **dev** pointer to a TPM2_DEV struct
- **sessionHandle** the handle of the current policy session, a session is required to use policy PCR
- **pub** pointer to a populated structure of TPM2B_PUBLIC type
- **checkTicket** returns the validation ticket proving the signature for digest was checked
- **pcrDigest** digest for the PCR(s) collected with wolfTPM2_PCRGetDigest
- **pcrDigestSz** size of the PCR digest
- **policyRef** optional nonce
- **policyRefSz** optional nonce size

See:

- wolfTPM2_GetPolicyDigest
- wolfTPM2_PolicyPCR
- wolfTPM2_PolicyAuthorize
- wolfTPM2_PolicyRestart
- wolfTPM2_PCRGetDigest

Return:

- TPM_RC_SUCCESS: successful
- INPUT_SIZE_E: policyDigestSz is too small to hold the returned digest
- BAD_FUNC_ARG: check the provided arguments

```
WOLFTPM_API int wolfTPM2_PCRGetDigest(
    WOLFTPM2_DEV * dev,
    TPM_ALG_ID pcrAlg,
    byte * pcrArray,
    word32 pcrArraySz,
    byte * pcrDigest,
    word32 * pcrDigestSz
)
```

Get a cumulative digest of the PCR's specified.

Parameters:

- **dev** pointer to a TPM2_DEV struct
- **pcrAlg** the hash algorithm to use with pcr policy
- **pcrArray** array of pcr Index to use when creating the policy
- **pcrArraySz** the number of Index in the pcrArray
- **pcrDigest** digest for the PCR(s) collected with wolfTPM2_PCRGetDigest
- **pcrDigestSz** size of the PCR digest

See:

- wolfTPM2_PolicyPCR
- wolfTPM2_PolicyAuthorize

Return:

- TPM_RC_SUCCESS: successful
- INPUT_SIZE_E: policyDigestSz is too small to hold the returned digest
- BAD_FUNC_ARG: check the provided arguments

```
WOLFTPM_API int wolfTPM2_PolicyRefMake(
    TPM_ALG_ID pcrAlg,
    byte * digest,
    word32 * digestSz,
    const byte * policyRef,
    word32 policyRefSz
)
```

Utility for generating a policy ref digest. If no policy reference (nonce) used then just rehash the provided digest again (update -> final)

Parameters:

- **pcrAlg** the hash algorithm to use with pcr policy
- **digest** input/out digest
- **digestSz** input/out digest size
- **policyRef** optional nonce
- **policyRefSz** optional nonce size

See:

- wolfTPM2_PolicyPCRMake
- wolfTPM2_PolicyAuthorizeMake

Return:

- TPM_RC_SUCCESS: successful
- INPUT_SIZE_E: policyDigestSz is too small to hold the returned digest
- BAD_FUNC_ARG: check the provided arguments

```
WOLFTPM_API int wolfTPM2_PolicyPCRMake(
    TPM_ALG_ID pcrAlg,
    byte * pcrArray,
    word32 pcrArraySz,
    const byte * pcrDigest,
    word32 pcrDigestSz,
    byte * digest,
    word32 * digestSz
)
```

Utility for generating a policy PCR digest.

Parameters:

- **pcrAlg** the supported hash algorithm to use with the PCR policy
- **pcrArray** non-NULL array of PCRs to include in the policy
- **pcrArraySz** number of entries in pcrArray; must be greater than zero
- **pcrDigest** digest for the PCR(s) collected (can get using wolfTPM2_PCRGetDigest); required when pcrDigestSz is nonzero
- **pcrDigestSz** size of the PCR digest

- **digest** input/output policy digest buffer
- **digestSz** input/output: current digest size and buffer capacity on input, which must be at least the selected hash size; selected hash size on output

See:

- wolfTPM2_PolicyPCRMake
- wolfTPM2_PolicyAuthorizeMake
- wolfTPM2_PCRGetDigest

Return:

- TPM_RC_SUCCESS: successful
- BUFFER_E: digest is too small for the selected hash or the PCR policy input exceeds the internal assembly buffer
- BAD_FUNC_ARG: check the provided arguments

Note: To start a fresh policy chain, zero the digest buffer and set digestSz to the selected hash size before calling.

```
WOLFTPM_API int wolfTPM2_PolicyTransportSPDMMake(
    TPM_ALG_ID hashAlg,
    const TPM2B_NAME * reqKeyName,
    const TPM2B_NAME * tpmKeyName,
    byte * digest,
    word32 * digestSz
)
```

Compute the policy digest that TPM2_PolicyTransportSPDM produces, without a TPM. Use it to build the authPolicy of an NV index or key that must only be reachable over an SPDM session.

Parameters:

- **hashAlg** the policy session hash algorithm
- **reqKeyName** requester SPDM key name to bind, or NULL for any
- **tpmKeyName** TPM SPDM key name to bind, or NULL for any
- **digest** input/output: policyDigestOld in (zeros for a fresh session), policyDigestNew out
- **digestSz** input/output: current digest size and buffer capacity on input, selected hash size on output

See:

- wolfTPM2_PolicyTransportSPDM
- wolfTPM2_PolicyHash
- wolfTPM2_NVCreateAuthPolicy

Return:

- TPM_RC_SUCCESS: successful
- BUFFER_E: digestSz is too small for the selected hash

- BAD_FUNC_ARG: check the provided arguments

```
WOLFTPM_API int wolfTPM2_PolicyCommandCodeMake(
    TPM_ALG_ID hashAlg,
    byte * digest,
    word32 * digestSz,
    TPM_CC cc
)
```

Compute the policy digest for PolicyCommandCode on a fresh session.

Parameters:

- **hashAlg** hash algorithm for the policy digest
- **digest** output policy digest buffer
- **digestSz** in/out: input buffer capacity, output digest size
- **cc** the command code to bind (for example TPM_CC_NV_Read)

See:

- wolfTPM2_PolicyCommandCode
- wolfTPM2_PolicyHash
- wolfTPM2_PolicyPCRMake

Return:

- TPM_RC_SUCCESS: successful
- BAD_FUNC_ARG: NULL digest/digestSz or unsupported hashAlg
- BUFFER_E: *digestSz (the buffer capacity) is smaller than the hash size for hashAlg

Note:

- policyDigest = hash(zeroDigest || TPM_CC_PolicyCommandCode || cc). Mirrors the running digest of a new policy session after wolfTPM2_PolicyCommandCode.
- digestSz is in/out. On input it is the capacity of the digest buffer in bytes, which must be at least the hash size or BUFFER_E is returned without writing to digest. On output it is the hash size written.

```
WOLFTPM_API int wolfTPM2_PolicyHash(
    TPM_ALG_ID hashAlg,
    byte * digest,
    word32 * digestSz,
    TPM_CC cc,
    const byte * input,
    word32 inputSz
)
```

Utility for creating a policy hash. Generic helper that takes command code and input array. policyDigestnew = hash(policyDigestOld || [cc] || [Input])

Parameters:

- **hashAlg** the hash algorithm to use with pcr policy
- **digest** input/out digest (input “old” / output “new”)
- **digestSz** input/out digest size
- **cc** is the command code used
- **input** pointer to a array to use (optional)
- **inputSz** size of input

See: wolfTPM2_PolicyPCRMake

Return:

- TPM_RC_SUCCESS: successful
- INPUT_SIZE_E: policyDigestSz is too small to hold the returned digest
- BAD_FUNC_ARG: check the provided arguments

```
WOLFTPM_API int wolfTPM2_PolicyAuthorizeMake(
    TPM_ALG_ID pcrAlg,
    const TPM2B_PUBLIC * pub,
    byte * digest,
    word32 * digestSz,
    const byte * policyRef,
    word32 policyRefSz
)
```

Utility for generating a policy authorization digest based on a public key.

Parameters:

- **pcrAlg** the hash algorithm to use with pcr policy
- **pub** pointer to a populated structure of TPM2B_PUBLIC type
- **digest** input/out digest
- **digestSz** input/out digest size
- **policyRef** optional nonce
- **policyRefSz** optional nonce size

See:

- wolfTPM2_PolicyPCRMake
- wolfTPM2_PolicyHash

Return:

- TPM_RC_SUCCESS: successful
- INPUT_SIZE_E: policyDigestSz is too small to hold the returned digest
- BAD_FUNC_ARG: check the provided arguments

```
WOLFTPM_API int wolfTPM2_PolicyPassword(
    WOLFTPM2_DEV * dev,
    WOLFTPM2_SESSION * tpmSession,
    const byte * auth,
    int authSz
)
```

Wrapper for setting a policy password and calling TPM2_PolicyPassword. This will set a password (in clear) for the policy session instead of HMAC.

Parameters:

- **dev** pointer to a TPM2_DEV struct
- **tpmSession** pointer to a WOLFTPM2_SESSION struct used with wolfTPM2_StartSession and wolfTPM2_SetAuthSession
- **auth** pointer to a string constant, specifying the password authorization for the policy session
- **authSz** integer value, specifying the size of the password authorization, in bytes

See:

- wolfTPM2_PolicyAuthValue
- wolfTPM2_PolicyCommandCode

Return:

- TPM_RC_SUCCESS: successful
- BAD_FUNC_ARG: check the provided arguments

```
WOLFTPM_API int wolfTPM2_PolicyAuthValue(
    WOLFTPM2_DEV * dev,
    WOLFTPM2_SESSION * tpmSession,
    const byte * auth,
    int authSz
)
```

Wrapper for setting a policy auth value that is added to the HMAC key for a policy session.

Parameters:

- **dev** pointer to a TPM2_DEV struct
- **tpmSession** pointer to a WOLFTPM2_SESSION struct used with wolfTPM2_StartSession and wolfTPM2_SetAuthSession
- **auth** pointer to a string constant, specifying the password authorization for the policy session
- **authSz** integer value, specifying the size of the password authorization, in bytes

See:

- wolfTPM2_PolicyPassword
- wolfTPM2_PolicyCommandCode

Return:

- TPM_RC_SUCCESS: successful
- BAD_FUNC_ARG: check the provided arguments

```
WOLFTPM_API int wolfTPM2_PolicyCommandCode(
    WOLFTPM2_DEV * dev,
    WOLFTPM2_SESSION * tpmSession,
    TPM_CC cc
)
```

Wrapper for setting a policy command code.

Parameters:

- **dev** pointer to a TPM2_DEV struct
- **tpmSession** pointer to a WOLFTPM2_SESSION struct used with wolfTPM2_StartSession and wolfTPM2_SetAuthSession
- **cc** TPM_CC command code

See:

- wolfTPM2_PolicyPassword
- wolfTPM2_PolicyAuthValue

Return:

- TPM_RC_SUCCESS: successful
- BAD_FUNC_ARG: check the provided arguments

```
WOLFTPM_API int wolfTPM2_PolicyOR(
    WOLFTPM2_DEV * dev,
    WOLFTPM2_SESSION * tpmSession,
    const TPML_DIGEST * pHashList
)
```

Wrapper for satisfying a policy session with a compound OR of digests.

Parameters:

- **dev** pointer to a TPM2_DEV struct
- **tpmSession** pointer to a WOLFTPM2_SESSION struct used with wolfTPM2_StartSession and wolfTPM2_SetAuthSession
- **pHashList** list of pre-computed policy branch digests to OR together

See:

- wolfTPM2_PolicyPCR
- wolfTPM2_PolicyAuthorize
- wolfTPM2_GetPolicyDigest

Return:

- TPM_RC_SUCCESS: successful
- BAD_FUNC_ARG: bad pointer, count out of range (< 2 or > capacity), or a branch size that exceeds the digest buffer

Note:

- The digest list is hash-agnostic (each branch carries its own size), so it supports SHA2-256 through SHA2-512 policy branches. The number of branches (pHashList->count) must be between 2 and the TPML_DIGEST capacity (the digests[] array length), and each branch's size must not exceed the digest buffer length; branches beyond count are ignored.
- A minimum of two branches is required by TPM 2.0 Part 3 Sec.23.6. A one-branch list is rejected here with BAD_FUNC_ARG rather than sent to the TPM, which would answer TPM_RC_VALUE.

```
WOLFTPM_API int wolfTPM2_SetPrimaryPolicy(
    WOLFTPM2_DEV * dev,
    TPMI_RH_HIERARCHY_AUTH authHandle,
    TPM_ALG_ID hashAlg,
    const byte * authPolicy,
    word32 authPolicySz
)
```

Set (or clear) the authPolicy of a hierarchy.

Parameters:

- **dev** pointer to a TPM2_DEV struct
- **authHandle** the hierarchy (for example TPM_RH_PLATFORM)
- **hashAlg** the policy digest hash algorithm (TPM_ALG_NULL to clear)
- **authPolicy** the policy digest to set (NULL to clear)
- **authPolicySz** size of the policy digest (0 to clear)

See:

- wolfTPM2_PolicyOR
- wolfTPM2_GetPolicyDigest

Return:

- TPM_RC_SUCCESS: successful
- BAD_FUNC_ARG: NULL dev, authPolicySz exceeds the digest buffer, or authPolicy is NULL with a non-zero authPolicySz

Note: Wraps TPM2_SetPrimaryPolicy for owner/endorsement/platform/lockout. Pass authPolicy=NULL, authPolicySz=0 and hashAlg=TPM_ALG_NULL to clear an existing policy. The command itself is authorized by the hierarchy's current auth (set it on the device's active session beforehand).

```
WOLFTPM_API int wolfTPM2_SetIdentityAuth(
    WOLFTPM2_DEV * dev,
    WOLFTPM2_HANDLE * handle,
    uint8_t * masterPassword,
    uint16_t masterPasswordSz
)
```

Set authentication for pre-provisioned identity keys.

Parameters:

- **dev** pointer to a TPM2_DEV struct
- **handle** pointer to WOLFTPM2_HANDLE for the identity key
- **masterPassword** pointer to master password data (NULL uses the ST33 sample password only on ST33-capable builds)
- **masterPasswordSz** size of master password in bytes

See: wolfTPM2_CreateAndLoadAIK

Return:

- TPM_RC_SUCCESS: successful
- TPM_RC_FAILURE: generic failure (check TPM IO and TPM return code)
- BAD_FUNC_ARG: NULL handle, or NULL/empty masterPassword on a non-ST33 build

Note:

- Used with IAK and IDevID keys on ST33KTPM devices
- On ST33 (or autodetect) builds a NULL masterPassword derives auth from the public sample password; other targets require an explicit masterPassword and return BAD_FUNC_ARG when it is NULL/empty

```
WOLFTPM_LOCAL int GetKeyTemplateRSA(
    TPMT_PUBLIC * publicTemplate,
    TPM_ALG_ID nameAlg,
    TPMA_OBJECT objectAttributes,
    int keyBits,
    long exponent,
    TPM_ALG_ID sigScheme,
    TPM_ALG_ID sigHash
)
```

Internal helper to create RSA key template.

Parameters:

- **publicTemplate** pointer to TPMT_PUBLIC template to populate
- **nameAlg** hash algorithm for key name
- **objectAttributes** TPM object attributes
- **keyBits** RSA key size in bits
- **exponent** RSA public exponent
- **sigScheme** signature scheme algorithm
- **sigHash** hash algorithm for signatures

See: GetKeyTemplateECC

Return:

- TPM_RC_SUCCESS: successful
- BAD_FUNC_ARG: check the provided arguments

Note: Used internally by key creation functions

```
WOLFTPM_LOCAL int GetKeyTemplateECC(
    TPMT_PUBLIC * publicTemplate,
    TPM_ALG_ID nameAlg,
    TPMA_OBJECT objectAttributes,
    TPM_ECC_CURVE curve,
    TPM_ALG_ID sigScheme,
    TPM_ALG_ID sigHash
)
```

Internal helper to create ECC key template.

Parameters:

- **publicTemplate** pointer to TPMT_PUBLIC template to populate
- **nameAlg** hash algorithm for key name
- **objectAttributes** TPM object attributes
- **curve** ECC curve identifier
- **sigScheme** signature scheme algorithm
- **sigHash** hash algorithm for signatures

See: GetKeyTemplateRSA

Return:

- TPM_RC_SUCCESS: successful
- BAD_FUNC_ARG: check the provided arguments

Note: Used internally by key creation functions

```
WOLFTPM_API int wolfTPM2_FirmwareUpgradeHash(
    WOLFTPM2_DEV * dev,
    TPM_ALG_ID hashAlg,
    uint8_t * manifest_hash,
    uint32_t manifest_hash_sz,
```

```
    uint8_t * manifest,
    uint32_t manifest_sz,
    wolfTPM2FwDataCb cb,
    void * cb_ctx
)
```

Perform TPM firmware upgrade with pre-computed manifest hash.

Parameters:

- **dev** pointer to a TPM2_DEV struct
- **hashAlg** hash algorithm to use (TPM_ALG_SHA384 or TPM_ALG_SHA512)
- **manifest_hash** buffer to store computed manifest hash
- **manifest_hash_sz** size of manifest hash buffer
- **manifest** pointer to firmware manifest data
- **manifest_sz** size of firmware manifest
- **cb** callback function for firmware data access
- **cb_ctx** context pointer passed to callback

See:

- wolfTPM2_FirmwareUpgrade
- wolfTPM2_FirmwareUpgradeRecover

Return:

- TPM_RC_SUCCESS: successful
- TPM_RC_FAILURE: generic failure (check TPM IO and TPM return code)
- BAD_FUNC_ARG: check the provided arguments

Note:

- Supports SHA2-384 or SHA2-512 for manifest hash
- For ST33KTPM: LMS vs non-LMS format is auto-detected from manifest size:
- 177 bytes: Non-LMS format (firmware < 512, e.g., 9.257)
- 2697 bytes: LMS format (firmware >= 512, e.g., 9.512)

```
WOLFTPM_API int wolfTPM2_FirmwareUpgradeHash_ex(
    WOLFTPM2_DEV * dev,
    TPM_ALG_ID hashAlg,
    uint8_t * manifest_hash,
    uint32_t manifest_hash_sz,
    uint8_t * manifest,
    uint32_t manifest_sz,
    wolfTPM2FwDataCb cb,
    void * cb_ctx,
    WOLFTPM2_SESSION * startSession
)
```

Perform TPM firmware upgrade using a caller-supplied authorization session.

Parameters:

- **dev** pointer to a TPM2_DEV struct
- **hashAlg** hash algorithm to use (TPM_ALG_SHA384 or TPM_ALG_SHA512)
- **manifest_hash** buffer to store computed manifest hash
- **manifest_hash_sz** size of manifest hash buffer
- **manifest** pointer to firmware manifest data
- **manifest_sz** size of firmware manifest
- **cb** callback function for firmware data access
- **cb_ctx** context pointer passed to callback
- **startSession** optional caller-satisfied session authorizing the firmware-start command (NULL for library-managed authorization)

See:

- wolfTPM2_FirmwareUpgradeHash
- wolfTPM2_PolicyOR
- wolfTPM2_StartSession_ex

Return:

- TPM_RC_SUCCESS: successful
- TPM_RC_FAILURE: generic failure (check TPM IO and TPM return code)
- BAD_FUNC_ARG: check the provided arguments, or startSession is not an unsalted/unbound policy session with no auth value

Note:

- Identical to wolfTPM2_FirmwareUpgradeHash except the caller controls how the firmware-start command is authorized against the platform hierarchy.
- When startSession is NULL this behaves exactly like wolfTPM2_FirmwareUpgradeHash (library-managed platform authorization).
- When startSession is non-NULL the caller is responsible for having satisfied the platform authPolicy on that session (for example via wolfTPM2_PolicyPCR / wolfTPM2_PolicyAuthorize / wolfTPM2_PolicyOR using SHA2-256 or SHA2-512). For Infineon the platform primary policy is left untouched (the caller provisions it); for ST33 the session replaces the default TPM_RS_PW password authorization.
- The vendor firmware-start command carries a session handle with an empty nonceCaller, zero session attributes and an empty HMAC, so startSession must be an unsalted, unbound TPM_SE_POLICY session with no auth value and no parameter encryption. Sessions satisfied with wolfTPM2_PolicyAuthValue or wolfTPM2_PolicyPassword are NOT supported, because the required session HMAC is not serialized. Any such session is rejected with BAD_FUNC_ARG before the command is sent.
- On a successful firmware start the TPM consumes the session, and this function sets startSession->handle.hndl to TPM_RH_NULL to record that. The caller must not flush it; wolfTPM2_UnloadHandle is a no-op on TPM_RH_NULL, so an unconditional cleanup call remains safe.

```
WOLFTPM_API int wolfTPM2_FirmwareUpgrade(
    WOLFTPM2_DEV * dev,
    uint8_t * manifest,
    uint32_t manifest_sz,
    wolfTPM2FwDataCb cb,
    void * cb_ctx
)
```

Perform TPM firmware upgrade.

Parameters:

- **dev** pointer to a TPM2_DEV struct
- **manifest** pointer to firmware manifest data
- **manifest_sz** size of firmware manifest
- **cb** callback function for firmware data access
- **cb_ctx** context pointer passed to callback

See:

- wolfTPM2_FirmwareUpgradeHash
- wolfTPM2_FirmwareUpgradeRecover

Return:

- TPM_RC_SUCCESS: successful
- TPM_RC_FAILURE: generic failure (check TPM IO and TPM return code)
- BAD_FUNC_ARG: check the provided arguments
- NOT_COMPILED_IN: wolfSSL not built with WOLFSSL_SHA384

Note:

- Upgrades TPM firmware using provided manifest and data callback
- Requires WOLFSSL_SHA384; returns NOT_COMPILED_IN otherwise
- For ST33KTPM: LMS vs non-LMS format is auto-detected from manifest size:
- 177 bytes: Non-LMS format (firmware < 512, e.g., 9.257)
- 2697 bytes: LMS format (firmware >= 512, e.g., 9.512)

```
WOLFTPM_API int wolfTPM2_FirmwareUpgrade_ex(
    WOLFTPM2_DEV * dev,
    uint8_t * manifest,
    uint32_t manifest_sz,
    wolfTPM2FwDataCb cb,
    void * cb_ctx,
    WOLFTPM2_SESSION * startSession
)
```

Perform TPM firmware upgrade using a caller-supplied authorization session.

Parameters:

- **dev** pointer to a TPM2_DEV struct
- **manifest** pointer to firmware manifest data
- **manifest_sz** size of firmware manifest
- **cb** callback function for firmware data access
- **cb_ctx** context pointer passed to callback
- **startSession** optional caller-satisfied session (NULL for default). See wolfTPM2_FirmwareUpgradeHash_ex for the supported session contract: unsalted, unbound policy session with no auth value.

See:

- wolfTPM2_FirmwareUpgrade
- wolfTPM2_FirmwareUpgradeHash_ex

Return:

- TPM_RC_SUCCESS: successful
- NOT_COMPILED_IN: wolfSSL not built with WOLFSSL_SHA384

Note: Same as wolfTPM2_FirmwareUpgrade but the caller controls how the firmware-start command is authorized (see wolfTPM2_FirmwareUpgradeHash_ex). startSession NULL preserves the default library-managed behavior.

```
WOLFTPM_API int wolfTPM2_FirmwareUpgradeRecover(
    WOLFTPM2_DEV * dev,
    uint8_t * manifest,
    uint32_t manifest_sz,
    wolfTPM2FwDataCb cb,
    void * cb_ctx
)
```

Recover from failed TPM firmware upgrade.

Parameters:

- **dev** pointer to a TPM2_DEV struct
- **manifest** pointer to firmware manifest data
- **manifest_sz** size of firmware manifest
- **cb** callback function for firmware data access
- **cb_ctx** context pointer passed to callback

See:

- wolfTPM2_FirmwareUpgrade
- wolfTPM2_FirmwareUpgradeHash

Return:

- TPM_RC_SUCCESS: successful
- TPM_RC_FAILURE: generic failure (check TPM IO and TPM return code)
- BAD_FUNC_ARG: check the provided arguments

Note: Attempts to recover TPM after interrupted/failed upgrade

```
WOLFTPM_API int wolfTPM2_FirmwareUpgradeRecover_ex(
    WOLFTPM2_DEV * dev,
    uint8_t * manifest,
    uint32_t manifest_sz,
    wolfTPM2FwDataCb cb,
    void * cb_ctx,
    WOLFTPM2_SESSION * startSession
)
```

Recover from a failed firmware upgrade using a caller-supplied session.

Parameters:

- **dev** pointer to a TPM2_DEV struct
- **manifest** pointer to firmware manifest data
- **manifest_sz** size of firmware manifest
- **cb** callback function for firmware data access
- **cb_ctx** context pointer passed to callback
- **startSession** optional caller-satisfied session (NULL for default). See wolfTPM2_FirmwareUpgradeHash_ex for the supported session contract: unsalted, unbound policy session with no auth value.

See:

- wolfTPM2_FirmwareUpgradeRecover
- wolfTPM2_FirmwareUpgradeHash_ex

Return:

- TPM_RC_SUCCESS: successful
- BAD_FUNC_ARG: check the provided arguments

Note: Same as wolfTPM2_FirmwareUpgradeRecover but with caller-controlled authorization (see wolfTPM2_FirmwareUpgradeHash_ex). startSession NULL preserves the default library-managed behavior.

```
WOLFTPM_API int wolfTPM2_FirmwareUpgradeCancel(
    WOLFTPM2_DEV * dev
)
```

Cancel ongoing TPM firmware upgrade.

Parameters:

- **dev** pointer to a TPM2_DEV struct

See:

- wolfTPM2_FirmwareUpgrade
- wolfTPM2_FirmwareUpgradeRecover

Return:

- TPM_RC_SUCCESS: successful
- TPM_RC_FAILURE: generic failure (check TPM IO and TPM return code)
- BAD_FUNC_ARG: check the provided arguments

Note: Aborts current firmware upgrade process

```
WOLFTPM_API int wolfTPM2_ST33_ManifestVersion(
    const uint8_t * manifest,
    uint32_t manifest_sz,
    word16 * major,
    word16 * minor
)
```

Read the firmware version an ST33 firmware image upgrades to.

Parameters:

- **manifest** pointer to the manifest (blob0) bytes
- **manifest_sz** size of the manifest in bytes
- **major** pointer to store the target major version (optional, may be NULL)
- **minor** pointer to store the target minor version (optional, may be NULL)

See:

- wolfTPM2_ST33_FwUpgradeCommands
- wolfTPM2_FirmwareUpgrade

Return:

- TPM_RC_SUCCESS: successful
- BAD_FUNC_ARG: manifest is NULL or too small to hold the version

Note: The manifest (blob0) header opens with a zero byte followed by the target version in the TPM_PT_FIRMWARE_VERSION_1 layout: UINT16 major then UINT16 minor, big endian (00 | 00 02 02 00 is 2.512)

```
WOLFTPM_API int wolfTPM2_ST33_CmdImplemented(
    TPM_CC cc,
    int * isImpl
)
```

Report whether the TPM implements a command code.

Parameters:

- **cc** command code to look for
- **isImpl** pointer to store 1 when implemented, 0 when not

See: wolfTPM2_ST33_GetFwUpgradeCommands

Return:

- TPM_RC_SUCCESS: query succeeded, isImpl set to 1 or 0
- BAD_FUNC_ARG: isImpl is NULL
- TPM_RC_VALUE: the TPM answered with a different capability

Note: Queries TPM_CAP_COMMANDS, which returns commands at or above the requested code, so a match at index 0 of a single-property query means it is implemented. Used to tell which field upgrade command pair an ST33 part supports

```
WOLFTPM_API int wolfTPM2_ST33_GetFwUpgradeCommands(
    const WOLFTPM2_CAPS * caps,
    word16 manifestMajor,
    TPM_CC * ccStart,
    TPM_CC * ccData,
    int * fromTpm
)
```

Choose the ST33 field upgrade command codes for the attached TPM.

Parameters:

- **caps** capabilities of the attached TPM, or NULL once it has entered firmware upgrade mode, where no capability query may be issued because the TPM accepts only FieldUpgradeData and any other command leaves the mode. With NULL the firmware version rule decides alone
- **manifestMajor** target major version from wolfTPM2_ST33_ManifestVersion
- **ccStart** pointer to store the field upgrade start command code
- **ccData** pointer to store the field upgrade data command code
- **fromTpm** pointer to store 1 when the TPM decided and 0 when the version rule did (optional, may be NULL)

See:

- wolfTPM2_ST33_CmdImplemented
- wolfTPM2_ST33_ManifestVersion

Return:

- TPM_RC_SUCCESS: successful
- BAD_FUNC_ARG: ccStart or ccData is NULL

Note: Asks the TPM which pair it implements, since the firmware version rule does not hold on every ST33 firmware line, and falls back to wolfTPM2_ST33_FwUpgradeCommands when the TPM does not settle it. This is what the library sends, so a caller binding a PolicyCommandCode for the upgrade should bind the start code this returns

```
WOLFTPM_TEST_API int wolfTPM2_ST33_FwUpgradeCommands(
    word16 fwVerMinor,
    int haveFwVer,
    word16 manifestMajor,
    TPM_CC * ccStart,
    TPM_CC * ccData
)
```

Choose the ST33 field upgrade command codes from the version fields.

Parameters:

- **fwVerMinor** running firmware minor version from WOLFTPM2_CAPS
- **haveFwVer** set to 1 when fwVerMinor is known, 0 when it is not
- **manifestMajor** target major version from wolfTPM2_ST33_ManifestVersion
- **ccStart** pointer to store the field upgrade start command code
- **ccData** pointer to store the field upgrade data command code

See:

- wolfTPM2_ST33_ManifestVersion
- wolfTPM2_FirmwareUpgrade

Return:

- TPM_RC_SUCCESS: successful
- BAD_FUNC_ARG: ccStart or ccData is NULL

Note: ST33 implements the field upgrade with either the standard TPM_CC_FieldUpgradeStart/Data or the ST33KTPM vendor codes, and the wrong pair is answered with TPM_RC_COMMAND_CODE. This applies the rule ST's reference tool uses. Callers want wolfTPM2_ST33_GetFwUpgradeCommands instead, which asks the TPM first and only falls back to this. Pass haveFwVer = 0 when the running version is unknown, as it is once the TPM has entered upgrade mode

29.8 wolftpm/tpm2.h

29.7.6.261 function wolfTPM2_ST33_FwUpgradeCommands

29.8.1 Classes

	Name
struct	**TPMS_ALGORITHM_DESCRIPTION**
union	**TPMU_HA**
struct	**TPMT_HA**
struct	**TPM2B_DIGEST**
struct	**TPM2B_DATA**
struct	**TPM2B_EVENT**
struct	**TPM2B_MAX_BUFFER**
struct	**TPM2B_MAX_NV_BUFFER**
struct	**TPM2B_IV**
struct	**TPM2B_SIGNATURE_CTX**
struct	**TPM2B_SIGNATURE_HINT**
struct	**TPM2B_KEM_CIPHERTEXT**
struct	**TPM2B_SHARED_SECRET**
struct	**TPM2B_PUBLIC_KEY_MLDSA**
struct	**TPM2B_PRIVATE_KEY_MLDSA**
struct	**TPM2B_PUBLIC_KEY_MLKEM**
struct	**TPM2B_PRIVATE_KEY_MLKEM**
struct	**TPM2B_MLDSA_SIGNATURE**
union	**TPMU_NAME**
struct	**TPM2B_NAME**
struct	**TPMS_PCR_SELECT**
struct	**TPMS_PCR_SELECTION**
struct	**TPMT_TK_CREATION**
struct	**TPMT_TK_VERIFIED**
struct	**TPMT_TK_AUTH**
struct	**TPMT_TK_HASHCHECK**
struct	**TPMS_ALG_PROPERTY**
struct	**TPMS_TAGGED_PROPERTY**
struct	**TPMS_TAGGED_PCR_SELECT**
struct	**TPMS_TAGGED_POLICY**
struct	**TPMS_TAGGED_CONFIG**
struct	**TPML_CC**
struct	**TPML_CCA**
struct	**TPML_ALG**
struct	**TPML_HANDLE**
struct	**TPML_DIGEST**
struct	**TPML_DIGEST_VALUES**
struct	**TPML_PCR_SELECTION**
struct	**TPML_ALG_PROPERTY**
struct	**TPML_TAGGED_TPM_PROPERTY**
struct	**TPML_TAGGED_PCR_PROPERTY**
struct	**TPML_ECC_CURVE**
struct	**TPML_TAGGED_POLICY**
struct	**TPML_TAGGED_CONFIG**
struct	**TPMS_ACT_DATA**
struct	**TPML_ACT_DATA**
struct	**TPMS_SPDM_SESSION_INFO**
struct	**TPML_SPDM_SESSION_INFO**
union	**TPMU_CAPABILITIES**
struct	**TPMS_CAPABILITY_DATA**
struct	**TPMS_CLOCK_INFO**
struct	**TPMS_TIME_INFO**

	Name
struct	**TPMS_TIME_ATTEST_INFO**
struct	**TPMS_CERTIFY_INFO**
struct	**TPMS_QUOTE_INFO**
struct	**TPMS_COMMAND_AUDIT_INFO**
struct	**TPMS_SESSION_AUDIT_INFO**
struct	**TPMS_CREATION_INFO**
struct	**TPMS_NV_CERTIFY_INFO**
struct	**TPMS_NV_DIGEST_CERTIFY_INFO**
union	**TPMU_ATTEST**
struct	**TPMS_ATTEST**
struct	**TPM2B_ATTEST**
union	**TPMU_SYM_KEY_BITS**
union	**TPMU_SYM_MODE**
struct	**TPMT_SYM_DEF**
struct	**TPM2B_SYM_KEY**
struct	**TPMS_SYMCIPHER_PARMS**
struct	**TPM2B_LABEL**
struct	**TPMS_DERIVE**
struct	**TPM2B_DERIVE**
union	**TPMU_SENSITIVE_CREATE**
struct	**TPM2B_SENSITIVE_DATA**
struct	**TPMS_SENSITIVE_CREATE**
struct	**TPM2B_SENSITIVE_CREATE**
struct	**TPMS_SCHEME_HASH**
struct	**TPMS_SCHEME_ECDAA**
struct	**TPMS_SCHEME_XOR**
union	**TPMU_SCHEME_KEYEDHASH**
struct	**TPMT_KEYEDHASH_SCHEME**
union	**TPMU_SIG_SCHEME**
struct	**TPMT_SIG_SCHEME**
union	**TPMU_KDF_SCHEME**
struct	**TPMT_KDF_SCHEME**
union	**TPMU_ASYM_SCHEME**
struct	**TPMT_ASYM_SCHEME**
struct	**TPMT_RSA_SCHEME**
struct	**TPMT_RSA_DECRYPT**
struct	**TPM2B_PUBLIC_KEY_RSA**
struct	**TPM2B_PRIVATE_KEY_RSA**
struct	**TPM2B_ECC_PARAMETER**
struct	**TPMS_ECC_POINT**
struct	**TPM2B_ECC_POINT**
struct	**TPMS_ALGORITHM_DETAIL_ECC**
struct	**TPMS_SIGNATURE_RSA**
struct	**TPMS_SIGNATURE_ECC**
struct	**TPMS_SIGNATURE_HASH_MLDSA**
union	**TPMU_SIGNATURE**
struct	**TPMT_SIGNATURE**
union	**TPMU_ENCRYPTED_SECRET**
struct	**TPM2B_ENCRYPTED_SECRET**
union	**TPMU_PUBLIC_ID**
struct	**TPMS_KEYEDHASH_PARMS**
struct	**TPMS_ASYM_PARMS**

	Name
struct	**TPMS_RSA_PARMS**
struct	**TPMS_ECC_PARMS**
struct	**TPMS_MLDSA_PARMS**
struct	**TPMS_HASH_MLDSA_PARMS**
struct	**TPMS_MLKEM_PARMS**
union	**TPMU_PUBLIC_PARMS**
struct	**TPMT_PUBLIC_PARMS**
struct	**TPMT_PUBLIC**
struct	**TPM2B_PUBLIC**
struct	**TPM2B_TEMPLATE**
struct	**TPM2B_PRIVATE_VENDOR_SPECIFIC**
union	**TPMU_SENSITIVE_COMPOSITE**
struct	**TPMT_SENSITIVE**
struct	**TPM2B_SENSITIVE**
struct	**TPMT_PRIVATE**
struct	**TPM2B_PRIVATE**
struct	**TPMS_ID_OBJECT**
struct	**TPM2B_ID_OBJECT**
struct	**TPMS_NV_PIN_COUNTER_PARAMETERS**
struct	**TPMS_NV_PUBLIC**
struct	**TPM2B_NV_PUBLIC**
struct	**TPM2B_CONTEXT_SENSITIVE**
struct	**TPMS_CONTEXT_DATA**
struct	**TPM2B_CONTEXT_DATA**
struct	**TPMS_CONTEXT**
struct	**TPMS_CREATION_DATA**
struct	**TPM2B_CREATION_DATA**
struct	**TPMS_AUTH_COMMAND**
struct	**TPMS_AUTH_RESPONSE**
struct	**TPM2_AUTH_SESSION**
struct	**wolfTPM_tcpContext**
struct	**wolfTPM_winContext**
struct	**TPM2_CTX**
struct	**Startup_In**
struct	**Shutdown_In**
struct	**GetCapability_In**
struct	**GetCapability_Out**
struct	**SetCapability_In**
struct	**SelfTest_In**
struct	**IncrementalSelfTest_In**
struct	**IncrementalSelfTest_Out**
struct	**GetTestResult_Out**
struct	**GetRandom_In**
struct	**GetRandom_Out**
struct	**StirRandom_In**
struct	**PCR_Read_In**
struct	**PCR_Read_Out**
struct	**PCR_Extend_In**
struct	**Create_In**
struct	**Create_Out**
struct	**CreateLoaded_In**
struct	**CreateLoaded_Out**

	Name
struct	**CreatePrimary_In**
struct	**CreatePrimary_Out**
struct	**Load_In**
struct	**Load_Out**
struct	**FlushContext_In**
struct	**Unseal_In**
struct	**Unseal_Out**
struct	**StartAuthSession_In**
struct	**StartAuthSession_Out**
struct	**PolicyRestart_In**
struct	**LoadExternal_In**
struct	**LoadExternal_Out**
struct	**ReadPublic_In**
struct	**ReadPublic_Out**
struct	**ActivateCredential_In**
struct	**ActivateCredential_Out**
struct	**MakeCredential_In**
struct	**MakeCredential_Out**
struct	**ObjectChangeAuth_In**
struct	**ObjectChangeAuth_Out**
struct	**Duplicate_In**
struct	**Duplicate_Out**
struct	**Rewrap_In**
struct	**Rewrap_Out**
struct	**Import_In**
struct	**Import_Out**
struct	**RSA_Encrypt_In**
struct	**RSA_Encrypt_Out**
struct	**RSA_Decrypt_In**
struct	**RSA_Decrypt_Out**
struct	**ECDH_KeyGen_In**
struct	**ECDH_KeyGen_Out**
struct	**ECDH_ZGen_In**
struct	**ECDH_ZGen_Out**
struct	**ECC_Parameters_In**
struct	**ECC_Parameters_Out**
struct	**ZGen_2Phase_In**
struct	**ZGen_2Phase_Out**
struct	**EncryptDecrypt_In**
struct	**EncryptDecrypt_Out**
struct	**EncryptDecrypt2_In**
struct	**EncryptDecrypt2_Out**
struct	**Hash_In**
struct	**Hash_Out**
struct	**HMAC_In**
struct	**HMAC_Out**
struct	**HMAC_Start_In**
struct	**HMAC_Start_Out**
struct	**HashSequenceStart_In**
struct	**HashSequenceStart_Out**
struct	**SequenceUpdate_In**
struct	**SequenceComplete_In**

	Name
struct	**SequenceComplete_Out**
struct	**EventSequenceComplete_In**
struct	**EventSequenceComplete_Out**
struct	**Certify_In**
struct	**Certify_Out**
struct	**CertifyCreation_In**
struct	**CertifyCreation_Out**
struct	**Quote_In**
struct	**Quote_Out**
struct	**GetSessionAuditDigest_In**
struct	**GetSessionAuditDigest_Out**
struct	**GetCommandAuditDigest_In**
struct	**GetCommandAuditDigest_Out**
struct	**GetTime_In**
struct	**GetTime_Out**
struct	**Commit_In**
struct	**Commit_Out**
struct	**EC_Ephemeral_In**
struct	**EC_Ephemeral_Out**
struct	**VerifySignature_In**
struct	**VerifySignature_Out**
struct	**Sign_In**
struct	**Sign_Out**
struct	**SignSequenceStart_In**
struct	**SignSequenceStart_Out**
struct	**VerifySequenceStart_In**
struct	**VerifySequenceStart_Out**
struct	**SignSequenceComplete_In**
struct	**SignSequenceComplete_Out**
struct	**VerifySequenceComplete_In**
struct	**VerifySequenceComplete_Out**
struct	**SignDigest_In**
struct	**SignDigest_Out**
struct	**VerifyDigestSignature_In**
struct	**VerifyDigestSignature_Out**
struct	**Encapsulate_In**
struct	**Encapsulate_Out**
struct	**Decapsulate_In**
struct	**Decapsulate_Out**
struct	**SetCommandCodeAuditStatus_In**
struct	**PCR_Event_In**
struct	**PCR_Event_Out**
struct	**PCR_Allocate_In**
struct	**PCR_Allocate_Out**
struct	**PCR_SetAuthPolicy_In**
struct	**PCR_SetAuthValue_In**
struct	**PCR_Reset_In**
struct	**PolicySigned_In**
struct	**PolicySigned_Out**
struct	**PolicySecret_In**
struct	**PolicySecret_Out**
struct	**PolicyTicket_In**

	Name
struct	**PolicyOR_In**
struct	**PolicyPCR_In**
struct	**PolicyTransportSPDM_In**
struct	**PolicyLocality_In**
struct	**PolicyNV_In**
struct	**PolicyCounterTimer_In**
struct	**PolicyCommandCode_In**
struct	**PolicyPhysicalPresence_In**
struct	**PolicyCpHash_In**
struct	**PolicyNameHash_In**
struct	**PolicyDuplicationSelect_In**
struct	**PolicyAuthorize_In**
struct	**PolicyAuthValue_In**
struct	**PolicyPassword_In**
struct	**PolicyGetDigest_In**
struct	**PolicyGetDigest_Out**
struct	**PolicyNvWritten_In**
struct	**PolicyTemplate_In**
struct	**PolicyAuthorizeNV_In**
struct	**HierarchyControl_In**
struct	**SetPrimaryPolicy_In**
struct	**ChangeSeed_In**
struct	**Clear_In**
struct	**ClearControl_In**
struct	**HierarchyChangeAuth_In**
struct	**DictionaryAttackLockReset_In**
struct	**DictionaryAttackParameters_In**
struct	**PP_Commands_In**
struct	**SetAlgorithmSet_In**
struct	**FieldUpgradeStart_In**
struct	**FieldUpgradeData_In**
struct	**FieldUpgradeData_Out**
struct	**FirmwareRead_In**
struct	**FirmwareRead_Out**
struct	**ContextSave_In**
struct	**ContextSave_Out**
struct	**ContextLoad_In**
struct	**ContextLoad_Out**
struct	**EvictControl_In**
struct	**ReadClock_Out**
struct	**ClockSet_In**
struct	**ClockRateAdjust_In**
struct	**TestParms_In**
struct	**NV_DefineSpace_In**
struct	**NV_UndefineSpace_In**
struct	**NV_UndefineSpaceSpecial_In**
struct	**NV_ReadPublic_In**
struct	**NV_ReadPublic_Out**
struct	**NV_Write_In**
struct	**NV_Increment_In**
struct	**NV_Extend_In**
struct	**NV_SetBits_In**

	Name
struct	**NV_WriteLock_In**
struct	**NV_GlobalWriteLock_In**
struct	**NV_Read_In**
struct	**NV_Read_Out**
struct	**NV_ReadLock_In**
struct	**NV_ChangeAuth_In**
struct	**NV_Certify_In**
struct	**NV_Certify_Out**
struct	**SetCommandSet_In**
struct	**TPM_MODE_SET**
struct	**SetMode_In**
struct	**GetRandom2_Out**
struct	**TPMS_GPIO_CONFIG**
struct	**TPML_GPIO_CONFIG**
struct	**GpioConfig_In**
struct	**CFG_STRUCT**
struct	**NTC2_PreConfig_In**
struct	**NTC2_GetConfig_Out**
struct	**Nations_IdentityKeySet_In**

29.8.2 Types

	Name
enum	**TPM_ALG_ID_T** { TPM_ALG_ERROR = 0x0000, TPM_ALG_FIRST = 0x0001, TPM_ALG_RSA = TPM_ALG_FIRST, TPM_ALG_SHA = 0x0004, TPM_ALG_SHA1 = TPM_ALG_SHA, TPM_ALG_HMAC = 0x0005, TPM_ALG_AES = 0x0006, TPM_ALG_MGF1 = 0x0007, TPM_ALG_KEYEDHASH = 0x0008, TPM_ALG_XOR = 0x000A, TPM_ALG_SHA256 = 0x000B, TPM_ALG_SHA384 = 0x000C, TPM_ALG_SHA512 = 0x000D, TPM_ALG_NULL = 0x0010, TPM_ALG_SM3_256 = 0x0012, TPM_ALG_SM4 = 0x0013, TPM_ALG_RSASSA = 0x0014, TPM_ALG_RSAES = 0x0015, TPM_ALG_RSAPSS = 0x0016, TPM_ALG_OAEP = 0x0017, TPM_ALG_ECDSA = 0x0018, TPM_ALG_ECDH = 0x0019, TPM_ALG_ECDAA = 0x001A, TPM_ALG_SM2 = 0x001B, TPM_ALG_ECSCHNORR = 0x001C, TPM_ALG_ECMQV = 0x001D, TPM_ALG_HKDF = 0x001F, TPM_ALG_KDF1_SP800_56A = 0x0020, TPM_ALG_KDF2 = 0x0021, TPM_ALG_KDF1_SP800_108 = 0x0022, TPM_ALG_ECC = 0x0023, TPM_ALG_SYMCIPHER = 0x0025, TPM_ALG_CAMELLIA = 0x0026, TPM_ALG_SHA3_256 = 0x0027, TPM_ALG_SHA3_384 = 0x0028, TPM_ALG_SHA3_512 = 0x0029, TPM_ALG_SHAKE128 = 0x002A, TPM_ALG_SHAKE256 = 0x002B, TPM_ALG_CTR = 0x0040, TPM_ALG_OFB = 0x0041, TPM_ALG_CBC = 0x0042, TPM_ALG_CFB = 0x0043, TPM_ALG_ECB = 0x0044, TPM_ALG_MLKEM = 0x00A0, TPM_ALG_MLDSA = 0x00A1, TPM_ALG_HASH_MLDSA = 0x00A2}
enum	**TPM_ECC_CURVE_T** { TPM_ECC_NONE = 0x0000, TPM_ECC_NIST_P192 = 0x0001, TPM_ECC_NIST_P224 = 0x0002, TPM_ECC_NIST_P256 = 0x0003, TPM_ECC_NIST_P384 = 0x0004, TPM_ECC_NIST_P521 = 0x0005, TPM_ECC_BN_P256 = 0x0010, TPM_ECC_BN_P638 = 0x0011, TPM_ECC_SM2_P256 = 0x0020, TPM_ECC_BP_P256_R1 = 0x0030, TPM_ECC_BP_P384_R1 = 0x0031, TPM_ECC_BP_P512_R1 = 0x0032}

	Name
enum	**TPM_CC_T** { TPM_CC_FIRST = 0x0000011F, TPM_CC_NV_UndefineSpaceSpecial = TPM_CC_FIRST, TPM_CC_EvictControl = 0x00000120, TPM_CC_HierarchyControl = 0x00000121, TPM_CC_NV_UndefineSpace = 0x00000122, TPM_CC_ChangeEPS = 0x00000124 TPM_CC_ChangePPS = 0x00000125, TPM_CC_Clear = 0x00000126, TPM_CC_ClearControl = 0x00000127, TPM_CC_ClockSet = 0x00000128, TPM_CC_HierarchyChangeAuth = 0x00000129, TPM_CC_NV_DefineSpace = 0x0000012A TPM_CC_PCR_Allocate = 0x0000012B, TPM_CC_PCR_SetAuthPolicy = 0x0000012C, TPM_CC_PP_Commands = 0x0000012D, TPM_CC_SetPrimaryPolicy = 0x0000012E, TPM_CC_FieldUpgradeStart = 0x0000012F, TPM_CC_ClockRateAdjust = 0x00000130 TPM_CC_CreatePrimary = 0x00000131, TPM_CC_NV_GlobalWriteLock = 0x00000132, TPM_CC_GetCommandAuditDigest = 0x00000133, TPM_CC_NV_Increment = 0x00000134, TPM_CC_NV_SetBits = 0x00000135, TPM_CC_NV_Extend = 0x00000136 TPM_CC_NV_Write = 0x00000137, TPM_CC_NV_WriteLock = 0x00000138, TPM_CC_DictionaryAttackLockReset = 0x00000139, TPM_CC_DictionaryAttackParameters = 0x0000013A, TPM_CC_NV_ChangeAuth = 0x0000013B TPM_CC_PCR_Event = 0x0000013C, TPM_CC_PCR_Reset = 0x0000013D, TPM_CC_SequenceComplete = 0x0000013E, TPM_CC_SetAlgorithmSet = 0x0000013F, TPM_CC_SetCommandCodeAuditStatus = 0x00000140, TPM_CC_FieldUpgradeData = 0x00000141 TPM_CC_IncrementalSelfTest = 0x00000142, TPM_CC_SelfTest = 0x00000143, TPM_CC_Startup = 0x00000144, TPM_CC_Shutdown = 0x00000145, TPM_CC_StirRandom = 0x00000146, TPM_CC_ActivateCredential = 0x00000147 TPM_CC_Certify = 0x00000148, TPM_CC_PolicyNV = 0x00000149, TPM_CC_CertifyCreation = 0x0000014A, TPM_CC_Duplicate = 0x0000014B, TPM_CC_GetTime = 0x0000014C, TPM_CC_GetSessionAuditDigest = 0x0000014D

	Name
	TPM_CC_NV_Read = 0x0000014E, TPM_CC_NV_ReadLock = 0x0000014F, TPM_CC_ObjectChangeAuth = 0x00000150, TPM_CC_PolicySecret = 0x00000151, TPM_CC_Rewrap = 0x00000152, TPM_CC_Create = 0x00000153, TPM_CC_ECDH_ZGen = 0x00000154 TPM_CC_HMAC = 0x00000155, TPM_CC_Import = 0x00000156, TPM_CC_Load = 0x00000157, TPM_CC_Quote = 0x00000158, TPM_CC_RSA_Decrypt = 0x00000159, TPM_CC_HMAC_Start = 0x0000015B, TPM_CC_SequenceUpdate = 0x0000015C TPM_CC_Sign = 0x0000015D, TPM_CC_Unseal = 0x0000015E, TPM_CC_PolicySigned = 0x00000160, TPM_CC_ContextLoad = 0x00000161, TPM_CC_ContextSave = 0x00000162, TPM_CC_ECDH_KeyGen = 0x00000163 TPM_CC_EncryptDecrypt = 0x00000164, TPM_CC_FlushContext = 0x00000165, TPM_CC_LoadExternal = 0x00000167, TPM_CC_MakeCredential = 0x00000168, TPM_CC_NV_ReadPublic = 0x00000169, TPM_CC_PolicyAuthorize = 0x0000016A TPM_CC_PolicyAuthValue = 0x0000016B, TPM_CC_PolicyCommandCode = 0x0000016C, TPM_CC_PolicyCounterTimer = 0x0000016D, TPM_CC_PolicyCpHash = 0x0000016E, TPM_CC_PolicyLocality = 0x0000016F, TPM_CC_PolicyNameHash = 0x00000170 TPM_CC_PolicyOR = 0x00000171, TPM_CC_PolicyTicket = 0x00000172, TPM_CC_ReadPublic = 0x00000173, TPM_CC_RSA_Encrypt = 0x00000174, TPM_CC_StartAuthSession = 0x00000176, TPM_CC_VerifySignature = 0x00000177 TPM_CC_ECC_Parameters = 0x00000178, TPM_CC_FirmwareRead = 0x00000179, TPM_CC_GetCapability = 0x0000017A, TPM_CC_GetRandom = 0x0000017B, TPM_CC_GetTestResult = 0x0000017C, TPM_CC_Hash = 0x0000017D TPM_CC_PCR_Read = 0x0000017E, TPM_CC_PolicyPCR = 0x0000017F, TPM_CC_PolicyRestart = 0x00000180, TPM_CC_ReadClock = 0x00000181, TPM_CC_PCR_Extend = 0x00000182, TPM_CC_PCR_SetAuthValue = 0x00000183

	Name
	TPM_CC_NV_Certify = 0x00000184, TPM_CC_EventSequenceComplete = 0x00000185, TPM_CC_HashSequenceStart = 0x00000186, TPM_CC_PolicyPhysicalPresence = 0x00000187, TPM_CC_PolicyDuplicationSelect = 0x00000188 TPM_CC_PolicyGetDigest = 0x00000189, TPM_CC_TestParms = 0x0000018A, TPM_CC_Commit = 0x0000018B, TPM_CC_PolicyPassword = 0x0000018C, TPM_CC_ZGen_2Phase = 0x0000018D, TPM_CC_EC_Ephemeral = 0x0000018E TPM_CC_PolicyNvWritten = 0x0000018F, TPM_CC_PolicyTemplate = 0x00000190, TPM_CC_CreateLoaded = 0x00000191, TPM_CC_PolicyAuthorizeNV = 0x00000192, TPM_CC_EncryptDecrypt2 = 0x00000193, TPM_CC_SetCapability = 0x0000019F TPM_CC_PolicyTransportSPDM = 0x000001A1, TPM_CC_VerifySequenceComplete = 0x000001A3, TPM_CC_SignSequenceComplete = 0x000001A4, TPM_CC_VerifyDigestSignature = 0x000001A5, TPM_CC_SignDigest = 0x000001A6 TPM_CC_Encapsulate = 0x000001A7, TPM_CC_Decapsulate = 0x000001A8, TPM_CC_VerifySequenceStart = 0x000001A9, TPM_CC_SignSequenceStart = 0x000001AA, TPM_CC_LAST = TPM_CC_SignSequenceStart TPM_CC_LAST = TPM_CC_SignSequenceStart, TPM_CC_LAST = TPM_CC_SignSequenceStart, CC_VEND = 0x20000000, TPM_CC_Vendor_TCG_Test = CC_VEND + 0x0000, TPM_CC_SetMode = CC_VEND + 0x0307, TPM_CC_SetCommandSet = CC_VEND + 0x0309 TPM_CC_GetRandom2 = CC_VEND + 0x030E, TPM_CC_RestoreEK = CC_VEND + 0x030A, TPM_CC_SetCommandSetLock = CC_VEND + 0x030B, TPM_CC_GPIO_Config = CC_VEND + 0x030F, TPM_CC_NTC2_PreConfig = CC_VEND + 0x0211 TPM_CC_NTC2_GetConfig = CC_VEND + 0x0213, TPM_CC_Nations_IdentityKeySet = 0x20000708, TPM_CC_FieldUpgradeStartVendor = CC_VEND + 0x12F, TPM_CC_FieldUpgradeAbandonVendor = CC_VEND + 0x130

	Name
	TPM_CC_FieldUpgradeManifestVendor = CC_VEND + 0x131, TPM_CC_FieldUpgradeDataVendor = CC_VEND + 0x132, TPM_CC_FieldUpgradeFinalizeVendor = CC_VEND + 0x133, TPM_CC_FieldUpgradeStartVendor_ST33 = CC_VEND + 0x030C TPM_CC_FieldUpgradeAbandonVendor_ST33 = CC_VEND + 0x030E, TPM_CC_FieldUpgradeDataVendor_ST33 = CC_VEND + 0x030D, TPM_CC_FieldUpgradeFinalizeVendor_ST33 = CC_VEND + 0x030F }
enum	**TPM_RC_T** { TPM_RC_SUCCESS = 0x000, TPM_RC_BAD_TAG = 0x01E, RC_VER1 = 0x100, TPM_RC_INITIALIZE = RC_VER1 + 0x000, TPM_RC_FAILURE = RC_VER1 + 0x001, TPM_RC_SEQUENCE = RC_VER1 + 0x003, TPM_RC_PRIVATE = RC_VER1 + 0x00B TPM_RC_HMAC = RC_VER1 + 0x019, TPM_RC_DISABLED = RC_VER1 + 0x020, TPM_RC_EXCLUSIVE = RC_VER1 + 0x021, TPM_RC_AUTH_TYPE = RC_VER1 + 0x024, TPM_RC_AUTH_MISSING = RC_VER1 + 0x025, TPM_RC_POLICY = RC_VER1 + 0x026 TPM_RC_PCR = RC_VER1 + 0x027, TPM_RC_PCR_CHANGED = RC_VER1 + 0x028, TPM_RC_UPGRADE = RC_VER1 + 0x02D, TPM_RC_TOO_MANY_CONTEXTS = RC_VER1 + 0x02E, TPM_RC_AUTH_UNAVAILABLE = RC_VER1 + 0x02F, TPM_RC_REBOOT = RC_VER1 + 0x030 TPM_RC_UNBALANCED = RC_VER1 + 0x031, TPM_RC_COMMAND_SIZE = RC_VER1 + 0x042, TPM_RC_COMMAND_CODE = RC_VER1 + 0x043, TPM_RC_AUTHSIZE = RC_VER1 + 0x044, TPM_RC_AUTH_CONTEXT = RC_VER1 + 0x045 TPM_RC_NV_RANGE = RC_VER1 + 0x046, TPM_RC_NV_SIZE = RC_VER1 + 0x047, TPM_RC_NV_LOCKED = RC_VER1 + 0x048, TPM_RC_NV_AUTHORIZATION = RC_VER1 + 0x049, TPM_RC_NV_UNINITIALIZED = RC_VER1 + 0x04A TPM_RC_NV_SPACE = RC_VER1 + 0x04B, TPM_RC_NV_DEFINED = RC_VER1 + 0x04C, TPM_RC_BAD_CONTEXT = RC_VER1 + 0x050, TPM_RC_CPHASH = RC_VER1 + 0x051, TPM_RC_PARENT = RC_VER1 + 0x052, TPM_RC_NEEDS_TEST = RC_VER1 + 0x053

	Name
	TPM_RC_NO_RESULT = RC_VER1 + 0x054, TPM_RC_SENSITIVE = RC_VER1 + 0x055, RC_MAX_FM0 = RC_VER1 + 0x07F, RC_FMT1 = 0x080, TPM_RC_ASYMMETRIC = RC_FMT1 + 0x001, TPM_RC_ATTRIBUTES = RC_FMT1 + 0x002 TPM_RC_HASH = RC_FMT1 + 0x003, TPM_RC_VALUE = RC_FMT1 + 0x004, TPM_RC_HIERARCHY = RC_FMT1 + 0x005, TPM_RC_KEY_SIZE = RC_FMT1 + 0x007, TPM_RC_MGF = RC_FMT1 + 0x008, TPM_RC_MODE = RC_FMT1 + 0x009 TPM_RC_TYPE = RC_FMT1 + 0x00A, TPM_RC_HANDLE = RC_FMT1 + 0x00B, TPM_RC_KDF = RC_FMT1 + 0x00C, TPM_RC_RANGE = RC_FMT1 + 0x00D, TPM_RC_AUTH_FAIL = RC_FMT1 + 0x00E, TPM_RC_NONCE = RC_FMT1 + 0x00F TPM_RC_PP = RC_FMT1 + 0x010, TPM_RC_SCHEME = RC_FMT1 + 0x012, TPM_RC_SIZE = RC_FMT1 + 0x015, TPM_RC_SYMMETRIC = RC_FMT1 + 0x016, TPM_RC_TAG = RC_FMT1 + 0x017, TPM_RC_SELECTOR = RC_FMT1 + 0x018 TPM_RC_INSUFFICIENT = RC_FMT1 + 0x01A, TPM_RC_SIGNATURE = RC_FMT1 + 0x01B, TPM_RC_KEY = RC_FMT1 + 0x01C, TPM_RC_POLICY_FAIL = RC_FMT1 + 0x01D, TPM_RC_INTEGRITY = RC_FMT1 + 0x01F, TPM_RC_TICKET = RC_FMT1 + 0x020 TPM_RC_RESERVED_BITS = RC_FMT1 + 0x021, TPM_RC_BAD_AUTH = RC_FMT1 + 0x022, TPM_RC_EXPIRED = RC_FMT1 + 0x023, TPM_RC_POLICY_CC = RC_FMT1 + 0x024, TPM_RC_BINDING = RC_FMT1 + 0x025, TPM_RC_CURVE = RC_FMT1 + 0x026 TPM_RC_ECC_POINT = RC_FMT1 + 0x027, TPM_RC_FW_LIMITED = RC_FMT1 + 0x028, TPM_RC_SVN_LIMITED = RC_FMT1 + 0x029, TPM_RC_PARMS = RC_FMT1 + 0x02A, TPM_RC_EXT_MU = RC_FMT1 + 0x02B, TPM_RC_ONE_SHOT_SIGNATURE = RC_FMT1 + 0x02C TPM_RC_SIGN_CONTEXT_KEY = RC_FMT1 + 0x02D, TPM_RC_CHANNEL = RC_FMT1 + 0x030, TPM_RC_CHANNEL_KEY = RC_FMT1 + 0x031, RC_MAX_FMT1 = RC_FMT1 + 0x03F, RC_WARN = 0x900, TPM_RC_CONTEXT_GAP = RC_WARN + 0x001

	Name
	TPM_RC_OBJECT_MEMORY = RC_WARN + 0x002, TPM_RC_SESSION_MEMORY = RC_WARN + 0x003, TPM_RC_MEMORY = RC_WARN + 0x004, TPM_RC_SESSION_HANDLES = RC_WARN + 0x005, TPM_RC_OBJECT_HANDLES = RC_WARN + 0x006 TPM_RC_LOCALITY = RC_WARN + 0x007, TPM_RC_YIELDED = RC_WARN + 0x008, TPM_RC_CANCELED = RC_WARN + 0x009, TPM_RC_TESTING = RC_WARN + 0x00A, TPM_RC_REFERENCE_H0 = RC_WARN + 0x010, TPM_RC_REFERENCE_H1 = RC_WARN + 0x011 TPM_RC_REFERENCE_H2 = RC_WARN + 0x012, TPM_RC_REFERENCE_H3 = RC_WARN + 0x013, TPM_RC_REFERENCE_H4 = RC_WARN + 0x014, TPM_RC_REFERENCE_H5 = RC_WARN + 0x015, TPM_RC_REFERENCE_H6 = RC_WARN + 0x016 TPM_RC_REFERENCE_S0 = RC_WARN + 0x018, TPM_RC_REFERENCE_S1 = RC_WARN + 0x019, TPM_RC_REFERENCE_S2 = RC_WARN + 0x01A, TPM_RC_REFERENCE_S3 = RC_WARN + 0x01B, TPM_RC_REFERENCE_S4 = RC_WARN + 0x01C TPM_RC_REFERENCE_S5 = RC_WARN + 0x01D, TPM_RC_REFERENCE_S6 = RC_WARN + 0x01E, TPM_RC_NV_RATE = RC_WARN + 0x020, TPM_RC_LOCKOUT = RC_WARN + 0x021, TPM_RC_RETRY = RC_WARN + 0x022, TPM_RC_NV_UNAVAILABLE = RC_WARN + 0x023 RC_MAX_WARN = RC_WARN + 0x03F, TPM_RC_NOT_USED = RC_WARN + 0x07F, TPM_RC_H = 0x000, TPM_RC_P = 0x040, TPM_RC_S = 0x800, TPM_RC_1 = 0x100, TPM_RC_2 = 0x200, TPM_RC_3 = 0x300, TPM_RC_4 = 0x400, TPM_RC_5 = 0x500 TPM_RC_6 = 0x600, TPM_RC_7 = 0x700, TPM_RC_8 = 0x800, TPM_RC_9 = 0x900, TPM_RC_A = 0xA00, TPM_RC_B = 0xB00, TPM_RC_C = 0xC00, TPM_RC_D = 0xD00, TPM_RC_E = 0xE00, TPM_RC_F = 0xF00, TPM_RC_N_MASK = 0xF00 TPM_RC_TIMEOUT = -100 }
enum	**TPM_CLOCK_ADJUST_T** { TPM_CLOCK_COARSE_SLOWER = -3, TPM_CLOCK_MEDIUM_SLOWER = -2, TPM_CLOCK_FINE_SLOWER = -1, TPM_CLOCK_NO_CHANGE = 0, TPM_CLOCK_FINE_FASTER = 1, TPM_CLOCK_MEDIUM_FASTER = 2, TPM_CLOCK_COARSE_FASTER = 3}

	Name
enum	**TPM_EO_T** { TPM_EO_EQ = 0x0000, TPM_EO_NEQ = 0x0001, TPM_EO_SIGNED_GT = 0x0002, TPM_EO_UNSIGNED_GT = 0x0003, TPM_EO_SIGNED_LT = 0x0004, TPM_EO_UNSIGNED_LT = 0x0005, TPM_EO_SIGNED_GE = 0x0006, TPM_EO_UNSIGNED_GE = 0x0007, TPM_EO_SIGNED_LE = 0x0008, TPM_EO_UNSIGNED_LE = 0x0009, TPM_EO_BITSET = 0x000A, TPM_EO_BITCLEAR = 0x000B}
enum	**TPM_ST_T** { TPM_ST_RSP_COMMAND = 0x00C4, TPM_ST_NULL = 0X8000, TPM_ST_NO_SESSIONS = 0x8001, TPM_ST_SESSIONS = 0x8002, TPM_ST_ATTEST_NV = 0x8014, TPM_ST_ATTEST_COMMAND_AUDIT = 0x8015, TPM_ST_ATTEST_SESSION_AUDIT = 0x8016, TPM_ST_ATTEST_CERTIFY = 0x8017, TPM_ST_ATTEST_QUOTE = 0x8018, TPM_ST_ATTEST_TIME = 0x8019, TPM_ST_ATTEST_CREATION = 0x801A, TPM_ST_ATTEST_NV_DIGEST = 0x801C, TPM_ST_CREATION = 0x8021, TPM_ST_VERIFIED = 0x8022, TPM_ST_AUTH_SECRET = 0x8023, TPM_ST_HASHCHECK = 0x8024, TPM_ST_AUTH_SIGNED = 0x8025, TPM_ST_MESSAGE_VERIFIED = 0x8026, TPM_ST_DIGEST_VERIFIED = 0x8027, TPM_ST_FU_MANIFEST = 0x8029}
enum	**TPM_SE_T** { TPM_SE_HMAC = 0x00, TPM_SE_POLICY = 0x01, TPM_SE_TRIAL = 0x03}
enum	**TPM_SU_T** { TPM_SU_CLEAR = 0x0000, TPM_SU_STATE = 0x0001}

	Name
enum	**TPM_CAP_T** { TPM_CAP_FIRST = 0x00000000, TPM_CAP_ALGS = TPM_CAP_FIRST, TPM_CAP_HANDLES = 0x00000001, TPM_CAP_COMMANDS = 0x00000002, TPM_CAP_PP_COMMANDS = 0x00000003, TPM_CAP_AUDIT_COMMANDS = 0x00000004, TPM_CAP_PCRS = 0x00000005, TPM_CAP_TPM_PROPERTIES = 0x00000006, TPM_CAP_PCR_PROPERTIES = 0x00000007, TPM_CAP_ECC_CURVES = 0x00000008, TPM_CAP_AUTH_POLICIES = 0x00000009, TPM_CAP_ACT = 0x0000000A, TPM_CAP_PUB_KEYS = 0x0000000B, TPM_CAP_SPDM_SESSION_INFO = 0x0000000C, TPM_CAP_LAST = TPM_CAP_SPDM_SESSION_INFO, TPM_CAP_LAST = TPM_CAP_SPDM_SESSION_INFO, TPM_CAP_VENDOR_PROPERTY = 0x00000100}
enum	**TPM_PT_CONFIG_T** { TPM_PT_CONFIG_ENCRYPTDECRYPT2 = 0x00000000, TPM_PT_CONFIG_CHANGEEPS = 0x00000001, TPM_PT_CONFIG_TPMID_NV = 0x00000002, TPM_PT_CONFIG_SPDM_PSK = 0x00000003}
enum	**TPM_PT_T** { TPM_PT_NONE = 0x00000000, PT_GROUP = 0x00000100, PT_FIXED = PT_GROUP * 1, TPM_PT_FAMILY_INDICATOR = PT_FIXED + 0, TPM_PT_LEVEL = PT_FIXED + 1, TPM_PT_REVISION = PT_FIXED + 2, TPM_PT_DAY_OF_YEAR = PT_FIXED + 3 TPM_PT_YEAR = PT_FIXED + 4, TPM_PT_MANUFACTURER = PT_FIXED + 5, TPM_PT_VENDOR_STRING_1 = PT_FIXED + 6, TPM_PT_VENDOR_STRING_2 = PT_FIXED + 7, TPM_PT_VENDOR_STRING_3 = PT_FIXED + 8, TPM_PT_VENDOR_STRING_4 = PT_FIXED + 9 TPM_PT_VENDOR_TPM_TYPE = PT_FIXED + 10, TPM_PT_FIRMWARE_VERSION_1 = PT_FIXED + 11, TPM_PT_FIRMWARE_VERSION_2 = PT_FIXED + 12, TPM_PT_INPUT_BUFFER = PT_FIXED + 13, TPM_PT_HR_TRANSIENT_MIN = PT_FIXED + 14 TPM_PT_HR_PERSISTENT_MIN = PT_FIXED + 15, TPM_PT_HR_LOADED_MIN = PT_FIXED + 16, TPM_PT_ACTIVE_SESSIONS_MAX = PT_FIXED + 17, TPM_PT_PCR_COUNT = PT_FIXED + 18, TPM_PT_PCR_SELECT_MIN = PT_FIXED + 19

COPYRIGHT ©2026 wolfSSL Inc.

	Name
	TPM_PT_CONTEXT_GAP_MAX = PT_FIXED + 20, TPM_PT_NV_COUNTERS_MAX = PT_FIXED + 22, TPM_PT_NV_INDEX_MAX = PT_FIXED + 23, TPM_PT_MEMORY = PT_FIXED + 24, TPM_PT_CLOCK_UPDATE = PT_FIXED + 25, TPM_PT_CONTEXT_HASH = PT_FIXED + 26 TPM_PT_CONTEXT_SYM = PT_FIXED + 27, TPM_PT_CONTEXT_SYM_SIZE = PT_FIXED + 28, TPM_PT_ORDERLY_COUNT = PT_FIXED + 29, TPM_PT_MAX_COMMAND_SIZE = PT_FIXED + 30, TPM_PT_MAX_RESPONSE_SIZE = PT_FIXED + 31 TPM_PT_MAX_DIGEST = PT_FIXED + 32, TPM_PT_MAX_OBJECT_CONTEXT = PT_FIXED + 33, TPM_PT_MAX_SESSION_CONTEXT = PT_FIXED + 34, TPM_PT_PS_FAMILY_INDICATOR = PT_FIXED + 35, TPM_PT_PS_LEVEL = PT_FIXED + 36 TPM_PT_PS_REVISION = PT_FIXED + 37, TPM_PT_PS_DAY_OF_YEAR = PT_FIXED + 38, TPM_PT_PS_YEAR = PT_FIXED + 39, TPM_PT_SPLIT_MAX = PT_FIXED + 40, TPM_PT_TOTAL_COMMANDS = PT_FIXED + 41, TPM_PT_LIBRARY_COMMANDS = PT_FIXED + 42 TPM_PT_VENDOR_COMMANDS = PT_FIXED + 43, TPM_PT_NV_BUFFER_MAX = PT_FIXED + 44, TPM_PT_MODES = PT_FIXED + 45, TPM_PT_MAX_CAP_BUFFER = PT_FIXED + 46, TPM_PT_FIRMWARE_SVN = PT_FIXED + 47 TPM_PT_FIRMWARE_MAX_SVN = PT_FIXED + 48, TPM_PT_ML_PARAMETER_SETS = PT_FIXED + 49, PT_VAR = PT_GROUP * 2, TPM_PT_PERMANENT = PT_VAR + 0, TPM_PT_STARTUP_CLEAR = PT_VAR + 1, TPM_PT_HR_NV_INDEX = PT_VAR + 2 TPM_PT_HR_LOADED = PT_VAR + 3, TPM_PT_HR_LOADED_AVAIL = PT_VAR + 4, TPM_PT_HR_ACTIVE = PT_VAR + 5, TPM_PT_HR_ACTIVE_AVAIL = PT_VAR + 6, TPM_PT_HR_TRANSIENT_AVAIL = PT_VAR + 7, TPM_PT_HR_PERSISTENT = PT_VAR + 8 TPM_PT_HR_PERSISTENT_AVAIL = PT_VAR + 9, TPM_PT_NV_COUNTERS = PT_VAR + 10, TPM_PT_NV_COUNTERS_AVAIL = PT_VAR + 11, TPM_PT_ALGORITHM_SET = PT_VAR + 12, TPM_PT_LOADED_CURVES = PT_VAR + 13 TPM_PT_LOCKOUT_COUNTER = PT_VAR + 14, TPM_PT_MAX_AUTH_FAIL = PT_VAR + 15, TPM_PT_LOCKOUT_INTERVAL = PT_VAR + 16, TPM_PT_LOCKOUT_RECOVERY = PT_VAR + 17, TPM_PT_NV_WRITE_RECOVERY = PT_VAR + 18 TPM_PT_AUDIT_COUNTER_0 = PT_VAR + 19, TPM_PT_AUDIT_COUNTER_1 = PT_VAR + 20 }

	Name
enum	**TPM_PT_PCR_T** { TPM_PT_PCR_FIRST = 0x00000000, TPM_PT_PCR_SAVE = TPM_PT_PCR_FIRST, TPM_PT_PCR_EXTEND_L0 = 0x00000001, TPM_PT_PCR_RESET_L0 = 0x00000002, TPM_PT_PCR_EXTEND_L1 = 0x00000003, TPM_PT_PCR_RESET_L1 = 0x00000004, TPM_PT_PCR_EXTEND_L2 = 0x00000005, TPM_PT_PCR_RESET_L2 = 0x00000006, TPM_PT_PCR_EXTEND_L3 = 0x00000007, TPM_PT_PCR_RESET_L3 = 0x00000008, TPM_PT_PCR_EXTEND_L4 = 0x00000009, TPM_PT_PCR_RESET_L4 = 0x0000000A, TPM_PT_PCR_NO_INCREMENT = 0x00000011, TPM_PT_PCR_DRTM_RESET = 0x00000012, TPM_PT_PCR_POLICY = 0x00000013, TPM_PT_PCR_AUTH = 0x00000014, TPM_PT_PCR_LAST = TPM_PT_PCR_AUTH}
enum	**TPM_PS_T** { TPM_PS_MAIN = 0x00000000, TPM_PS_PC = 0x00000001, TPM_PS_PDA = 0x00000002, TPM_PS_CELL_PHONE = 0x00000003, TPM_PS_SERVER = 0x00000004, TPM_PS_PERIPHERAL = 0x00000005, TPM_PS_TSS = 0x00000006, TPM_PS_STORAGE = 0x00000007, TPM_PS_AUTHENTICATION = 0x00000008, TPM_PS_EMBEDDED = 0x00000009, TPM_PS_HARDCOPY = 0x0000000A, TPM_PS_INFRASTRUCTURE = 0x0000000B, TPM_PS_VIRTUALIZATION = 0x0000000C, TPM_PS_TNC = 0x0000000D, TPM_PS_MULTI_TENANT = 0x0000000E, TPM_PS_TC = 0x0000000F}
enum	**TPM_HT_T** { TPM_HT_PCR = 0x00, TPM_HT_NV_INDEX = 0x01, TPM_HT_HMAC_SESSION = 0x02, TPM_HT_LOADED_SESSION = 0x02, TPM_HT_POLICY_SESSION = 0x03, TPM_HT_ACTIVE_SESSION = 0x03, TPM_HT_PERMANENT = 0x40, TPM_HT_TRANSIENT = 0x80, TPM_HT_PERSISTENT = 0x81}

	Name
enum	**TPM_RH_T** { TPM_RH_FIRST = 0x40000000, TPM_RH_SRK = TPM_RH_FIRST, TPM_RH_OWNER = 0x40000001, TPM_RH_REVOKE = 0x40000002, TPM_RH_TRANSPORT = 0x40000003, TPM_RH_OPERATOR = 0x40000004, TPM_RH_ADMIN = 0x40000005, TPM_RH_EK = 0x40000006, TPM_RH_NULL = 0x40000007, TPM_RH_UNASSIGNED = 0x40000008, TPM_RS_PW = 0x40000009, TPM_RH_LOCKOUT = 0x4000000A, TPM_RH_ENDORSEMENT = 0x4000000B, TPM_RH_PLATFORM = 0x4000000C, TPM_RH_PLATFORM_NV = 0x4000000D, TPM_RH_AUTH_00 = 0x40000010, TPM_RH_AUTH_FF = 0x4000010F, TPM_RH_LAST = TPM_RH_AUTH_FF}
enum	**TPMA_ALGORITHM_mask** { TPMA_ALGORITHM_asymmetric = 0x00000001, TPMA_ALGORITHM_symmetric = 0x00000002, TPMA_ALGORITHM_hash = 0x00000004, TPMA_ALGORITHM_object = 0x00000008, TPMA_ALGORITHM_signing = 0x00000100, TPMA_ALGORITHM_encrypting = 0x00000200, TPMA_ALGORITHM_method = 0x00000400}
enum	**TPMA_OBJECT_mask** { TPMA_OBJECT_fixedTPM = 0x00000002, TPMA_OBJECT_stClear = 0x00000004, TPMA_OBJECT_fixedParent = 0x00000010, TPMA_OBJECT_sensitiveDataOrigin = 0x00000020, TPMA_OBJECT_userWithAuth = 0x00000040, TPMA_OBJECT_adminWithPolicy = 0x00000080, TPMA_OBJECT_noDA = 0x00000400, TPMA_OBJECT_encryptedDuplication = 0x00000800, TPMA_OBJECT_restricted = 0x00010000, TPMA_OBJECT_decrypt = 0x00020000, TPMA_OBJECT_sign = 0x00040000, TPMA_OBJECT_derivedDataOrigin = 0x00000200, TPMA_OBJECT_firmwareLimited = 0x00000100, TPMA_OBJECT_svnLimited = 0x00000200, TPMA_OBJECT_x509sign = 0x00080000}
enum	**TPMA_SESSION_mask** { TPMA_SESSION_continueSession = 0x01, TPMA_SESSION_auditExclusive = 0x02, TPMA_SESSION_auditReset = 0x04, TPMA_SESSION_decrypt = 0x20, TPMA_SESSION_encrypt = 0x40, TPMA_SESSION_audit = 0x80}

	Name
enum	**TPMA_LOCALITY_mask** { TPM_LOC_ZERO = 0x01, TPM_LOC_ONE = 0x02, TPM_LOC_TWO = 0x04, TPM_LOC_THREE = 0x08, TPM_LOC_FOUR = 0x10}
enum	**TPMA_PERMANENT_mask** { TPMA_PERMANENT_ownerAuthSet = 0x00000001, TPMA_PERMANENT_endorsementAuthSet = 0x00000002, TPMA_PERMANENT_lockoutAuthSet = 0x00000004, TPMA_PERMANENT_disableClear = 0x00000100, TPMA_PERMANENT_inLockout = 0x00000200, TPMA_PERMANENT_tpmGeneratedEPS = 0x00000400}
enum	**TPMA_MEMORY_mask** { TPMA_MEMORY_sharedRAM = 0x00000001, TPMA_MEMORY_sharedNV = 0x00000002, TPMA_MEMORY_objectCopiedToRam = 0x00000004}
enum	**TPMA_CC_mask** { TPMA_CC_commandIndex = 0x0000FFFF, TPMA_CC_nv = 0x00400000, TPMA_CC_extensive = 0x00800000, TPMA_CC_flushed = 0x01000000, TPMA_CC_cHandles = 0x0E000000, TPMA_CC_rHandle = 0x10000000, TPMA_CC_V = 0x20000000}
enum	**TPMA_MODES_mask** { TPMA_MODES_FIPS_140_2 = 0x00000001, TPMA_MODES_FIPS_140_3 = 0x00000002}
enum	**TPMA_ML_PARAMETER_SET_mask** { TPMA_ML_PARAMETER_SET_mlKem_512 = 0x00000001, TPMA_ML_PARAMETER_SET_mlKem_768 = 0x00000002, TPMA_ML_PARAMETER_SET_mlKem_1024 = 0x00000004, TPMA_ML_PARAMETER_SET_mlDsa_44 = 0x00000008, TPMA_ML_PARAMETER_SET_mlDsa_65 = 0x00000010, TPMA_ML_PARAMETER_SET_mlDsa_87 = 0x00000020, TPMA_ML_PARAMETER_SET_extMu = 0x00000040}
enum	**TPMA_ACT_T** { TPMA_ACT_signaled = 0x00000001, TPMA_ACT_preserveSignaled = 0x00000002}
enum	**TPM_NT** { TPM_NT_ORDINARY = 0x0, TPM_NT_COUNTER = 0x1, TPM_NT_BITS = 0x2, TPM_NT_EXTEND = 0x4, TPM_NT_PIN_FAIL = 0x8, TPM_NT_PIN_PASS = 0x9}

	Name
enum	**TPM_MODE_Vendor_Mask** { TPMLib_2 = 0x01, TPMFips = 0x02, TPMLowPowerOff = 0x00, TPMLowPowerByRegister = 0x04, TPMLowPowerByGpio = 0x08, TPMLowPowerAuto = 0x0C}
enum	**TPMI_GPIO_NAME_T** { TPM_GPIO_PP = 0x00000000, TPM_GPIO_LP = 0x00000001, TPM_GPIO_C = 0x00000002, TPM_GPIO_D = 0x00000003}
enum	**TPMI_GPIO_MODE_T** { TPM_GPIO_MODE_STANDARD = 0x00000000, TPM_GPIO_MODE_FLOATING = 0x00000001, TPM_GPIO_MODE_PULLUP = 0x00000002, TPM_GPIO_MODE_PULLDOWN = 0x00000003, TPM_GPIO_MODE_OPENDRAIN = 0x00000004, TPM_GPIO_MODE_PUSHPULL = 0x00000005, TPM_GPIO_MODE_UNCONFIG = 0x00000006, TPM_GPIO_MODE_DEFAULT = TPM_GPIO_MODE_PULLDOWN, TPM_GPIO_MODE_MAX = TPM_GPIO_MODE_UNCONFIG, TPM_GPIO_MODE_INPUT_MIN = TPM_GPIO_MODE_FLOATING, TPM_GPIO_MODE_INPUT_MAX = TPM_GPIO_MODE_PULLDOWN, TPM_GPIO_MODE_PUSHPULL = 0x00000005, TPM_GPIO_MODE_OPENDRAIN = 0x00000004, TPM_GPIO_MODE_PULLUP = 0x00000002, TPM_GPIO_MODE_UNCONFIG = 0x00000006, TPM_GPIO_MODE_DEFAULT = TPM_GPIO_MODE_PULLDOWN, TPM_GPIO_MODE_MAX = TPM_GPIO_MODE_UNCONFIG, TPM_GPIO_MODE_INPUT_MIN = TPM_GPIO_MODE_FLOATING, TPM_GPIO_MODE_INPUT_MAX = TPM_GPIO_MODE_PULLDOWN}

	Name
enum	**TPMI_GPIO_MODE_T** { TPM_GPIO_MODE_STANDARD = 0x00000000, TPM_GPIO_MODE_FLOATING = 0x00000001, TPM_GPIO_MODE_PULLUP = 0x00000002, TPM_GPIO_MODE_PULLDOWN = 0x00000003, TPM_GPIO_MODE_OPENDRAIN = 0x00000004, TPM_GPIO_MODE_PUSHPULL = 0x00000005, TPM_GPIO_MODE_UNCONFIG = 0x00000006, TPM_GPIO_MODE_DEFAULT = TPM_GPIO_MODE_PULLDOWN, TPM_GPIO_MODE_MAX = TPM_GPIO_MODE_UNCONFIG, TPM_GPIO_MODE_INPUT_MIN = TPM_GPIO_MODE_FLOATING, TPM_GPIO_MODE_INPUT_MAX = TPM_GPIO_MODE_PULLDOWN, TPM_GPIO_MODE_PUSHPULL = 0x00000005, TPM_GPIO_MODE_OPENDRAIN = 0x00000004, TPM_GPIO_MODE_PULLUP = 0x00000002, TPM_GPIO_MODE_UNCONFIG = 0x00000006, TPM_GPIO_MODE_DEFAULT = TPM_GPIO_MODE_PULLDOWN, TPM_GPIO_MODE_MAX = TPM_GPIO_MODE_UNCONFIG, TPM_GPIO_MODE_INPUT_MIN = TPM_GPIO_MODE_FLOATING, TPM_GPIO_MODE_INPUT_MAX = TPM_GPIO_MODE_PULLDOWN}
enum	**TPM_Vendor_t** { TPM_VENDOR_UNKNOWN = 0, TPM_VENDOR_INFINEON = 0x15d1, TPM_VENDOR_STM = 0x104a, TPM_VENDOR_MCHP = 0x1114, TPM_VENDOR_NUVOTON = 0x1050, TPM_VENDOR_NATIONTECH = 0x1B4E, TPM_VENDOR_SEALSQ = 0x2406}
typedef UINT32	**TPM_MODIFIER_INDICATOR**
typedef UINT32	**TPM_AUTHORIZATION_SIZE**
typedef UINT32	**TPM_PARAMETER_SIZE**
typedef UINT16	**TPM_KEY_SIZE**
typedef UINT16	**TPM_KEY_BITS**
typedef UINT32	**TPM_GENERATED**
typedef UINT16	**TPM_ALG_ID**
typedef UINT16	**TPM_ECC_CURVE**
typedef UINT16	**TPMI_MLKEM_PARAMETER_SET**
typedef UINT16	**TPMI_MLDSA_PARAMETER_SET**
typedef UINT32	**TPM_CC**
typedef INT32	**TPM_RC**
typedef UINT8	**TPM_CLOCK_ADJUST**
typedef UINT16	**TPM_EO**
typedef UINT16	**TPM_ST**
typedef UINT8	**TPM_SE**
typedef UINT16	**TPM_SU**

	Name
typedef UINT32	**TPM_CAP**
typedef UINT32	**TPM_PT_CONFIG**
typedef UINT32	**TPM_PT**
typedef UINT32	**TPM_PT_PCR**
typedef UINT32	**TPM_PS**
typedef UINT32	**TPM_HANDLE**
typedef UINT8	**TPM_HT**
typedef UINT32	**TPM_RH**
typedef UINT32	**TPM_HC**
typedef UINT32	**TPMA_ALGORITHM**
typedef UINT32	**TPMA_OBJECT**
typedef BYTE	**TPMA_SESSION**
typedef BYTE	**TPMA_LOCALITY**
typedef UINT32	**TPMA_PERMANENT**
typedef UINT32	**TPMA_STARTUP_CLEAR**
typedef UINT32	**TPMA_MEMORY**
typedef UINT32	**TPMA_CC**
typedef UINT32	**TPMA_MODES**
typedef UINT32	**TPMA_ML_PARAMETER_SET**
typedef BYTE	**TPMI_YES_NO**
typedef TPM_HANDLE**	
typedef TPM_HANDLE**	
typedef TPM_HANDLE**	
typedef TPM_HANDLE**	
typedef TPM_HANDLE**	
typedef TPM_HANDLE**	
typedef TPM_HANDLE**	
typedef TPM_HANDLE**	
typedef TPM_HANDLE**	
typedef TPM_HANDLE**	
typedef TPM_HANDLE**	
typedef TPM_HANDLE**	
typedef TPM_HANDLE**	
typedef TPM_HANDLE**	
typedef TPM_HANDLE**	
typedef TPM_HANDLE**	
typedef TPM_HANDLE**	
typedef TPM_HANDLE**	
typedef TPM_HANDLE**	
typedef TPM_ALG_ID**	
typedef TPM_ALG_ID**	
typedef TPM_ALG_ID**	
typedef TPM_ALG_ID**	
typedef TPM_ALG_ID**	
typedef TPM_ALG_ID**	
typedef TPM_ALG_ID**	
typedef TPM_ALG_ID**	
typedef TPM_ST**	
typedef struct TPMS_ALGORITHM_DESCRIPTION**	

Name
typedef union TPMU_HA**
typedef struct TPMT_HA**
typedef struct TPM2B_DIGEST**
typedef struct TPM2B_DATA**
typedef TPM2B_DIGEST**
typedef TPM2B_DIGEST**
typedef TPM2B_DIGEST**
typedef struct TPM2B_EVENT**
typedef struct TPM2B_MAX_BUFFER**
typedef struct TPM2B_MAX_NV_BUFFER**
typedef TPM2B_DIGEST**
typedef struct TPM2B_IV**
typedef struct TPM2B_SIGNATURE_CTX**
typedef struct TPM2B_SIGNATURE_HINT**
typedef struct TPM2B_KEM_CIPHERTEXT**
typedef struct TPM2B_SHARED_SECRET**
typedef struct TPM2B_PUBLIC_KEY_MLDSA**
typedef struct TPM2B_PRIVATE_KEY_MLDSA**
typedef struct TPM2B_PUBLIC_KEY_MLKEM**
typedef struct TPM2B_PRIVATE_KEY_MLKEM**
typedef struct TPM2B_MLDSA_SIGNATURE**
typedef union TPMU_NAME**
typedef struct TPM2B_NAME**
typedef struct TPMS_PCR_SELECT**
typedef struct TPMS_PCR_SELECTION**
typedef struct TPMT_TK_CREATION**
typedef struct TPMT_TK_VERIFIED**
typedef struct TPMT_TK_AUTH**
typedef struct TPMT_TK_HASHCHECK**
typedef struct TPMS_ALG_PROPERTY**
typedef struct TPMS_TAGGED_PROPERTY**
typedef struct TPMS_TAGGED_PCR_SELECT**
typedef struct TPMS_TAGGED_POLICY**
typedef struct TPMS_TAGGED_CONFIG**
typedef struct TPML_CC**
typedef struct TPML_CCA**
typedef struct TPML_ALG**
typedef struct TPML_HANDLE**
typedef struct TPML_DIGEST**
typedef struct TPML_DIGEST_VALUES**
typedef struct TPML_PCR_SELECTION**
typedef struct TPML_ALG_PROPERTY**
typedef struct TPML_TAGGED_TPM_PROPERTY**
typedef struct TPML_TAGGED_PCR_PROPERTY**
typedef struct TPML_ECC_CURVE**
typedef struct TPML_TAGGED_POLICY**
typedef struct TPML_TAGGED_CONFIG**
typedef UINT32 **TPMA_ACT**
typedef struct TPMS_ACT_DATA**
typedef struct TPML_ACT_DATA**

Name
typedef struct TPMS_SPDM_SESSION_INFO**
typedef struct TPML_SPDM_SESSION_INFO**
typedef union TPMU_CAPABILITIES**
typedef struct TPMS_CAPABILITY_DATA**
typedef struct TPMS_CLOCK_INFO**
typedef struct TPMS_TIME_INFO**
typedef struct TPMS_TIME_ATTEST_INFO**
typedef struct TPMS_CERTIFY_INFO**
typedef struct TPMS_QUOTE_INFO**
typedef struct TPMS_COMMAND_AUDIT_INFO**
typedef struct TPMS_SESSION_AUDIT_INFO**
typedef struct TPMS_CREATION_INFO**
typedef struct TPMS_NV_CERTIFY_INFO**
typedef struct TPMS_NV_DIGEST_CERTIFY_INFO**
typedef TPM_ST**
typedef union TPMU_ATTEST**
typedef struct TPMS_ATTEST**
typedef struct TPM2B_ATTEST**
typedef TPM_KEY_BITS**
typedef union TPMU_SYM_KEY_BITS**
typedef union TPMU_SYM_MODE**
typedef struct TPMT_SYM_DEF**
typedef TPMT_SYM_DEF**
typedef struct TPM2B_SYM_KEY**
typedef struct TPMS_SYMCIPHER_PARMS**
typedef struct TPM2B_LABEL**
typedef struct TPMS_DERIVE**
typedef struct TPM2B_DERIVE**
typedef union TPMU_SENSITIVE_CREATE**
typedef struct TPM2B_SENSITIVE_DATA**
typedef struct TPMS_SENSITIVE_CREATE**
typedef struct TPM2B_SENSITIVE_CREATE**
typedef struct TPMS_SCHEME_HASH**
typedef struct TPMS_SCHEME_ECDAA**
typedef TPM_ALG_ID**
typedef TPMS_SCHEME_HASH**
typedef struct TPMS_SCHEME_XOR**
typedef union TPMU_SCHEME_KEYEDHASH**
typedef struct TPMT_KEYEDHASH_SCHEME**
typedef TPMS_SCHEME_HASH**
typedef TPMS_SCHEME_HASH**
typedef TPMS_SCHEME_HASH**
typedef TPMS_SCHEME_ECDAA**
typedef union TPMU_SIG_SCHEME**
typedef struct TPMT_SIG_SCHEME**
typedef TPMS_SCHEME_HASH**
typedef TPMS_SCHEME_HASH**
typedef TPMS_SCHEME_HASH**
typedef TPMS_SCHEME_HASH**
typedef TPMS_SCHEME_HASH**

Name
typedef TPMS_SCHEME_HASH**
typedef TPMS_SCHEME_HASH**
typedef union TPMU_KDF_SCHEME**
typedef struct TPMT_KDF_SCHEME**
typedef TPM_ALG_ID**
typedef union TPMU_ASYM_SCHEME**
typedef struct TPMT_ASYM_SCHEME**
typedef TPM_ALG_ID**
typedef struct TPMT_RSA_SCHEME**
typedef TPM_ALG_ID**
typedef struct TPMT_RSA_DECRYPT**
typedef struct TPM2B_PUBLIC_KEY_RSA**
typedef TPM_KEY_BITS**
typedef struct TPM2B_PRIVATE_KEY_RSA**
typedef struct TPM2B_ECC_PARAMETER**
typedef struct TPMS_ECC_POINT**
typedef struct TPM2B_ECC_POINT**
typedef TPM_ALG_ID**
typedef TPM_ECC_CURVE**
typedef TPMT_SIG_SCHEME**
typedef struct TPMS_ALGORITHM_DETAIL_ECC**
typedef struct TPMS_SIGNATURE_RSA**
typedef TPMS_SIGNATURE_RSA**
typedef TPMS_SIGNATURE_RSA**
typedef struct TPMS_SIGNATURE_ECC**
typedef TPMS_SIGNATURE_ECC**
typedef TPMS_SIGNATURE_ECC**
typedef struct TPMS_SIGNATURE_HASH_MLDSA**
typedef union TPMU_SIGNATURE**
typedef struct TPMT_SIGNATURE**
typedef union TPMU_ENCRYPTED_SECRET**
typedef struct TPM2B_ENCRYPTED_SECRET**
typedef TPM_ALG_ID**
typedef union TPMU_PUBLIC_ID**
typedef struct TPMS_KEYEDHASH_PARMS**
typedef struct TPMS_ASYM_PARMS**
typedef struct TPMS_RSA_PARMS**
typedef struct TPMS_ECC_PARMS**
typedef struct TPMS_MLDSA_PARMS**
typedef struct TPMS_HASH_MLDSA_PARMS**
typedef struct TPMS_MLKEM_PARMS**
typedef union TPMU_PUBLIC_PARMS**
typedef struct TPMT_PUBLIC_PARMS**
typedef struct TPMT_PUBLIC**
typedef struct TPM2B_PUBLIC**
typedef struct TPM2B_TEMPLATE**
typedef struct TPM2B_PRIVATE_VENDOR_SPECIFIC**
typedef union TPMU_SENSITIVE_COMPOSITE**
typedef struct TPMT_SENSITIVE**

Name
typedef struct TPM2B_SENSITIVE**
typedef struct TPMT_PRIVATE**
typedef struct TPM2B_PRIVATE**
typedef struct TPMS_ID_OBJECT**
typedef struct TPM2B_ID_OBJECT**
typedef UINT32 **TPM_NV_INDEX**
typedef enum TPM_NT**
typedef struct TPMS_NV_PIN_COUNTER_PARAMETERS**
typedef UINT32 **TPMA_NV**
typedef struct TPMS_NV_PUBLIC**
typedef struct TPM2B_NV_PUBLIC**
typedef struct TPM2B_CONTEXT_SENSITIVE**
typedef struct TPMS_CONTEXT_DATA**
typedef struct TPM2B_CONTEXT_DATA**
typedef struct TPMS_CONTEXT**
typedef struct TPMS_CREATION_DATA**
typedef struct TPM2B_CREATION_DATA**
typedef struct TPMS_AUTH_COMMAND**
typedef struct TPMS_AUTH_RESPONSE**
typedef struct TPM2_AUTH_SESSION**
typedef int()*(struct TPM2_CTX* , INT32 isRead, UINT32 addr, BYTE *xferBuf, UINT16 xferSz, void* userCtx) **TPM2HalIoCb**
typedef struct TPM2_CTX**
typedef ChangeSeed_In**
typedef ChangeSeed_In**
typedef struct TPM_MODE_SET**
typedef GetRandom_In**
typedef UINT32 **TPMI_GPIO_NAME**
typedef UINT32 **TPMI_GPIO_MODE**
typedef struct TPMS_GPIO_CONFIG**
typedef struct TPML_GPIO_CONFIG**

29.8.3 Functions

Name
WOLFTPM_API TPM_RC**(Startup_In * in)
WOLFTPM_API TPM_RC**(Shutdown_In * in)
WOLFTPM_API TPM_RC**(GetCapability_In * out)
WOLFTPM_API TPM_RC**(SetCapability_In * in)
WOLFTPM_API TPM_RC**(SelfTest_In * in)
WOLFTPM_API TPM_RC**(IncrementalSelfTest_In * out)
WOLFTPM_API TPM_RC**(GetTestResult_Out * out)
WOLFTPM_API TPM_RC**(GetRandom_In * out)
WOLFTPM_API TPM_RC**(StirRandom_In * in)
WOLFTPM_API TPM_RC**(PCR_Read_In * out)

Name
WOLFTPM_API TPM_RC**(PCR_Extend_In * in)
WOLFTPM_API TPM_RC**(Create_In * out)
WOLFTPM_API TPM_RC**(CreateLoaded_In * out)
WOLFTPM_API TPM_RC**(CreatePrimary_In * out)
WOLFTPM_API TPM_RC**(Load_In * out)
WOLFTPM_API TPM_RC**(FlushContext_In * in)
WOLFTPM_API TPM_RC**(Unseal_In * out)
WOLFTPM_API TPM_RC**(StartAuthSession_In * out)
WOLFTPM_API TPM_RC**(PolicyRestart_In * in)
WOLFTPM_API TPM_RC**(LoadExternal_In * out)
WOLFTPM_API TPM_RC**(ReadPublic_In * out)
WOLFTPM_API TPM_RC**(ActivateCredential_In * out)
WOLFTPM_API TPM_RC**(MakeCredential_In * out)
WOLFTPM_API TPM_RC**(ObjectChangeAuth_In * out)
WOLFTPM_API TPM_RC**(Duplicate_In * out)
WOLFTPM_API TPM_RC**(Rewrap_In * out)
WOLFTPM_API TPM_RC**(Import_In * out)
WOLFTPM_API TPM_RC**(RSA_Encrypt_In * out)
WOLFTPM_API TPM_RC**(RSA_Decrypt_In * out)
WOLFTPM_API TPM_RC**(ECDH_KeyGen_In * out)
WOLFTPM_API TPM_RC**(ECDH_ZGen_In * out)
WOLFTPM_API TPM_RC**(ECC_Parameters_In * out)
WOLFTPM_API TPM_RC**(ZGen_2Phase_In * out)
WOLFTPM_API TPM_RC**(EncryptDecrypt_In * out)
WOLFTPM_API TPM_RC**(EncryptDecrypt2_In * out)
WOLFTPM_API TPM_RC**(Hash_In * out)
WOLFTPM_API TPM_RC**(HMAC_In * out)
WOLFTPM_API TPM_RC**(HMAC_Start_In * out)
WOLFTPM_API TPM_RC**(HashSequenceStart_In * out)
WOLFTPM_API TPM_RC**(SequenceUpdate_In * in)
WOLFTPM_API TPM_RC**(SequenceComplete_In * out)
WOLFTPM_API TPM_RC**(EventSequenceComplete_In * out)
WOLFTPM_API TPM_RC**(Certify_In * out)

Name	
WOLFTPM_API TPM_RC**(CertifyCreation_In * out)	
WOLFTPM_API TPM_RC**(Quote_In * out)	
WOLFTPM_API TPM_RC**(GetSessionAuditDigest_In * out)	
WOLFTPM_API TPM_RC**(GetCommandAuditDigest_In * out)	
WOLFTPM_API TPM_RC**(GetTime_In * out)	
WOLFTPM_API TPM_RC**(Commit_In * out)	
WOLFTPM_API TPM_RC**(EC_Ephemeral_In * out)	
WOLFTPM_API TPM_RC**(VerifySignature_In * out)	
WOLFTPM_API TPM_RC**(Sign_In * out)	
WOLFTPM_API TPM_RC**(SignSequenceStart_In * out)	
WOLFTPM_API TPM_RC**(VerifySequenceStart_In * out)	
WOLFTPM_API TPM_RC**(SignSequenceComplete_In * out)	
WOLFTPM_API TPM_RC**(VerifySequenceComplete_In * out)	
WOLFTPM_API TPM_RC**(SignDigest_In * out)	
WOLFTPM_API TPM_RC**(VerifyDigestSignature_In * out)	
WOLFTPM_API TPM_RC**(Encapsulate_In * out)	
WOLFTPM_API TPM_RC**(Decapsulate_In * out)	
WOLFTPM_API TPM_RC**(SetCommandCodeAuditStatus_In * in)	
WOLFTPM_API TPM_RC**(PCR_Event_In * out)	
WOLFTPM_API TPM_RC **TPM2_PCR_Allocate * out)Set which PCR banks the TPM allocates.	
WOLFTPM_API TPM_RC**(PCR_SetAuthPolicy_In * in)	
WOLFTPM_API TPM_RC**(PCR_SetAuthValue_In * in)	
WOLFTPM_API TPM_RC**(PCR_Reset_In * in)	
WOLFTPM_API TPM_RC**(PolicySigned_In * out)	
WOLFTPM_API TPM_RC**(PolicySecret_In * out)	
WOLFTPM_API TPM_RC**(PolicyTicket_In * in)	
WOLFTPM_API TPM_RC**(PolicyOR_In * in)	
WOLFTPM_API TPM_RC**(PolicyPCR_In * in)	
WOLFTPM_API TPM_RC**(PolicyTransportSPDM_In * in)	
WOLFTPM_API TPM_RC**(PolicyLocality_In * in)	
WOLFTPM_API TPM_RC**(PolicyNV_In * in)	
WOLFTPM_API TPM_RC**(PolicyCounterTimer_In * in)	

Name
WOLFTPM_API TPM_RC**(PolicyCommandCode_In * in)
WOLFTPM_API TPM_RC**(PolicyPhysicalPresence_In * in)
WOLFTPM_API TPM_RC**(PolicyCpHash_In * in)
WOLFTPM_API TPM_RC**(PolicyNameHash_In * in)
WOLFTPM_API TPM_RC**(PolicyDuplicationSelect_In * in)
WOLFTPM_API TPM_RC**(PolicyAuthorize_In * in)
WOLFTPM_API TPM_RC**(PolicyAuthValue_In * in)
WOLFTPM_API TPM_RC**(PolicyPassword_In * in)
WOLFTPM_API TPM_RC**(PolicyGetDigest_In * out)
WOLFTPM_API TPM_RC**(PolicyNvWritten_In * in)
WOLFTPM_API TPM_RC**(PolicyTemplate_In * in)
WOLFTPM_API TPM_RC**(PolicyAuthorizeNV_In * in)
WOLFTPM_API void **_TPM_Hash_Start**(void)
WOLFTPM_API void **_TPM_Hash_Data**(UINT32 dataSize, BYTE * data)
WOLFTPM_API void **_TPM_Hash_End**(void)
WOLFTPM_API TPM_RC**(HierarchyControl_In * in)
WOLFTPM_API TPM_RC**(SetPrimaryPolicy_In * in)
WOLFTPM_API TPM_RC * in)
WOLFTPM_API TPM_RC * in)
WOLFTPM_API TPM_RC**(Clear_In * in)
WOLFTPM_API TPM_RC**(ClearControl_In * in)
WOLFTPM_API TPM_RC**(HierarchyChangeAuth_In * in)
WOLFTPM_API TPM_RC**(DictionaryAttackLockReset_In * in)
WOLFTPM_API TPM_RC**(DictionaryAttackParameters_In * in)
WOLFTPM_API TPM_RC**(PP_Commands_In * in)
WOLFTPM_API TPM_RC**(SetAlgorithmSet_In * in)
WOLFTPM_API TPM_RC**(FieldUpgradeStart_In * in)
WOLFTPM_API TPM_RC**(FieldUpgradeData_In * out)
WOLFTPM_API TPM_RC**(FirmwareRead_In * out)

Name
WOLFTPM_API TPM_RC**(ContextSave_In * out)
WOLFTPM_API TPM_RC**(ContextLoad_In * out)
WOLFTPM_API TPM_RC**(EvictControl_In * in)
WOLFTPM_API TPM_RC**(ReadClock_Out * out)
WOLFTPM_API TPM_RC**(ClockSet_In * in)
WOLFTPM_API TPM_RC**(ClockRateAdjust_In * in)
WOLFTPM_API TPM_RC**(TestParms_In * in)
WOLFTPM_API TPM_RC**(NV_DefineSpace_In * in)
WOLFTPM_API TPM_RC**(NV_UndefineSpace_In * in)
WOLFTPM_API TPM_RC**(NV_UndefineSpaceSpecial_In * in)
WOLFTPM_API TPM_RC**(NV_ReadPublic_In * out)
WOLFTPM_API TPM_RC**(NV_Write_In * in)
WOLFTPM_API TPM_RC**(NV_Increment_In * in)
WOLFTPM_API TPM_RC**(NV_Extend_In * in)
WOLFTPM_API TPM_RC**(NV_SetBits_In * in)
WOLFTPM_API TPM_RC**(NV_WriteLock_In * in)
WOLFTPM_API TPM_RC**(NV_GlobalWriteLock_In * in)
WOLFTPM_API TPM_RC**(NV_Read_In * out)
WOLFTPM_API TPM_RC**(NV_ReadLock_In * in)
WOLFTPM_API TPM_RC**(NV_ChangeAuth_In * in)
WOLFTPM_API TPM_RC**(NV_Certify_In * out)
WOLFTPM_API int **TPM2_SetCommandSet**(SetCommandSet_In * in)
WOLFTPM_API int **TPM2_SetMode**(SetMode_In * in)
WOLFTPM_API TPM_RC * in, GetRandom2_Out * out)
WOLFTPM_API TPM_RC**(uint8_t * info, uint16_t size)
WOLFTPM_API int **TPM2_IFX_FieldUpgradeStart sessionHandle, uint8_t * data, uint32_t size)
WOLFTPM_API int **TPM2_IFX_FieldUpgradeCommand cc, uint8_t * data, uint32_t size)
WOLFTPM_API int **TPM2_ST33_FieldUpgradeStart sessionHandle, uint8_t * data, uint32_t size)
WOLFTPM_API int **TPM2_ST33_FieldUpgradeStart_ex cc, uint8_t * data, uint32_t size)
WOLFTPM_API int **TPM2_ST33_FieldUpgradeCommand cc, uint8_t * data, uint32_t size)
WOLFTPM_API int **TPM2_GPIO_Config**(GpioConfig_In * in)
WOLFTPM_API int **TPM2_NTC2_PreConfig**(NTC2_PreConfig_In * in)

	Name
WOLFTPM_API int	**TPM2_NTC2_GetConfig**(NTC2_GetConfig_Out * out)
WOLFTPM_API int	**TPM2_Nations_IdentityKeySet**(Nations_IdentityKeySet_In * in)
WOLFTPM_API TPM_RC ioCb, void * userCtx)Initializes a TPM with HAL IO callback and user supplied context. When using wolfTPM with –enable-devtpm or –enable-swtpm configuration, the ioCb and userCtx are not used.	
WOLFTPM_API TPM_RC ioCb, void * userCtx, int timeoutTries)Initializes a TPM with timeoutTries, HAL IO callback and user supplied context.	
WOLFTPM_API TPM_RC	**TPM2_Init_minimal * ctx)Initializes a TPM and sets the wolfTPM2 context that will be used. This function is typically used for rich operating systems, like Windows.
WOLFTPM_API TPM_RC	**TPM2_Cleanup * ctx)Deinitializes a TPM and wolfcrypt (if it was initialized)
WOLFTPM_API TPM_RC	**TPM2_ChipStartup * ctx, int timeoutTries)Makes sure the TPM2 startup has completed and extracts the TPM device information.
WOLFTPM_API TPM_RC ioCb, void * userCtx)Sets the user's context and IO callbacks needed for TPM communication.	
WOLFTPM_API TPM_RC	**TPM2_SetCommandRetries * ctx, int retries)Sets the number of times a command is transparently resubmitted on TPM_RC_RETRY.
WOLFTPM_API int	**TPM2_GetCommandRetries * ctx)Returns the number of times a command is transparently resubmitted on TPM_RC_RETRY.
WOLFTPM_API TPM_RC	**TPM2_SetSessionAuth * session)Sets the structure holding the TPM Authorizations.
WOLFTPM_API int	**TPM2_GetSessionAuthCount * ctx)Determine the number of currently set TPM Authorizations.
WOLFTPM_API void	**TPM2_SetActiveCtx * ctx)Sets a new TPM2 context for use.
WOLFTPM_API TPM2_CTX**(void)Provides a pointer to the TPM2 context in use.	
WOLFTPM_API int	**TPM2_GetHashDigestSize hashAlg)Determine the size in bytes of a TPM 2.0 hash digest.
WOLFTPM_API int	**TPM2_GetHashType hashAlg)Translate a TPM2 hash type to its corresponding wolfcrypt hash type.
WOLFTPM_API TPMI_ALG_HASH	**TPM2_GetTpmHashType**(int hashType)Translate a wolfCrypt hash type to TPM2 hash type.

	Name
WOLFTPM_API int	**TPM2_GetNonce**(byte * nonceBuf, int nonceSz)Generate a fresh nonce of random numbers.
WOLFTPM_LOCAL int	**TPM2_GetNonceNoLock**(byte * nonceBuf, int nonceSz)
WOLFTPM_API void	**TPM2_SetupPCRSel alg, int pcrIndex)Helper function to prepare a correct PCR selection For example, when preparing to create a TPM2_Quote.
WOLFTPM_API void	**TPM2_SetupPCRSelArray alg, byte * pcrArray, word32 pcrArraySz)Helper function to prepare a correct PCR selection with multiple indices For example, when preparing to create a TPM2_Quote.
WOLFTPM_API int	**TPM2_IsAlgSupported alg, int * isSupported)Report whether the TPM implements a given algorithm.
WOLFTPM_API int	**TPM2_IsPcrBankAllocated hashAlg, int pcrIndex, int * isAllocated)Report whether a PCR index is allocated in a bank of a given hash.
WOLFTPM_API const char *	**TPM2_GetRCString**(int rc)Get a human readable string for any TPM 2.0 return code.
WOLFTPM_API const char *	**TPM2_GetAlgName alg)Get a human readable string for any TPM 2.0 algorithm.
WOLFTPM_API TPM_ALG_ID	**TPM2_GetAlgId**(const char * name)Translates a TPM algorithm name to its TPM algorithm ID.
WOLFTPM_API const char *	**TPM2_GetHierarchyDesc authHandle)Get readable string for TPM 2.0 hierarchy.
WOLFTPM_API int	**TPM2_GetCurveSize curveID)Determine the size in bytes of any TPM ECC Curve.
WOLFTPM_API TPM_ALG_ID curveID)Determine the recommended message digest for a TPM ECC Curve.	
WOLFTPM_API int	**TPM2_GetTpmCurve**(int curveID)Translate a wolfcrypt curve type to its corresponding TPM curve type.
WOLFTPM_API int	**TPM2_GetWolfCurve**(int curve_id)Translate a TPM curve type to its corresponding wolfcrypt curve type.
WOLFTPM_API int	**TPM2_ParseAttest structure.
WOLFTPM_API int	**TPM2_HashNvPublic * nvPublic, byte * buffer, UINT16 * size)Computes fresh NV Index name based on a nvPublic structure.
WOLFTPM_API int	**TPM2_AppendPublic structure based on a user provided buffer.
WOLFTPM_API int	**TPM2_ParsePublic structure and stores in a user provided buffer.
WOLFTPM_LOCAL int	**TPM2_GetName * name)Provides the Name of a TPM object.
WOLFTPM_API int	**TPM2_GetWolfRng**(WC_RNG ** rng)
WOLFTPM_API UINT16	**TPM2_GetVendorID**(void)Provides the vendorID of the active TPM2 context.

	Name
WOLFTPM_API const char *	**TPM2_GetCapsFipsStr**(int fips140_3, int fips140_2)Returns a human-readable FIPS mode string from FIPS capability bits: “140-3” when fips140_3 is set, “140-2” when fips140_2 is set, otherwise “none”. Used for capability reporting in examples and tools.
WOLFTPM_LOCAL void	**TPM2_ForceZero**(void * mem, word32 len)Forcefully zero a memory region. Unlike memset, this call is not subject to dead-store elimination, so it is safe to use for erasing sensitive material (keys, intermediate HMAC state, etc.) just before the memory goes out of scope.
WOLFTPM_TEST_API int	**TPM2_ConstantCompare**(const byte * a, const byte * b, word32 len)Constant-time memory comparison. Returns 0 on byte-for-byte match and non-zero on mismatch. Timing is independent of where the first differing byte is located, which is required when comparing HMAC digests or authentication values.
WOLFTPM_API void	**TPM2_PrintBin**(const byte * buffer, word32 length)Helper function to print a binary buffer in a formatted way.
WOLFTPM_API void	**TPM2_PrintAuth type in a human readable way.
WOLFTPM_API void	**TPM2_PrintPublicArea type in a human readable way.

29.8.4 Attributes

	Name
	C
const BYTE[]	**TPM_20_EK_AUTH_POLICY**
const BYTE[]	**TPM_20_EK_AUTH_POLICY_SHA256**
const BYTE[]	**TPM_20_EK_AUTH_POLICY_SHA384**
const BYTE[]	**TPM_20_EK_AUTH_POLICY_SHA512**
const BYTE[]	**TPM_20_IDEVID_POLICY**
const BYTE[]	**TPM_20_IAK_POLICY**
const BYTE[]	**TPM_20_IDEVID_POLICY_SHA384**
const BYTE[]	**TPM_20_IAK_POLICY_SHA384**
const BYTE[]	**TPM_20_IDEVID_POLICY_SHA512**
const BYTE[]	**TPM_20_IAK_POLICY_SHA512**

29.8.5 Types Documentation

29.8.5.1 enum TPM_ALG_ID_T

Enumerator	Value	Description
TPM_ALG_ERROR	0x0000	
TPM_ALG_FIRST	0x0001	
TPM_ALG_RSA	TPM_ALG_FIRST	
TPM_ALG_SHA	0x0004	
TPM_ALG_SHA1	TPM_ALG_SHA	
TPM_ALG_HMAC	0x0005	
TPM_ALG_AES	0x0006	
TPM_ALG_MGF1	0x0007	
TPM_ALG_KEYEDHASH	0x0008	
TPM_ALG_XOR	0x000A	
TPM_ALG_SHA256	0x000B	
TPM_ALG_SHA384	0x000C	
TPM_ALG_SHA512	0x000D	
TPM_ALG_NULL	0x0010	
TPM_ALG_SM3_256	0x0012	
TPM_ALG_SM4	0x0013	
TPM_ALG_RSASSA	0x0014	
TPM_ALG_RSAES	0x0015	
TPM_ALG_RSAPSS	0x0016	
TPM_ALG_OAEP	0x0017	
TPM_ALG_ECDSA	0x0018	
TPM_ALG_ECDH	0x0019	
TPM_ALG_ECDAA	0x001A	
TPM_ALG_SM2	0x001B	
TPM_ALG_ECSCHNORR	0x001C	
TPM_ALG_ECMQV	0x001D	
TPM_ALG_HKDF	0x001F	
TPM_ALG_KDF1_SP800_56A	0x0020	
TPM_ALG_KDF2	0x0021	
TPM_ALG_KDF1_SP800_108	0x0022	
TPM_ALG_ECC	0x0023	
TPM_ALG_SYMCIPHER	0x0025	
TPM_ALG_CAMELLIA	0x0026	
TPM_ALG_SHA3_256	0x0027	
TPM_ALG_SHA3_384	0x0028	
TPM_ALG_SHA3_512	0x0029	
TPM_ALG_SHAKE128	0x002A	
TPM_ALG_SHAKE256	0x002B	
TPM_ALG_CTR	0x0040	
TPM_ALG_OFB	0x0041	
TPM_ALG_CBC	0x0042	
TPM_ALG_CFB	0x0043	
TPM_ALG_ECB	0x0044	
TPM_ALG_MLKEM	0x00A0	
TPM_ALG_MLDSA	0x00A1	
TPM_ALG_HASH_MLDSA	0x00A2	

29.8.5.2 enum TPM_ECC_CURVE_T

Enumerator	Value	Description
TPM_ECC_NONE	0x0000	
TPM_ECC_NIST_P192	0x0001	
TPM_ECC_NIST_P224	0x0002	
TPM_ECC_NIST_P256	0x0003	
TPM_ECC_NIST_P384	0x0004	
TPM_ECC_NIST_P521	0x0005	
TPM_ECC_BN_P256	0x0010	
TPM_ECC_BN_P638	0x0011	
TPM_ECC_SM2_P256	0x0020	
TPM_ECC_BP_P256_R1	0x0030	
TPM_ECC_BP_P384_R1	0x0031	
TPM_ECC_BP_P512_R1	0x0032	

29.8.5.3 enum TPM_CC_T

Enumerator	Value	Description
TPM_CC_FIRST	0x0000011F	
TPM_CC_NV_UndefineSpaceSpecial	TPM_CC_FIRST	
TPM_CC_EvictControl	0x00000120	
TPM_CC_HierarchyControl	0x00000121	
TPM_CC_NV_UndefineSpace	0x00000122	
TPM_CC_ChangeEPS	0x00000124	
TPM_CC_ChangePPS	0x00000125	
TPM_CC_Clear	0x00000126	
TPM_CC_ClearControl	0x00000127	
TPM_CC_ClockSet	0x00000128	
TPM_CC_HierarchyChangeAuth	0x00000129	
TPM_CC_NV_DefineSpace	0x0000012A	
TPM_CC_PCR_Allocate	0x0000012B	
TPM_CC_PCR_SetAuthPolicy	0x0000012C	
TPM_CC_PP_Commands	0x0000012D	
TPM_CC_SetPrimaryPolicy	0x0000012E	
TPM_CC_FieldUpgradeStart	0x0000012F	
TPM_CC_ClockRateAdjust	0x00000130	
TPM_CC_CreatePrimary	0x00000131	
TPM_CC_NV_GlobalWriteLock	0x00000132	
TPM_CC_GetCommandAuditDigest	0x00000133	
TPM_CC_NV_Increment	0x00000134	
TPM_CC_NV_SetBits	0x00000135	
TPM_CC_NV_Extend	0x00000136	
TPM_CC_NV_Write	0x00000137	
TPM_CC_NV_WriteLock	0x00000138	
TPM_CC_DictionaryAttackLockReset	0x00000139	
TPM_CC_DictionaryAttackParameters	0x0000013A	
TPM_CC_NV_ChangeAuth	0x0000013B	
TPM_CC_PCR_Event	0x0000013C	
TPM_CC_PCR_Reset	0x0000013D	
TPM_CC_SequenceComplete	0x0000013E	
TPM_CC_SetAlgorithmSet	0x0000013F	
TPM_CC_SetCommandCodeAuditStatus	0x00000140	

Enumerator	Value	Description
TPM_CC_FieldUpgradeData	0x00000141	
TPM_CC_IncrementalSelfTest	0x00000142	
TPM_CC_SelfTest	0x00000143	
TPM_CC_Startup	0x00000144	
TPM_CC_Shutdown	0x00000145	
TPM_CC_StirRandom	0x00000146	
TPM_CC_ActivateCredential	0x00000147	
TPM_CC_Certify	0x00000148	
TPM_CC_PolicyNV	0x00000149	
TPM_CC_CertifyCreation	0x0000014A	
TPM_CC_Duplicate	0x0000014B	
TPM_CC_GetTime	0x0000014C	
TPM_CC_GetSessionAuditDigest	0x0000014D	
TPM_CC_NV_Read	0x0000014E	
TPM_CC_NV_ReadLock	0x0000014F	
TPM_CC_ObjectChangeAuth	0x00000150	
TPM_CC_PolicySecret	0x00000151	
TPM_CC_Rewrap	0x00000152	
TPM_CC_Create	0x00000153	
TPM_CC_ECDH_ZGen	0x00000154	
TPM_CC_HMAC	0x00000155	
TPM_CC_Import	0x00000156	
TPM_CC_Load	0x00000157	
TPM_CC_Quote	0x00000158	
TPM_CC_RSA_Decrypt	0x00000159	
TPM_CC_HMAC_Start	0x0000015B	
TPM_CC_SequenceUpdate	0x0000015C	
TPM_CC_Sign	0x0000015D	
TPM_CC_Unseal	0x0000015E	
TPM_CC_PolicySigned	0x00000160	
TPM_CC_ContextLoad	0x00000161	
TPM_CC_ContextSave	0x00000162	
TPM_CC_ECDH_KeyGen	0x00000163	
TPM_CC_EncryptDecrypt	0x00000164	
TPM_CC_FlushContext	0x00000165	
TPM_CC_LoadExternal	0x00000167	
TPM_CC_MakeCredential	0x00000168	
TPM_CC_NV_ReadPublic	0x00000169	
TPM_CC_PolicyAuthorize	0x0000016A	
TPM_CC_PolicyAuthValue	0x0000016B	
TPM_CC_PolicyCommandCode	0x0000016C	
TPM_CC_PolicyCounterTimer	0x0000016D	
TPM_CC_PolicyCpHash	0x0000016E	
TPM_CC_PolicyLocality	0x0000016F	
TPM_CC_PolicyNameHash	0x00000170	
TPM_CC_PolicyOR	0x00000171	
TPM_CC_PolicyTicket	0x00000172	
TPM_CC_ReadPublic	0x00000173	
TPM_CC_RSA_Encrypt	0x00000174	
TPM_CC_StartAuthSession	0x00000176	
TPM_CC_VerifySignature	0x00000177	
TPM_CC_ECC_Parameters	0x00000178	

COPYRIGHT ©2026 wolfSSL Inc.

Enumerator	Value	Description
TPM_CC_FirmwareRead	0x00000179	
TPM_CC_GetCapability	0x0000017A	
TPM_CC_GetRandom	0x0000017B	
TPM_CC_GetTestResult	0x0000017C	
TPM_CC_Hash	0x0000017D	
TPM_CC_PCR_Read	0x0000017E	
TPM_CC_PolicyPCR	0x0000017F	
TPM_CC_PolicyRestart	0x00000180	
TPM_CC_ReadClock	0x00000181	
TPM_CC_PCR_Extend	0x00000182	
TPM_CC_PCR_SetAuthValue	0x00000183	
TPM_CC_NV_Certify	0x00000184	
TPM_CC_EventSequenceComplete	0x00000185	
TPM_CC_HashSequenceStart	0x00000186	
TPM_CC_PolicyPhysicalPresence	0x00000187	
TPM_CC_PolicyDuplicationSelect	0x00000188	
TPM_CC_PolicyGetDigest	0x00000189	
TPM_CC_TestParms	0x0000018A	
TPM_CC_Commit	0x0000018B	
TPM_CC_PolicyPassword	0x0000018C	
TPM_CC_ZGen_2Phase	0x0000018D	
TPM_CC_EC_Ephemeral	0x0000018E	
TPM_CC_PolicyNvWritten	0x0000018F	
TPM_CC_PolicyTemplate	0x00000190	
TPM_CC_CreateLoaded	0x00000191	
TPM_CC_PolicyAuthorizeNV	0x00000192	
TPM_CC_EncryptDecrypt2	0x00000193	
TPM_CC_SetCapability	0x0000019F	
TPM_CC_PolicyTransportSPDM	0x000001A1	
TPM_CC_VerifySequenceComplete	0x000001A3	
TPM_CC_SignSequenceComplete	0x000001A4	
TPM_CC_VerifyDigestSignature	0x000001A5	
TPM_CC_SignDigest	0x000001A6	
TPM_CC_Encapsulate	0x000001A7	
TPM_CC_Decapsulate	0x000001A8	
TPM_CC_VerifySequenceStart	0x000001A9	
TPM_CC_SignSequenceStart	0x000001AA	
TPM_CC_LAST	TPM_CC_SignSequenceStart	
TPM_CC_LAST	TPM_CC_SignSequenceStart	
TPM_CC_LAST	TPM_CC_SignSequenceStart	
CC_VEND	0x20000000	
TPM_CC_Vendor_TCG_Test	CC_VEND + 0x0000	
TPM_CC_SetMode	CC_VEND + 0x0307	
TPM_CC_SetCommandSet	CC_VEND + 0x0309	
TPM_CC_GetRandom2	CC_VEND + 0x030E	
TPM_CC_RestoreEK	CC_VEND + 0x030A	
TPM_CC_SetCommandSetLock	CC_VEND + 0x030B	
TPM_CC_GPIO_Config	CC_VEND + 0x030F	
TPM_CC_NTC2_PreConfig	CC_VEND + 0x0211	
TPM_CC_NTC2_GetConfig	CC_VEND + 0x0213	
TPM_CC_Nations_IdentityKeySet	0x20000708	
TPM_CC_FieldUpgradeStartVendor	CC_VEND + 0x12F	

COPYRIGHT ©2026 wolfSSL Inc.

Enumerator	Value	Description
TPM_CC_FieldUpgradeAbandonVendor	CC_VEND + 0x130	
TPM_CC_FieldUpgradeManifestVendor	CC_VEND + 0x131	
TPM_CC_FieldUpgradeDataVendor	CC_VEND + 0x132	
TPM_CC_FieldUpgradeFinalizeVendor	CC_VEND + 0x133	
TPM_CC_FieldUpgradeStartVendor_ST33	CC_VEND + 0x030C	
TPM_CC_FieldUpgradeAbandonVendor_ST33	CC_VEND + 0x030E	
TPM_CC_FieldUpgradeDataVendor_ST33	CC_VEND + 0x030D	
TPM_CC_FieldUpgradeFinalizeVendor_ST33	CC_VEND + 0x030F	

29.8.5.4 enum TPM_RC_T

Enumerator	Value	Description
TPM_RC_SUCCESS	0x000	
TPM_RC_BAD_TAG	0x01E	
RC_VER1	0x100	
TPM_RC_INITIALIZE	RC_VER1 + 0x000	
TPM_RC_FAILURE	RC_VER1 + 0x001	
TPM_RC_SEQUENCE	RC_VER1 + 0x003	
TPM_RC_PRIVATE	RC_VER1 + 0x00B	
TPM_RC_HMAC	RC_VER1 + 0x019	
TPM_RC_DISABLED	RC_VER1 + 0x020	
TPM_RC_EXCLUSIVE	RC_VER1 + 0x021	
TPM_RC_AUTH_TYPE	RC_VER1 + 0x024	
TPM_RC_AUTH_MISSING	RC_VER1 + 0x025	
TPM_RC_POLICY	RC_VER1 + 0x026	
TPM_RC_PCR	RC_VER1 + 0x027	
TPM_RC_PCR_CHANGED	RC_VER1 + 0x028	
TPM_RC_UPGRADE	RC_VER1 + 0x02D	
TPM_RC_TOO_MANY_CONTEXTS	RC_VER1 + 0x02E	
TPM_RC_AUTH_UNAVAILABLE	RC_VER1 + 0x02F	
TPM_RC_REBOOT	RC_VER1 + 0x030	
TPM_RC_UNBALANCED	RC_VER1 + 0x031	
TPM_RC_COMMAND_SIZE	RC_VER1 + 0x042	
TPM_RC_COMMAND_CODE	RC_VER1 + 0x043	
TPM_RC_AUTHSIZE	RC_VER1 + 0x044	
TPM_RC_AUTH_CONTEXT	RC_VER1 + 0x045	
TPM_RC_NV_RANGE	RC_VER1 + 0x046	
TPM_RC_NV_SIZE	RC_VER1 + 0x047	
TPM_RC_NV_LOCKED	RC_VER1 + 0x048	
TPM_RC_NV_AUTHORIZATION	RC_VER1 + 0x049	
TPM_RC_NV_UNINITIALIZED	RC_VER1 + 0x04A	
TPM_RC_NV_SPACE	RC_VER1 + 0x04B	
TPM_RC_NV_DEFINED	RC_VER1 + 0x04C	
TPM_RC_BAD_CONTEXT	RC_VER1 + 0x050	
TPM_RC_CPHASH	RC_VER1 + 0x051	
TPM_RC_PARENT	RC_VER1 + 0x052	
TPM_RC_NEEDS_TEST	RC_VER1 + 0x053	
TPM_RC_NO_RESULT	RC_VER1 + 0x054	
TPM_RC_SENSITIVE	RC_VER1 + 0x055	
RC_MAX_FM0	RC_VER1 + 0x07F	

Enumerator	Value	Description
RC_FMT1	0x080	
TPM_RC_ASYMMETRIC	RC_FMT1 + 0x001	
TPM_RC_ATTRIBUTES	RC_FMT1 + 0x002	
TPM_RC_HASH	RC_FMT1 + 0x003	
TPM_RC_VALUE	RC_FMT1 + 0x004	
TPM_RC_HIERARCHY	RC_FMT1 + 0x005	
TPM_RC_KEY_SIZE	RC_FMT1 + 0x007	
TPM_RC_MGF	RC_FMT1 + 0x008	
TPM_RC_MODE	RC_FMT1 + 0x009	
TPM_RC_TYPE	RC_FMT1 + 0x00A	
TPM_RC_HANDLE	RC_FMT1 + 0x00B	
TPM_RC_KDF	RC_FMT1 + 0x00C	
TPM_RC_RANGE	RC_FMT1 + 0x00D	
TPM_RC_AUTH_FAIL	RC_FMT1 + 0x00E	
TPM_RC_NONCE	RC_FMT1 + 0x00F	
TPM_RC_PP	RC_FMT1 + 0x010	
TPM_RC_SCHEME	RC_FMT1 + 0x012	
TPM_RC_SIZE	RC_FMT1 + 0x015	
TPM_RC_SYMMETRIC	RC_FMT1 + 0x016	
TPM_RC_TAG	RC_FMT1 + 0x017	
TPM_RC_SELECTOR	RC_FMT1 + 0x018	
TPM_RC_INSUFFICIENT	RC_FMT1 + 0x01A	
TPM_RC_SIGNATURE	RC_FMT1 + 0x01B	
TPM_RC_KEY	RC_FMT1 + 0x01C	
TPM_RC_POLICY_FAIL	RC_FMT1 + 0x01D	
TPM_RC_INTEGRITY	RC_FMT1 + 0x01F	
TPM_RC_TICKET	RC_FMT1 + 0x020	
TPM_RC_RESERVED_BITS	RC_FMT1 + 0x021	
TPM_RC_BAD_AUTH	RC_FMT1 + 0x022	
TPM_RC_EXPIRED	RC_FMT1 + 0x023	
TPM_RC_POLICY_CC	RC_FMT1 + 0x024	
TPM_RC_BINDING	RC_FMT1 + 0x025	
TPM_RC_CURVE	RC_FMT1 + 0x026	
TPM_RC_ECC_POINT	RC_FMT1 + 0x027	
TPM_RC_FW_LIMITED	RC_FMT1 + 0x028	
TPM_RC_SVN_LIMITED	RC_FMT1 + 0x029	
TPM_RC_PARMS	RC_FMT1 + 0x02A	
TPM_RC_EXT_MU	RC_FMT1 + 0x02B	
TPM_RC_ONE_SHOT_SIGNATURE	RC_FMT1 + 0x02C	
TPM_RC_SIGN_CONTEXT_KEY	RC_FMT1 + 0x02D	
TPM_RC_CHANNEL	RC_FMT1 + 0x030	
TPM_RC_CHANNEL_KEY	RC_FMT1 + 0x031	
RC_MAX_FMT1	RC_FMT1 + 0x03F	
RC_WARN	0x900	
TPM_RC_CONTEXT_GAP	RC_WARN + 0x001	
TPM_RC_OBJECT_MEMORY	RC_WARN + 0x002	
TPM_RC_SESSION_MEMORY	RC_WARN + 0x003	
TPM_RC_MEMORY	RC_WARN + 0x004	
TPM_RC_SESSION_HANDLES	RC_WARN + 0x005	
TPM_RC_OBJECT_HANDLES	RC_WARN + 0x006	
TPM_RC_LOCALITY	RC_WARN + 0x007	
TPM_RC_YIELDED	RC_WARN + 0x008	

Enumerator	Value	Description
TPM_RC_CANCELED	RC_WARN + 0x009	
TPM_RC_TESTING	RC_WARN + 0x00A	
TPM_RC_REFERENCE_H0	RC_WARN + 0x010	
TPM_RC_REFERENCE_H1	RC_WARN + 0x011	
TPM_RC_REFERENCE_H2	RC_WARN + 0x012	
TPM_RC_REFERENCE_H3	RC_WARN + 0x013	
TPM_RC_REFERENCE_H4	RC_WARN + 0x014	
TPM_RC_REFERENCE_H5	RC_WARN + 0x015	
TPM_RC_REFERENCE_H6	RC_WARN + 0x016	
TPM_RC_REFERENCE_S0	RC_WARN + 0x018	
TPM_RC_REFERENCE_S1	RC_WARN + 0x019	
TPM_RC_REFERENCE_S2	RC_WARN + 0x01A	
TPM_RC_REFERENCE_S3	RC_WARN + 0x01B	
TPM_RC_REFERENCE_S4	RC_WARN + 0x01C	
TPM_RC_REFERENCE_S5	RC_WARN + 0x01D	
TPM_RC_REFERENCE_S6	RC_WARN + 0x01E	
TPM_RC_NV_RATE	RC_WARN + 0x020	
TPM_RC_LOCKOUT	RC_WARN + 0x021	
TPM_RC_RETRY	RC_WARN + 0x022	
TPM_RC_NV_UNAVAILABLE	RC_WARN + 0x023	
RC_MAX_WARN	RC_WARN + 0x03F	
TPM_RC_NOT_USED	RC_WARN + 0x07F	
TPM_RC_H	0x000	
TPM_RC_P	0x040	
TPM_RC_S	0x800	
TPM_RC_1	0x100	
TPM_RC_2	0x200	
TPM_RC_3	0x300	
TPM_RC_4	0x400	
TPM_RC_5	0x500	
TPM_RC_6	0x600	
TPM_RC_7	0x700	
TPM_RC_8	0x800	
TPM_RC_9	0x900	
TPM_RC_A	0xA00	
TPM_RC_B	0xB00	
TPM_RC_C	0xC00	
TPM_RC_D	0xD00	
TPM_RC_E	0xE00	
TPM_RC_F	0xF00	
TPM_RC_N_MASK	0xF00	
TPM_RC_TIMEOUT	-100	

29.8.5.5 enum TPM_CLOCK_ADJUST_T

Enumerator	Value	Description
TPM_CLOCK_COARSE_SLOWER	-3	
TPM_CLOCK_MEDIUM_SLOWER	-2	
TPM_CLOCK_FINE_SLOWER	-1	
TPM_CLOCK_NO_CHANGE	0	

Enumerator	Value	Description
TPM_CLOCK_FINE_FASTER	1	
TPM_CLOCK_MEDIUM_FASTER	2	
TPM_CLOCK_COARSE_FASTER	3	

29.8.5.6 enum TPM_EO_T

Enumerator	Value	Description
TPM_EO_EQ	0x0000	
TPM_EO_NEQ	0x0001	
TPM_EO_SIGNED_GT	0x0002	
TPM_EO_UNSIGNED_GT	0x0003	
TPM_EO_SIGNED_LT	0x0004	
TPM_EO_UNSIGNED_LT	0x0005	
TPM_EO_SIGNED_GE	0x0006	
TPM_EO_UNSIGNED_GE	0x0007	
TPM_EO_SIGNED_LE	0x0008	
TPM_EO_UNSIGNED_LE	0x0009	
TPM_EO_BITSET	0x000A	
TPM_EO_BITCLEAR	0x000B	

29.8.5.7 enum TPM_ST_T

Enumerator	Value	Description
TPM_ST_RSP_COMMAND	0x00C4	
TPM_ST_NULL	0X8000	
TPM_ST_NO_SESSIONS	0x8001	
TPM_ST_SESSIONS	0x8002	
TPM_ST_ATTEST_NV	0x8014	
TPM_ST_ATTEST_COMMAND_AUDIT	0x8015	
TPM_ST_ATTEST_SESSION_AUDIT	0x8016	
TPM_ST_ATTEST_CERTIFY	0x8017	
TPM_ST_ATTEST_QUOTE	0x8018	
TPM_ST_ATTEST_TIME	0x8019	
TPM_ST_ATTEST_CREATION	0x801A	
TPM_ST_ATTEST_NV_DIGEST	0x801C	
TPM_ST_CREATION	0x8021	
TPM_ST_VERIFIED	0x8022	
TPM_ST_AUTH_SECRET	0x8023	
TPM_ST_HASHCHECK	0x8024	
TPM_ST_AUTH_SIGNED	0x8025	
TPM_ST_MESSAGE_VERIFIED	0x8026	
TPM_ST_DIGEST_VERIFIED	0x8027	
TPM_ST_FU_MANIFEST	0x8029	

29.8.5.8 enum TPM_SE_T

Enumerator	Value	Description
TPM_SE_HMAC	0x00	
TPM_SE_POLICY	0x01	
TPM_SE_TRIAL	0x03	

29.8.5.9 enum TPM_SU_T

Enumerator	Value	Description
TPM_SU_CLEAR	0x0000	
TPM_SU_STATE	0x0001	

29.8.5.10 enum TPM_CAP_T

Enumerator	Value	Description
TPM_CAP_FIRST	0x00000000	
TPM_CAP_ALGS	TPM_CAP_FIRST	
TPM_CAP_HANDLES	0x00000001	
TPM_CAP_COMMANDS	0x00000002	
TPM_CAP_PP_COMMANDS	0x00000003	
TPM_CAP_AUDIT_COMMANDS	0x00000004	
TPM_CAP_PCRS	0x00000005	
TPM_CAP_TPM_PROPERTIES	0x00000006	
TPM_CAP_PCR_PROPERTIES	0x00000007	
TPM_CAP_ECC_CURVES	0x00000008	
TPM_CAP_AUTH_POLICIES	0x00000009	
TPM_CAP_ACT	0x0000000A	
TPM_CAP_PUB_KEYS	0x0000000B	
TPM_CAP_SPDM_SESSION_INFO	0x0000000C	
TPM_CAP_LAST	TPM_CAP_SPDM_SESSION_INFO	
TPM_CAP_LAST	TPM_CAP_SPDM_SESSION_INFO	
TPM_CAP_VENDOR_PROPERTY	0x00000100	

29.8.5.11 enum TPM_PT_CONFIG_T

Enumerator	Value	Description
TPM_PT_CONFIG_ENCRYPTDECRYPT2	0x00000000	
TPM_PT_CONFIG_CHANGEEPS	0x00000001	
TPM_PT_CONFIG_TPMID_NV	0x00000002	
TPM_PT_CONFIG_SPDM_PSK	0x00000003	

29.8.5.12 enum TPM_PT_T

Enumerator	Value	Description
TPM_PT_NONE	0x00000000	
PT_GROUP	0x00000100	
PT_FIXED	PT_GROUP * 1	

Enumerator	Value	Description
TPM_PT_FAMILY_INDICATOR	PT_FIXED + 0	
TPM_PT_LEVEL	PT_FIXED + 1	
TPM_PT_REVISION	PT_FIXED + 2	
TPM_PT_DAY_OF_YEAR	PT_FIXED + 3	
TPM_PT_YEAR	PT_FIXED + 4	
TPM_PT_MANUFACTURER	PT_FIXED + 5	
TPM_PT_VENDOR_STRING_1	PT_FIXED + 6	
TPM_PT_VENDOR_STRING_2	PT_FIXED + 7	
TPM_PT_VENDOR_STRING_3	PT_FIXED + 8	
TPM_PT_VENDOR_STRING_4	PT_FIXED + 9	
TPM_PT_VENDOR_TPM_TYPE	PT_FIXED + 10	
TPM_PT_FIRMWARE_VERSION_1	PT_FIXED + 11	
TPM_PT_FIRMWARE_VERSION_2	PT_FIXED + 12	
TPM_PT_INPUT_BUFFER	PT_FIXED + 13	
TPM_PT_HR_TRANSIENT_MIN	PT_FIXED + 14	
TPM_PT_HR_PERSISTENT_MIN	PT_FIXED + 15	
TPM_PT_HR_LOADED_MIN	PT_FIXED + 16	
TPM_PT_ACTIVE_SESSIONS_MAX	PT_FIXED + 17	
TPM_PT_PCR_COUNT	PT_FIXED + 18	
TPM_PT_PCR_SELECT_MIN	PT_FIXED + 19	
TPM_PT_CONTEXT_GAP_MAX	PT_FIXED + 20	
TPM_PT_NV_COUNTERS_MAX	PT_FIXED + 22	
TPM_PT_NV_INDEX_MAX	PT_FIXED + 23	
TPM_PT_MEMORY	PT_FIXED + 24	
TPM_PT_CLOCK_UPDATE	PT_FIXED + 25	
TPM_PT_CONTEXT_HASH	PT_FIXED + 26	
TPM_PT_CONTEXT_SYM	PT_FIXED + 27	
TPM_PT_CONTEXT_SYM_SIZE	PT_FIXED + 28	
TPM_PT_ORDERLY_COUNT	PT_FIXED + 29	
TPM_PT_MAX_COMMAND_SIZE	PT_FIXED + 30	
TPM_PT_MAX_RESPONSE_SIZE	PT_FIXED + 31	
TPM_PT_MAX_DIGEST	PT_FIXED + 32	
TPM_PT_MAX_OBJECT_CONTEXT	PT_FIXED + 33	
TPM_PT_MAX_SESSION_CONTEXT	PT_FIXED + 34	
TPM_PT_PS_FAMILY_INDICATOR	PT_FIXED + 35	
TPM_PT_PS_LEVEL	PT_FIXED + 36	
TPM_PT_PS_REVISION	PT_FIXED + 37	
TPM_PT_PS_DAY_OF_YEAR	PT_FIXED + 38	
TPM_PT_PS_YEAR	PT_FIXED + 39	
TPM_PT_SPLIT_MAX	PT_FIXED + 40	
TPM_PT_TOTAL_COMMANDS	PT_FIXED + 41	
TPM_PT_LIBRARY_COMMANDS	PT_FIXED + 42	
TPM_PT_VENDOR_COMMANDS	PT_FIXED + 43	
TPM_PT_NV_BUFFER_MAX	PT_FIXED + 44	
TPM_PT_MODES	PT_FIXED + 45	
TPM_PT_MAX_CAP_BUFFER	PT_FIXED + 46	
TPM_PT_FIRMWARE_SVN	PT_FIXED + 47	
TPM_PT_FIRMWARE_MAX_SVN	PT_FIXED + 48	
TPM_PT_ML_PARAMETER_SETS	PT_FIXED + 49	
PT_VAR	PT_GROUP * 2	
TPM_PT_PERMANENT	PT_VAR + 0	
TPM_PT_STARTUP_CLEAR	PT_VAR + 1	

Enumerator	Value	Description
TPM_PT_HR_NV_INDEX	PT_VAR + 2	
TPM_PT_HR_LOADED	PT_VAR + 3	
TPM_PT_HR_LOADED_AVAIL	PT_VAR + 4	
TPM_PT_HR_ACTIVE	PT_VAR + 5	
TPM_PT_HR_ACTIVE_AVAIL	PT_VAR + 6	
TPM_PT_HR_TRANSIENT_AVAIL	PT_VAR + 7	
TPM_PT_HR_PERSISTENT	PT_VAR + 8	
TPM_PT_HR_PERSISTENT_AVAIL	PT_VAR + 9	
TPM_PT_NV_COUNTERS	PT_VAR + 10	
TPM_PT_NV_COUNTERS_AVAIL	PT_VAR + 11	
TPM_PT_ALGORITHM_SET	PT_VAR + 12	
TPM_PT_LOADED_CURVES	PT_VAR + 13	
TPM_PT_LOCKOUT_COUNTER	PT_VAR + 14	
TPM_PT_MAX_AUTH_FAIL	PT_VAR + 15	
TPM_PT_LOCKOUT_INTERVAL	PT_VAR + 16	
TPM_PT_LOCKOUT_RECOVERY	PT_VAR + 17	
TPM_PT_NV_WRITE_RECOVERY	PT_VAR + 18	
TPM_PT_AUDIT_COUNTER_0	PT_VAR + 19	
TPM_PT_AUDIT_COUNTER_1	PT_VAR + 20	

29.8.5.13 enum TPM_PT_PCR_T

Enumerator	Value	Description
TPM_PT_PCR_FIRST	0x00000000	
TPM_PT_PCR_SAVE	TPM_PT_PCR_FIRST	
TPM_PT_PCR_EXTEND_L0	0x00000001	
TPM_PT_PCR_RESET_L0	0x00000002	
TPM_PT_PCR_EXTEND_L1	0x00000003	
TPM_PT_PCR_RESET_L1	0x00000004	
TPM_PT_PCR_EXTEND_L2	0x00000005	
TPM_PT_PCR_RESET_L2	0x00000006	
TPM_PT_PCR_EXTEND_L3	0x00000007	
TPM_PT_PCR_RESET_L3	0x00000008	
TPM_PT_PCR_EXTEND_L4	0x00000009	
TPM_PT_PCR_RESET_L4	0x0000000A	
TPM_PT_PCR_NO_INCREMENT	0x00000011	
TPM_PT_PCR_DRTM_RESET	0x00000012	
TPM_PT_PCR_POLICY	0x00000013	
TPM_PT_PCR_AUTH	0x00000014	
TPM_PT_PCR_LAST	TPM_PT_PCR_AUTH	

29.8.5.14 enum TPM_PS_T

Enumerator	Value	Description
TPM_PS_MAIN	0x00000000	
TPM_PS_PC	0x00000001	
TPM_PS_PDA	0x00000002	
TPM_PS_CELL_PHONE	0x00000003	

Enumerator	Value	Description
TPM_PS_SERVER	0x00000004	
TPM_PS_PERIPHERAL	0x00000005	
TPM_PS_TSS	0x00000006	
TPM_PS_STORAGE	0x00000007	
TPM_PS_AUTHENTICATION	0x00000008	
TPM_PS_EMBEDDED	0x00000009	
TPM_PS_HARDCOPY	0x0000000A	
TPM_PS_INFRASTRUCTURE	0x0000000B	
TPM_PS_VIRTUALIZATION	0x0000000C	
TPM_PS_TNC	0x0000000D	
TPM_PS_MULTI_TENANT	0x0000000E	
TPM_PS_TC	0x0000000F	

29.8.5.15 enum TPM_HT_T

Enumerator	Value	Description
TPM_HT_PCR	0x00	
TPM_HT_NV_INDEX	0x01	
TPM_HT_HMAC_SESSION	0x02	
TPM_HT_LOADED_SESSION	0x02	
TPM_HT_POLICY_SESSION	0x03	
TPM_HT_ACTIVE_SESSION	0x03	
TPM_HT_PERMANENT	0x40	
TPM_HT_TRANSIENT	0x80	
TPM_HT_PERSISTENT	0x81	

29.8.5.16 enum TPM_RH_T

Enumerator	Value	Description
TPM_RH_FIRST	0x40000000	
TPM_RH_SRK	TPM_RH_FIRST	
TPM_RH_OWNER	0x40000001	
TPM_RH_REVOKE	0x40000002	
TPM_RH_TRANSPORT	0x40000003	
TPM_RH_OPERATOR	0x40000004	
TPM_RH_ADMIN	0x40000005	
TPM_RH_EK	0x40000006	
TPM_RH_NULL	0x40000007	
TPM_RH_UNASSIGNED	0x40000008	
TPM_RS_PW	0x40000009	
TPM_RH_LOCKOUT	0x4000000A	
TPM_RH_ENDORSEMENT	0x4000000B	
TPM_RH_PLATFORM	0x4000000C	
TPM_RH_PLATFORM_NV	0x4000000D	
TPM_RH_AUTH_00	0x40000010	
TPM_RH_AUTH_FF	0x4000010F	
TPM_RH_LAST	TPM_RH_AUTH_FF	

29.8.5.17 enum TPMA_ALGORITHM_mask

Enumerator	Value	Description
TPMA_ALGORITHM_asymmetric	0x00000001	
TPMA_ALGORITHM_symmetric	0x00000002	
TPMA_ALGORITHM_hash	0x00000004	
TPMA_ALGORITHM_object	0x00000008	
TPMA_ALGORITHM_signing	0x00000100	
TPMA_ALGORITHM_encrypting	0x00000200	
TPMA_ALGORITHM_method	0x00000400	

29.8.5.18 enum TPMA_OBJECT_mask

Enumerator	Value	Description
TPMA_OBJECT_fixedTPM	0x00000002	
TPMA_OBJECT_stClear	0x00000004	
TPMA_OBJECT_fixedParent	0x00000010	
TPMA_OBJECT_sensitiveDataOrigin	0x00000020	
TPMA_OBJECT_userWithAuth	0x00000040	
TPMA_OBJECT_adminWithPolicy	0x00000080	
TPMA_OBJECT_noDA	0x00000400	
TPMA_OBJECT_encryptedDuplication	0x00000800	
TPMA_OBJECT_restricted	0x00010000	
TPMA_OBJECT_decrypt	0x00020000	
TPMA_OBJECT_sign	0x00040000	
TPMA_OBJECT_derivedDataOrigin	0x00000200	
TPMA_OBJECT_firmwareLimited	0x00000100	
TPMA_OBJECT_svnLimited	0x00000200	
TPMA_OBJECT_x509sign	0x00080000	

29.8.5.19 enum TPMA_SESSION_mask

Enumerator	Value	Description
TPMA_SESSION_continueSession	0x01	
TPMA_SESSION_auditExclusive	0x02	
TPMA_SESSION_auditReset	0x04	
TPMA_SESSION_decrypt	0x20	
TPMA_SESSION_encrypt	0x40	
TPMA_SESSION_audit	0x80	

29.8.5.20 enum TPMA_LOCALITY_mask

Enumerator	Value	Description
TPM_LOC_ZERO	0x01	
TPM_LOC_ONE	0x02	
TPM_LOC_TWO	0x04	
TPM_LOC_THREE	0x08	

Enumerator	Value	Description
TPM_LOC_FOUR	0x10	

29.8.5.21 enum TPMA_PERMANENT_mask

Enumerator	Value	Description
TPMA_PERMANENT_ownerAuthSet	0x00000001	
TPMA_PERMANENT_endorsementAuthSet	0x00000002	
TPMA_PERMANENT_lockoutAuthSet	0x00000004	
TPMA_PERMANENT_disableClear	0x00000100	
TPMA_PERMANENT_inLockout	0x00000200	
TPMA_PERMANENT_tpmGeneratedEPS	0x00000400	

29.8.5.22 enum TPMA_MEMORY_mask

Enumerator	Value	Description
TPMA_MEMORY_sharedRAM	0x00000001	
TPMA_MEMORY_sharedNV	0x00000002	
TPMA_MEMORY_objectCopiedToRam	0x00000004	

29.8.5.23 enum TPMA_CC_mask

Enumerator	Value	Description
TPMA_CC_commandIndex	0x0000FFFF	
TPMA_CC_nv	0x00400000	
TPMA_CC_extensive	0x00800000	
TPMA_CC_flushed	0x01000000	
TPMA_CC_cHandles	0x0E000000	
TPMA_CC_rHandle	0x10000000	
TPMA_CC_V	0x20000000	

29.8.5.24 enum TPMA_MODES_mask

Enumerator	Value	Description
TPMA_MODES_FIPS_140_2	0x00000001	
TPMA_MODES_FIPS_140_3	0x00000002	

29.8.5.25 enum TPMA_ML_PARAMETER_SET_mask

Enumerator	Value	Description
TPMA_ML_PARAMETER_SET_mlKem_512	0x00000001	
TPMA_ML_PARAMETER_SET_mlKem_768	0x00000002	
TPMA_ML_PARAMETER_SET_mlKem_1024	0x00000004	
TPMA_ML_PARAMETER_SET_mlDsa_44	0x00000008	

Enumerator	Value	Description
TPMA_ML_PARAMETER_SET_mlDsa_65	0x00000010	
TPMA_ML_PARAMETER_SET_mlDsa_87	0x00000020	
TPMA_ML_PARAMETER_SET_extMu	0x00000040	

29.8.5.26 enum TPMA_ACT_T

Enumerator	Value	Description
TPMA_ACT_signaled	0x00000001	
TPMA_ACT_preserveSignaled	0x00000002	

29.8.5.27 enum TPM_NT

Enumerator	Value	Description
TPM_NT_ORDINARY	0x0	
TPM_NT_COUNTER	0x1	
TPM_NT_BITS	0x2	
TPM_NT_EXTEND	0x4	
TPM_NT_PIN_FAIL	0x8	
TPM_NT_PIN_PASS	0x9	

29.8.5.28 enum TPM_MODE_Vendor_Mask

Enumerator	Value	Description
TPMLib_2	0x01	
TPMFips	0x02	
TPMLowPowerOff	0x00	
TPMLowPowerByRegister	0x04	
TPMLowPowerByGpio	0x08	
TPMLowPowerAuto	0x0C	

29.8.5.29 enum TPMI_GPIO_NAME_T

Enumerator	Value	Description
TPM_GPIO_PP	0x00000000	
TPM_GPIO_LP	0x00000001	
TPM_GPIO_C	0x00000002	
TPM_GPIO_D	0x00000003	

29.8.5.30 enum TPMI_GPIO_MODE_T

Enumerator	Value	Description
TPM_GPIO_MODE_STANDARD	0x00000000	
TPM_GPIO_MODE_FLOATING	0x00000001	

Enumerator	Value	Description
TPM_GPIO_MODE_PULLUP	0x00000002	
TPM_GPIO_MODE_PULLDOWN	0x00000003	
TPM_GPIO_MODE_OPENDRAIN	0x00000004	
TPM_GPIO_MODE_PUSHPULL	0x00000005	
TPM_GPIO_MODE_UNCONFIG	0x00000006	
TPM_GPIO_MODE_DEFAULT	TPM_GPIO_MODE_PULLDOWN	
TPM_GPIO_MODE_MAX	TPM_GPIO_MODE_UNCONFIG	
TPM_GPIO_MODE_INPUT_MIN	TPM_GPIO_MODE_FLOATING	
TPM_GPIO_MODE_INPUT_MAX	TPM_GPIO_MODE_PULLDOWN	
TPM_GPIO_MODE_PUSHPULL	0x00000005	
TPM_GPIO_MODE_OPENDRAIN	0x00000004	
TPM_GPIO_MODE_PULLUP	0x00000002	
TPM_GPIO_MODE_UNCONFIG	0x00000006	
TPM_GPIO_MODE_DEFAULT	TPM_GPIO_MODE_PULLDOWN	
TPM_GPIO_MODE_MAX	TPM_GPIO_MODE_UNCONFIG	
TPM_GPIO_MODE_INPUT_MIN	TPM_GPIO_MODE_FLOATING	
TPM_GPIO_MODE_INPUT_MAX	TPM_GPIO_MODE_PULLDOWN	

29.8.5.31 enum TPMI_GPIO_MODE_T

Enumerator	Value	Description
TPM_GPIO_MODE_STANDARD	0x00000000	
TPM_GPIO_MODE_FLOATING	0x00000001	
TPM_GPIO_MODE_PULLUP	0x00000002	
TPM_GPIO_MODE_PULLDOWN	0x00000003	
TPM_GPIO_MODE_OPENDRAIN	0x00000004	
TPM_GPIO_MODE_PUSHPULL	0x00000005	
TPM_GPIO_MODE_UNCONFIG	0x00000006	
TPM_GPIO_MODE_DEFAULT	TPM_GPIO_MODE_PULLDOWN	
TPM_GPIO_MODE_MAX	TPM_GPIO_MODE_UNCONFIG	
TPM_GPIO_MODE_INPUT_MIN	TPM_GPIO_MODE_FLOATING	
TPM_GPIO_MODE_INPUT_MAX	TPM_GPIO_MODE_PULLDOWN	
TPM_GPIO_MODE_PUSHPULL	0x00000005	
TPM_GPIO_MODE_OPENDRAIN	0x00000004	
TPM_GPIO_MODE_PULLUP	0x00000002	
TPM_GPIO_MODE_UNCONFIG	0x00000006	
TPM_GPIO_MODE_DEFAULT	TPM_GPIO_MODE_PULLDOWN	
TPM_GPIO_MODE_MAX	TPM_GPIO_MODE_UNCONFIG	
TPM_GPIO_MODE_INPUT_MIN	TPM_GPIO_MODE_FLOATING	
TPM_GPIO_MODE_INPUT_MAX	TPM_GPIO_MODE_PULLDOWN	

29.8.5.32 enum TPM_Vendor_t

Enumerator	Value	Description
TPM_VENDOR_UNKNOWN	0	
TPM_VENDOR_INFINEON	0x15d1	
TPM_VENDOR_STM	0x104a	
TPM_VENDOR_MCHP	0x1114	

Enumerator	Value	Description
TPM_VENDOR_NUVOTON	0x1050	
TPM_VENDOR_NATIONTECH	0x1B4E	
TPM_VENDOR_SEALSQ	0x2406	

29.8.5.33 typedef TPM_MODIFIER_INDICATOR

```
typedef UINT32 TPM_MODIFIER_INDICATOR;
```

29.8.5.34 typedef TPM_AUTHORIZATION_SIZE

```
typedef UINT32 TPM_AUTHORIZATION_SIZE;
```

29.8.5.35 typedef TPM_PARAMETER_SIZE

```
typedef UINT32 TPM_PARAMETER_SIZE;
```

29.8.5.36 typedef TPM_KEY_SIZE

```
typedef UINT16 TPM_KEY_SIZE;
```

29.8.5.37 typedef TPM_KEY_BITS

```
typedef UINT16 TPM_KEY_BITS;
```

29.8.5.38 typedef TPM_GENERATED

```
typedef UINT32 TPM_GENERATED;
```

29.8.5.39 typedef TPM_ALG_ID

```
typedef UINT16 TPM_ALG_ID;
```

29.8.5.40 typedef TPM_ECC_CURVE

```
typedef UINT16 TPM_ECC_CURVE;
```

29.8.5.41 typedef TPMI_MLKEM_PARAMETER_SET

```
typedef UINT16 TPMI_MLKEM_PARAMETER_SET;
```

29.8.5.42 typedef TPMI_MLDSA_PARAMETER_SET

```
typedef UINT16 TPMI_MLDSA_PARAMETER_SET;
```

29.8.5.43 typedef TPM_CC

```
typedef UINT32 TPM_CC;
```

29.8.5.44 typedef TPM_RC

```
typedef INT32 TPM_RC;
```

29.8.5.45 typedef TPM_CLOCK_ADJUST

```
typedef UINT8 TPM_CLOCK_ADJUST;
```

29.8.5.46 typedef TPM_EO

```
typedef UINT16 TPM_EO;
```

29.8.5.47 typedef TPM_ST

```
typedef UINT16 TPM_ST;
```

29.8.5.48 typedef TPM_SE

```
typedef UINT8 TPM_SE;
```

29.8.5.49 typedef TPM_SU

```
typedef UINT16 TPM_SU;
```

29.8.5.50 typedef TPM_CAP

```
typedef UINT32 TPM_CAP;
```

29.8.5.51 typedef TPM_PT_CONFIG

```
typedef UINT32 TPM_PT_CONFIG;
```

29.8.5.52 typedef TPM_PT

```
typedef UINT32 TPM_PT;
```

29.8.5.53 typedef TPM_PT_PCR

```
typedef UINT32 TPM_PT_PCR;
```

29.8.5.54 typedef TPM_PS

```
typedef UINT32 TPM_PS;
```

29.8.5.55 typedef TPM_HANDLE

```
typedef UINT32 TPM_HANDLE;
```

29.8.5.56 typedef TPM_HT

```
typedef UINT8 TPM_HT;
```

29.8.5.57 typedef TPM_RH

```
typedef UINT32 TPM_RH;
```

29.8.5.58 typedef TPM_HC

```
typedef UINT32 TPM_HC;
```

29.8.5.59 typedef TPMA_ALGORITHM

```
typedef UINT32 TPMA_ALGORITHM;
```

29.8.5.60 typedef TPMA_OBJECT

```
typedef UINT32 TPMA_OBJECT;
```

29.8.5.61 typedef TPMA_SESSION

```
typedef BYTE TPMA_SESSION;
```

29.8.5.62 typedef TPMA_LOCALITY

```
typedef BYTE TPMA_LOCALITY;
```

29.8.5.63 typedef TPMA_PERMANENT

```
typedef UINT32 TPMA_PERMANENT;
```

29.8.5.64 typedef TPMA_STARTUP_CLEAR

```
typedef UINT32 TPMA_STARTUP_CLEAR;
```

29.8.5.65 typedef TPMA_MEMORY

```
typedef UINT32 TPMA_MEMORY;
```

29.8.5.66 typedef TPMA_CC

```
typedef UINT32 TPMA_CC;
```

29.8.5.67 typedef TPMA_MODES

```
typedef UINT32 TPMA_MODES;
```

29.8.5.68 typedef TPMA_ML_PARAMETER_SET

```
typedef UINT32 TPMA_ML_PARAMETER_SET;
```

29.8.5.69 typedef TPMI_YES_NO

```
typedef BYTE TPMI_YES_NO;
```

29.8.5.70 typedef TPMI_DH_OBJECT

```
typedef TPM_HANDLE TPMI_DH_OBJECT;
```

29.8.5.71 typedef TPMI_DH_PARENT

```
typedef TPM_HANDLE TPMI_DH_PARENT;
```

29.8.5.72 typedef TPMI_DH_PERSISTENT

```
typedef TPM_HANDLE TPMI_DH_PERSISTENT;
```

29.8.5.73 typedef TPMI_DH_ENTITY

```
typedef TPM_HANDLE TPMI_DH_ENTITY;
```

29.8.5.74 typedef TPMI_DH_PCR

```
typedef TPM_HANDLE TPMI_DH_PCR;
```

29.8.5.75 typedef TPMI_SH_AUTH_SESSION

```
typedef TPM_HANDLE TPMI_SH_AUTH_SESSION;
```

29.8.5.76 typedef TPMI_SH_HMAC

```
typedef TPM_HANDLE TPMI_SH_HMAC;
```

29.8.5.77 typedef TPMI_SH_POLICY

```
typedef TPM_HANDLE TPMI_SH_POLICY;
```

29.8.5.78 typedef TPMI_DH_CONTEXT

```
typedef TPM_HANDLE TPMI_DH_CONTEXT;
```

29.8.5.79 typedef TPMI_RH_HIERARCHY

```
typedef TPM_HANDLE TPMI_RH_HIERARCHY;
```

29.8.5.80 typedef TPMI_RH_ENABLES

```
typedef TPM_HANDLE TPMI_RH_ENABLES;
```

29.8.5.81 typedef TPMI_RH_HIERARCHY_AUTH

```
typedef TPM_HANDLE TPMI_RH_HIERARCHY_AUTH;
```

29.8.5.82 typedef TPMI_RH_PLATFORM

```
typedef TPM_HANDLE TPMI_RH_PLATFORM;
```

29.8.5.83 typedef TPMI_RH_OWNER

```
typedef TPM_HANDLE TPMI_RH_OWNER;
```

29.8.5.84 typedef TPMI_RH_ENDORSEMENT

```
typedef TPM_HANDLE TPMI_RH_ENDORSEMENT;
```

29.8.5.85 typedef TPMI_RH_PROVISION

```
typedef TPM_HANDLE TPMI_RH_PROVISION;
```

29.8.5.86 typedef TPMI_RH_CLEAR

```
typedef TPM_HANDLE TPMI_RH_CLEAR;
```

29.8.5.87 typedef TPMI_RH_NV_AUTH

```
typedef TPM_HANDLE TPMI_RH_NV_AUTH;
```

29.8.5.88 typedef TPMI_RH_LOCKOUT

```
typedef TPM_HANDLE TPMI_RH_LOCKOUT;
```

29.8.5.89 typedef TPMI_RH_NV_INDEX

```
typedef TPM_HANDLE TPMI_RH_NV_INDEX;
```

29.8.5.90 typedef TPMI_DH_AC

```
typedef TPM_HANDLE TPMI_DH_AC;
```

29.8.5.91 typedef TPMI_ALG_HASH

```
typedef TPM_ALG_ID TPMI_ALG_HASH;
```

29.8.5.92 typedef TPMI_ALG_ASYM

```
typedef TPM_ALG_ID TPMI_ALG_ASYM;
```

29.8.5.93 typedef TPMI_ALG_SYM

```
typedef TPM_ALG_ID TPMI_ALG_SYM;
```

29.8.5.94 typedef TPMI_ALG_SYM_OBJECT

```
typedef TPM_ALG_ID TPMI_ALG_SYM_OBJECT;
```

29.8.5.95 typedef TPMI_ALG_SYM_MODE

```
typedef TPM_ALG_ID TPMI_ALG_SYM_MODE;
```

29.8.5.96 typedef TPMI_ALG_KDF

```
typedef TPM_ALG_ID TPMI_ALG_KDF;
```

29.8.5.97 typedef TPMI_ALG_SIG_SCHEME

```
typedef TPM_ALG_ID TPMI_ALG_SIG_SCHEME;
```

29.8.5.98 typedef TPMI_ECC_KEY_EXCHANGE

```
typedef TPM_ALG_ID TPMI_ECC_KEY_EXCHANGE;
```

29.8.5.99 typedef TPMI_ST_COMMAND_TAG

```
typedef TPM_ST TPMI_ST_COMMAND_TAG;
```

29.8.5.100 typedef TPMS_ALGORITHM_DESCRIPTION

```
typedef struct TPMS_ALGORITHM_DESCRIPTION TPMS_ALGORITHM_DESCRIPTION;
```

29.8.5.101 typedef TPMU_HA

```
typedef union TPMU_HA TPMU_HA;
```

29.8.5.102 typedef TPMT_HA

```
typedef struct TPMT_HA TPMT_HA;
```

29.8.5.103 typedef TPM2B_DIGEST

```
typedef struct TPM2B_DIGEST TPM2B_DIGEST;
```

29.8.5.104 typedef TPM2B_DATA

```
typedef struct TPM2B_DATA TPM2B_DATA;
```

29.8.5.105 typedef TPM2B_NONCE

```
typedef TPM2B_DIGEST TPM2B_NONCE;
```

29.8.5.106 typedef TPM2B_AUTH

```
typedef TPM2B_DIGEST TPM2B_AUTH;
```

29.8.5.107 typedef TPM2B_OPERAND

```
typedef TPM2B_DIGEST TPM2B_OPERAND;
```

29.8.5.108 typedef TPM2B_EVENT

```
typedef struct TPM2B_EVENT TPM2B_EVENT;
```

29.8.5.109 typedef TPM2B_MAX_BUFFER

```
typedef struct TPM2B_MAX_BUFFER TPM2B_MAX_BUFFER;
```

29.8.5.110 typedef TPM2B_MAX_NV_BUFFER

```
typedef struct TPM2B_MAX_NV_BUFFER TPM2B_MAX_NV_BUFFER;
```

29.8.5.111 typedef TPM2B_TIMEOUT

```
typedef TPM2B_DIGEST TPM2B_TIMEOUT;
```

29.8.5.112 typedef TPM2B_IV

```
typedef struct TPM2B_IV TPM2B_IV;
```

29.8.5.113 typedef TPM2B_SIGNATURE_CTX

```
typedef struct TPM2B_SIGNATURE_CTX TPM2B_SIGNATURE_CTX;
```

29.8.5.114 typedef TPM2B_SIGNATURE_HINT

```
typedef struct TPM2B_SIGNATURE_HINT TPM2B_SIGNATURE_HINT;
```

29.8.5.115 typedef TPM2B_KEM_CIPHERTEXT

```
typedef struct TPM2B_KEM_CIPHERTEXT TPM2B_KEM_CIPHERTEXT;
```

29.8.5.116 typedef TPM2B_SHARED_SECRET

```
typedef struct TPM2B_SHARED_SECRET TPM2B_SHARED_SECRET;
```

29.8.5.117 typedef TPM2B_PUBLIC_KEY_MLDSA

```
typedef struct TPM2B_PUBLIC_KEY_MLDSA TPM2B_PUBLIC_KEY_MLDSA;
```

29.8.5.118 typedef TPM2B_PRIVATE_KEY_MLDSA

```
typedef struct TPM2B_PRIVATE_KEY_MLDSA TPM2B_PRIVATE_KEY_MLDSA;
```

29.8.5.119 typedef TPM2B_PUBLIC_KEY_MLKEM

```
typedef struct TPM2B_PUBLIC_KEY_MLKEM TPM2B_PUBLIC_KEY_MLKEM;
```

29.8.5.120 typedef TPM2B_PRIVATE_KEY_MLKEM

```
typedef struct TPM2B_PRIVATE_KEY_MLKEM TPM2B_PRIVATE_KEY_MLKEM;
```

29.8.5.121 typedef TPM2B_MLDSA_SIGNATURE

```
typedef struct TPM2B_MLDSA_SIGNATURE TPM2B_MLDSA_SIGNATURE;
```

29.8.5.122 typedef TPMU_NAME

```
typedef union TPMU_NAME TPMU_NAME;
```

29.8.5.123 typedef TPM2B_NAME

```
typedef struct TPM2B_NAME TPM2B_NAME;
```

29.8.5.124 typedef TPMS_PCR_SELECT

```
typedef struct TPMS_PCR_SELECT TPMS_PCR_SELECT;
```

29.8.5.125 typedef TPMS_PCR_SELECTION

```
typedef struct TPMS_PCR_SELECTION TPMS_PCR_SELECTION;
```

29.8.5.126 typedef TPMT_TK_CREATION

```
typedef struct TPMT_TK_CREATION TPMT_TK_CREATION;
```

29.8.5.127 typedef TPMT_TK_VERIFIED

```
typedef struct TPMT_TK_VERIFIED TPMT_TK_VERIFIED;
```

29.8.5.128 typedef TPMT_TK_AUTH

```
typedef struct TPMT_TK_AUTH TPMT_TK_AUTH;
```

29.8.5.129 typedef TPMT_TK_HASHCHECK

```
typedef struct TPMT_TK_HASHCHECK TPMT_TK_HASHCHECK;
```

29.8.5.130 typedef TPMS_ALG_PROPERTY

```
typedef struct TPMS_ALG_PROPERTY TPMS_ALG_PROPERTY;
```

29.8.5.131 typedef TPMS_TAGGED_PROPERTY

```
typedef struct TPMS_TAGGED_PROPERTY TPMS_TAGGED_PROPERTY;
```

29.8.5.132 typedef TPMS_TAGGED_PCR_SELECT

```
typedef struct TPMS_TAGGED_PCR_SELECT TPMS_TAGGED_PCR_SELECT;
```

29.8.5.133 typedef TPMS_TAGGED_POLICY

```
typedef struct TPMS_TAGGED_POLICY TPMS_TAGGED_POLICY;
```

29.8.5.134 typedef TPMS_TAGGED_CONFIG

```
typedef struct TPMS_TAGGED_CONFIG TPMS_TAGGED_CONFIG;
```

29.8.5.135 typedef TPML_CC

```
typedef struct TPML_CC TPML_CC;
```

29.8.5.136 typedef TPML_CCA

```
typedef struct TPML_CCA TPML_CCA;
```

29.8.5.137 typedef TPML_ALG

```
typedef struct TPML_ALG TPML_ALG;
```

29.8.5.138 typedef TPML_HANDLE

```
typedef struct TPML_HANDLE TPML_HANDLE;
```

29.8.5.139 typedef TPML_DIGEST

```
typedef struct TPML_DIGEST TPML_DIGEST;
```

29.8.5.140 typedef TPML_DIGEST_VALUES

```
typedef struct TPML_DIGEST_VALUES TPML_DIGEST_VALUES;
```

29.8.5.141 typedef TPML_PCR_SELECTION

```
typedef struct TPML_PCR_SELECTION TPML_PCR_SELECTION;
```

29.8.5.142 typedef TPML_ALG_PROPERTY

```
typedef struct TPML_ALG_PROPERTY TPML_ALG_PROPERTY;
```

29.8.5.143 typedef TPML_TAGGED_TPM_PROPERTY

```
typedef struct TPML_TAGGED_TPM_PROPERTY TPML_TAGGED_TPM_PROPERTY;
```

29.8.5.144 typedef TPML_TAGGED_PCR_PROPERTY

```
typedef struct TPML_TAGGED_PCR_PROPERTY TPML_TAGGED_PCR_PROPERTY;
```

29.8.5.145 typedef TPML_ECC_CURVE

```
typedef struct TPML_ECC_CURVE TPML_ECC_CURVE;
```

29.8.5.146 typedef TPML_TAGGED_POLICY

```
typedef struct TPML_TAGGED_POLICY TPML_TAGGED_POLICY;
```

29.8.5.147 typedef TPML_TAGGED_CONFIG

```
typedef struct TPML_TAGGED_CONFIG TPML_TAGGED_CONFIG;
```

29.8.5.148 typedef TPMA_ACT

```
typedef UINT32 TPMA_ACT;
```

29.8.5.149 typedef TPMS_ACT_DATA

```
typedef struct TPMS_ACT_DATA TPMS_ACT_DATA;
```

29.8.5.150 typedef TPML_ACT_DATA

```
typedef struct TPML_ACT_DATA TPML_ACT_DATA;
```

29.8.5.151 typedef TPMS_SPDM_SESSION_INFO

```
typedef struct TPMS_SPDM_SESSION_INFO TPMS_SPDM_SESSION_INFO;
```

29.8.5.152 typedef TPML_SPDM_SESSION_INFO

```
typedef struct TPML_SPDM_SESSION_INFO TPML_SPDM_SESSION_INFO;
```

29.8.5.153 typedef TPMU_CAPABILITIES

```
typedef union TPMU_CAPABILITIES TPMU_CAPABILITIES;
```

29.8.5.154 typedef TPMS_CAPABILITY_DATA

```
typedef struct TPMS_CAPABILITY_DATA TPMS_CAPABILITY_DATA;
```

29.8.5.155 typedef TPMS_CLOCK_INFO

```
typedef struct TPMS_CLOCK_INFO TPMS_CLOCK_INFO;
```

29.8.5.156 typedef TPMS_TIME_INFO

```
typedef struct TPMS_TIME_INFO TPMS_TIME_INFO;
```

29.8.5.157 typedef TPMS_TIME_ATTEST_INFO

```
typedef struct TPMS_TIME_ATTEST_INFO TPMS_TIME_ATTEST_INFO;
```

29.8.5.158 typedef TPMS_CERTIFY_INFO

```
typedef struct TPMS_CERTIFY_INFO TPMS_CERTIFY_INFO;
```

29.8.5.159 typedef TPMS_QUOTE_INFO

```
typedef struct TPMS_QUOTE_INFO TPMS_QUOTE_INFO;
```

29.8.5.160 typedef TPMS_COMMAND_AUDIT_INFO

```
typedef struct TPMS_COMMAND_AUDIT_INFO TPMS_COMMAND_AUDIT_INFO;
```

29.8.5.161 typedef TPMS_SESSION_AUDIT_INFO

```
typedef struct TPMS_SESSION_AUDIT_INFO TPMS_SESSION_AUDIT_INFO;
```

29.8.5.162 typedef TPMS_CREATION_INFO

```
typedef struct TPMS_CREATION_INFO TPMS_CREATION_INFO;
```

29.8.5.163 typedef TPMS_NV_CERTIFY_INFO

```
typedef struct TPMS_NV_CERTIFY_INFO TPMS_NV_CERTIFY_INFO;
```

29.8.5.164 typedef TPMS_NV_DIGEST_CERTIFY_INFO

```
typedef struct TPMS_NV_DIGEST_CERTIFY_INFO TPMS_NV_DIGEST_CERTIFY_INFO;
```

29.8.5.165 typedef TPMI_ST_ATTEST

```
typedef TPM_ST TPMI_ST_ATTEST;
```

29.8.5.166 typedef TPMU_ATTEST

```
typedef union TPMU_ATTEST TPMU_ATTEST;
```

29.8.5.167 typedef TPMS_ATTEST

```
typedef struct TPMS_ATTEST TPMS_ATTEST;
```

29.8.5.168 typedef TPM2B_ATTEST

```
typedef struct TPM2B_ATTEST TPM2B_ATTEST;
```

29.8.5.169 typedef TPMI_AES_KEY_BITS

```
typedef TPM_KEY_BITS TPMI_AES_KEY_BITS;
```

29.8.5.170 typedef TPMU_SYM_KEY_BITS

```
typedef union TPMU_SYM_KEY_BITS TPMU_SYM_KEY_BITS;
```

29.8.5.171 typedef TPMU_SYM_MODE

```
typedef union TPMU_SYM_MODE TPMU_SYM_MODE;
```

29.8.5.172 typedef TPMT_SYM_DEF

```
typedef struct TPMT_SYM_DEF TPMT_SYM_DEF;
```

29.8.5.173 typedef TPMT_SYM_DEF_OBJECT

```
typedef TPMT_SYM_DEF TPMT_SYM_DEF_OBJECT;
```

29.8.5.174 typedef TPM2B_SYM_KEY

```
typedef struct TPM2B_SYM_KEY TPM2B_SYM_KEY;
```

29.8.5.175 typedef TPMS_SYMCIPHER_PARMS

```
typedef struct TPMS_SYMCIPHER_PARMS TPMS_SYMCIPHER_PARMS;
```

29.8.5.176 typedef TPM2B_LABEL

```
typedef struct TPM2B_LABEL TPM2B_LABEL;
```

29.8.5.177 typedef TPMS_DERIVE

```
typedef struct TPMS_DERIVE TPMS_DERIVE;
```

29.8.5.178 typedef TPM2B_DERIVE

```
typedef struct TPM2B_DERIVE TPM2B_DERIVE;
```

29.8.5.179 typedef TPMU_SENSITIVE_CREATE

```
typedef union TPMU_SENSITIVE_CREATE TPMU_SENSITIVE_CREATE;
```

29.8.5.180 typedef TPM2B_SENSITIVE_DATA

```
typedef struct TPM2B_SENSITIVE_DATA TPM2B_SENSITIVE_DATA;
```

29.8.5.181 typedef TPMS_SENSITIVE_CREATE

```
typedef struct TPMS_SENSITIVE_CREATE TPMS_SENSITIVE_CREATE;
```

29.8.5.182 typedef TPM2B_SENSITIVE_CREATE

```
typedef struct TPM2B_SENSITIVE_CREATE TPM2B_SENSITIVE_CREATE;
```

29.8.5.183 typedef TPMS_SCHEME_HASH

```
typedef struct TPMS_SCHEME_HASH TPMS_SCHEME_HASH;
```

29.8.5.184 typedef TPMS_SCHEME_ECDAA

```
typedef struct TPMS_SCHEME_ECDAA TPMS_SCHEME_ECDAA;
```

29.8.5.185 typedef TPMI_ALG_KEYEDHASH_SCHEME

```
typedef TPM_ALG_ID TPMI_ALG_KEYEDHASH_SCHEME;
```

29.8.5.186 typedef TPMS_SCHEME_HMAC

```
typedef TPMS_SCHEME_HASH TPMS_SCHEME_HMAC;
```

29.8.5.187 typedef TPMS_SCHEME_XOR

```
typedef struct TPMS_SCHEME_XOR TPMS_SCHEME_XOR;
```

29.8.5.188 typedef TPMU_SCHEME_KEYEDHASH

```
typedef union TPMU_SCHEME_KEYEDHASH TPMU_SCHEME_KEYEDHASH;
```

29.8.5.189 typedef TPMT_KEYEDHASH_SCHEME

```
typedef struct TPMT_KEYEDHASH_SCHEME TPMT_KEYEDHASH_SCHEME;
```

29.8.5.190 typedef TPMS_SIG_SCHEME_RSASSA

```
typedef TPMS_SCHEME_HASH TPMS_SIG_SCHEME_RSASSA;
```

29.8.5.191 typedef TPMS_SIG_SCHEME_RSAPSS

```
typedef TPMS_SCHEME_HASH TPMS_SIG_SCHEME_RSAPSS;
```

29.8.5.192 typedef TPMS_SIG_SCHEME_ECDSA

```
typedef TPMS_SCHEME_HASH TPMS_SIG_SCHEME_ECDSA;
```

29.8.5.193 typedef TPMS_SIG_SCHEME_ECDAA

```
typedef TPMS_SCHEME_ECDAA TPMS_SIG_SCHEME_ECDAA;
```

29.8.5.194 typedef TPMU_SIG_SCHEME

```
typedef union TPMU_SIG_SCHEME TPMU_SIG_SCHEME;
```

29.8.5.195 typedef TPMT_SIG_SCHEME

```
typedef struct TPMT_SIG_SCHEME TPMT_SIG_SCHEME;
```

29.8.5.196 typedef TPMS_ENC_SCHEME_OAEP

```
typedef TPMS_SCHEME_HASH TPMS_ENC_SCHEME_OAEP;
```

29.8.5.197 typedef TPMS_KEY_SCHEME_ECDH

```
typedef TPMS_SCHEME_HASH TPMS_KEY_SCHEME_ECDH;
```

29.8.5.198 typedef TPMS_KEY_SCHEME_ECMQV

```
typedef TPMS_SCHEME_HASH TPMS_KEY_SCHEME_ECMQV;
```

29.8.5.199 typedef TPMS_SCHEME_MGF1

```
typedef TPMS_SCHEME_HASH TPMS_SCHEME_MGF1;
```

29.8.5.200 typedef TPMS_SCHEME_KDF1_SP800_56A

```
typedef TPMS_SCHEME_HASH TPMS_SCHEME_KDF1_SP800_56A;
```

29.8.5.201 typedef TPMS_SCHEME_KDF2

```
typedef TPMS_SCHEME_HASH TPMS_SCHEME_KDF2;
```

29.8.5.202 typedef TPMS_SCHEME_KDF1_SP800_108

```
typedef TPMS_SCHEME_HASH TPMS_SCHEME_KDF1_SP800_108;
```

29.8.5.203 typedef TPMU_KDF_SCHEME

```
typedef union TPMU_KDF_SCHEME TPMU_KDF_SCHEME;
```

29.8.5.204 typedef TPMT_KDF_SCHEME

```
typedef struct TPMT_KDF_SCHEME TPMT_KDF_SCHEME;
```

29.8.5.205 typedef TPMI_ALG_ASYM_SCHEME

```
typedef TPM_ALG_ID TPMI_ALG_ASYM_SCHEME;
```

29.8.5.206 typedef TPMU_ASYM_SCHEME

```
typedef union TPMU_ASYM_SCHEME TPMU_ASYM_SCHEME;
```

29.8.5.207 typedef TPMT_ASYM_SCHEME

```
typedef struct TPMT_ASYM_SCHEME TPMT_ASYM_SCHEME;
```

29.8.5.208 typedef TPMI_ALG_RSA_SCHEME

```
typedef TPM_ALG_ID TPMI_ALG_RSA_SCHEME;
```

29.8.5.209 typedef TPMT_RSA_SCHEME

```
typedef struct TPMT_RSA_SCHEME TPMT_RSA_SCHEME;
```

29.8.5.210 typedef TPMI_ALG_RSA_DECRYPT

```
typedef TPM_ALG_ID TPMI_ALG_RSA_DECRYPT;
```

29.8.5.211 typedef TPMT_RSA_DECRYPT

```
typedef struct TPMT_RSA_DECRYPT TPMT_RSA_DECRYPT;
```

29.8.5.212 typedef TPM2B_PUBLIC_KEY_RSA

```
typedef struct TPM2B_PUBLIC_KEY_RSA TPM2B_PUBLIC_KEY_RSA;
```

29.8.5.213 typedef TPMI_RSA_KEY_BITS

```
typedef TPM_KEY_BITS TPMI_RSA_KEY_BITS;
```

29.8.5.214 typedef TPM2B_PRIVATE_KEY_RSA

```
typedef struct TPM2B_PRIVATE_KEY_RSA TPM2B_PRIVATE_KEY_RSA;
```

29.8.5.215 typedef TPM2B_ECC_PARAMETER

```
typedef struct TPM2B_ECC_PARAMETER TPM2B_ECC_PARAMETER;
```

29.8.5.216 typedef TPMS_ECC_POINT

```
typedef struct TPMS_ECC_POINT TPMS_ECC_POINT;
```

29.8.5.217 typedef TPM2B_ECC_POINT

```
typedef struct TPM2B_ECC_POINT TPM2B_ECC_POINT;
```

29.8.5.218 typedef TPMI_ALG_ECC_SCHEME

```
typedef TPM_ALG_ID TPMI_ALG_ECC_SCHEME;
```

29.8.5.219 typedef TPMI_ECC_CURVE

```
typedef TPM_ECC_CURVE TPMI_ECC_CURVE;
```

29.8.5.220 typedef TPMT_ECC_SCHEME

```
typedef TPMT_SIG_SCHEME TPMT_ECC_SCHEME;
```

29.8.5.221 typedef TPMS_ALGORITHM_DETAIL_ECC

```
typedef struct TPMS_ALGORITHM_DETAIL_ECC TPMS_ALGORITHM_DETAIL_ECC;
```

29.8.5.222 typedef TPMS_SIGNATURE_RSA

```
typedef struct TPMS_SIGNATURE_RSA TPMS_SIGNATURE_RSA;
```

29.8.5.223 typedef TPMS_SIGNATURE_RSASSA

```
typedef TPMS_SIGNATURE_RSA TPMS_SIGNATURE_RSASSA;
```

29.8.5.224 typedef TPMS_SIGNATURE_RSAPSS

```
typedef TPMS_SIGNATURE_RSA TPMS_SIGNATURE_RSAPSS;
```

29.8.5.225 typedef TPMS_SIGNATURE_ECC

```
typedef struct TPMS_SIGNATURE_ECC TPMS_SIGNATURE_ECC;
```

29.8.5.226 typedef TPMS_SIGNATURE_ECDSA

```
typedef TPMS_SIGNATURE_ECC TPMS_SIGNATURE_ECDSA;
```

29.8.5.227 typedef TPMS_SIGNATURE_ECDAA

```
typedef TPMS_SIGNATURE_ECC TPMS_SIGNATURE_ECDAA;
```

29.8.5.228 typedef TPMS_SIGNATURE_HASH_MLDSA

```
typedef struct TPMS_SIGNATURE_HASH_MLDSA TPMS_SIGNATURE_HASH_MLDSA;
```

29.8.5.229 typedef TPMU_SIGNATURE

```
typedef union TPMU_SIGNATURE TPMU_SIGNATURE;
```

29.8.5.230 typedef TPMT_SIGNATURE

```
typedef struct TPMT_SIGNATURE TPMT_SIGNATURE;
```

29.8.5.231 typedef TPMU_ENCRYPTED_SECRET

```
typedef union TPMU_ENCRYPTED_SECRET TPMU_ENCRYPTED_SECRET;
```

29.8.5.232 typedef TPM2B_ENCRYPTED_SECRET

```
typedef struct TPM2B_ENCRYPTED_SECRET TPM2B_ENCRYPTED_SECRET;
```

29.8.5.233 typedef TPMI_ALG_PUBLIC

```
typedef TPM_ALG_ID TPMI_ALG_PUBLIC;
```

29.8.5.234 typedef TPMU_PUBLIC_ID

```
typedef union TPMU_PUBLIC_ID TPMU_PUBLIC_ID;
```

29.8.5.235 typedef TPMS_KEYEDHASH_PARMS

```
typedef struct TPMS_KEYEDHASH_PARMS TPMS_KEYEDHASH_PARMS;
```

29.8.5.236 typedef TPMS_ASYM_PARMS

```
typedef struct TPMS_ASYM_PARMS TPMS_ASYM_PARMS;
```

29.8.5.237 typedef TPMS_RSA_PARMS

```
typedef struct TPMS_RSA_PARMS TPMS_RSA_PARMS;
```

29.8.5.238 typedef TPMS_ECC_PARMS

```
typedef struct TPMS_ECC_PARMS TPMS_ECC_PARMS;
```

29.8.5.239 typedef TPMS_MLDSA_PARMS

```
typedef struct TPMS_MLDSA_PARMS TPMS_MLDSA_PARMS;
```

29.8.5.240 typedef TPMS_HASH_MLDSA_PARMS

```
typedef struct TPMS_HASH_MLDSA_PARMS TPMS_HASH_MLDSA_PARMS;
```

29.8.5.241 typedef TPMS_MLKEM_PARMS

```
typedef struct TPMS_MLKEM_PARMS TPMS_MLKEM_PARMS;
```

29.8.5.242 typedef TPMU_PUBLIC_PARMS

```
typedef union TPMU_PUBLIC_PARMS TPMU_PUBLIC_PARMS;
```

29.8.5.243 typedef TPMT_PUBLIC_PARMS

```
typedef struct TPMT_PUBLIC_PARMS TPMT_PUBLIC_PARMS;
```

29.8.5.244 typedef TPMT_PUBLIC

```
typedef struct TPMT_PUBLIC TPMT_PUBLIC;
```

29.8.5.245 typedef TPM2B_PUBLIC

```
typedef struct TPM2B_PUBLIC TPM2B_PUBLIC;
```

29.8.5.246 typedef TPM2B_TEMPLATE

```
typedef struct TPM2B_TEMPLATE TPM2B_TEMPLATE;
```

29.8.5.247 typedef TPM2B_PRIVATE_VENDOR_SPECIFIC

```
typedef struct TPM2B_PRIVATE_VENDOR_SPECIFIC TPM2B_PRIVATE_VENDOR_SPECIFIC;
```

29.8.5.248 typedef TPMU_SENSITIVE_COMPOSITE

```
typedef union TPMU_SENSITIVE_COMPOSITE TPMU_SENSITIVE_COMPOSITE;
```

29.8.5.249 typedef TPMT_SENSITIVE

```
typedef struct TPMT_SENSITIVE TPMT_SENSITIVE;
```

29.8.5.250 typedef TPM2B_SENSITIVE

```
typedef struct TPM2B_SENSITIVE TPM2B_SENSITIVE;
```

29.8.5.251 typedef TPMT_PRIVATE

```
typedef struct TPMT_PRIVATE TPMT_PRIVATE;
```

29.8.5.252 typedef TPM2B_PRIVATE

```
typedef struct TPM2B_PRIVATE TPM2B_PRIVATE;
```

29.8.5.253 typedef TPMS_ID_OBJECT

```
typedef struct TPMS_ID_OBJECT TPMS_ID_OBJECT;
```

29.8.5.254 typedef TPM2B_ID_OBJECT

```
typedef struct TPM2B_ID_OBJECT TPM2B_ID_OBJECT;
```

29.8.5.255 typedef TPM_NV_INDEX

```
typedef UINT32 TPM_NV_INDEX;
```

29.8.5.256 typedef TPM_NT

```
typedef enum TPM_NT TPM_NT;
```

29.8.5.257 typedef TPMS_NV_PIN_COUNTER_PARAMETERS

```
typedef struct TPMS_NV_PIN_COUNTER_PARAMETERS TPMS_NV_PIN_COUNTER_PARAMETERS;
```

29.8.5.258 typedef TPMA_NV

```
typedef UINT32 TPMA_NV;
```

29.8.5.259 typedef TPMS_NV_PUBLIC

```
typedef struct TPMS_NV_PUBLIC TPMS_NV_PUBLIC;
```

29.8.5.260 typedef TPM2B_NV_PUBLIC

```
typedef struct TPM2B_NV_PUBLIC TPM2B_NV_PUBLIC;
```

29.8.5.261 typedef TPM2B_CONTEXT_SENSITIVE

```
typedef struct TPM2B_CONTEXT_SENSITIVE TPM2B_CONTEXT_SENSITIVE;
```

29.8.5.262 typedef TPMS_CONTEXT_DATA

```
typedef struct TPMS_CONTEXT_DATA TPMS_CONTEXT_DATA;
```

29.8.5.263 typedef TPM2B_CONTEXT_DATA

```
typedef struct TPM2B_CONTEXT_DATA TPM2B_CONTEXT_DATA;
```

29.8.5.264 typedef TPMS_CONTEXT

```
typedef struct TPMS_CONTEXT TPMS_CONTEXT;
```

29.8.5.265 typedef TPMS_CREATION_DATA

```
typedef struct TPMS_CREATION_DATA TPMS_CREATION_DATA;
```

29.8.5.266 typedef TPM2B_CREATION_DATA

```
typedef struct TPM2B_CREATION_DATA TPM2B_CREATION_DATA;
```

29.8.5.267 typedef TPMS_AUTH_COMMAND

```
typedef struct TPMS_AUTH_COMMAND TPMS_AUTH_COMMAND;
```

29.8.5.268 typedef TPMS_AUTH_RESPONSE

```
typedef struct TPMS_AUTH_RESPONSE TPMS_AUTH_RESPONSE;
```

29.8.5.269 typedef TPM2_AUTH_SESSION

```
typedef struct TPM2_AUTH_SESSION TPM2_AUTH_SESSION;
```

29.8.5.270 typedef TPM2HalIoCb

```
typedef int(* TPM2HalIoCb)(struct TPM2_CTX *, const BYTE *txBuf, BYTE *rxBuf,
 ↪  UINT16 xferSz, void *userCtx);
```

29.8.5.271 typedef TPM2_CTX

```
typedef struct TPM2_CTX TPM2_CTX;
```

29.8.5.272 typedef ChangePPS_In

```
typedef ChangeSeed_In ChangePPS_In;
```

29.8.5.273 typedef ChangeEPS_In

```
typedef ChangeSeed_In ChangeEPS_In;
```

29.8.5.274 typedef TPM_MODE_SET

```
typedef struct TPM_MODE_SET TPM_MODE_SET;
```

29.8.5.275 typedef GetRandom2_In

```
typedef GetRandom_In GetRandom2_In;
```

29.8.5.276 typedef TPMI_GPIO_NAME

```
typedef UINT32 TPMI_GPIO_NAME;
```

29.8.5.277 typedef TPMI_GPIO_MODE

```
typedef UINT32 TPMI_GPIO_MODE;
```

29.8.5.278 typedef TPMS_GPIO_CONFIG

```
typedef struct TPMS_GPIO_CONFIG TPMS_GPIO_CONFIG;
```

29.8.5.279 typedef TPML_GPIO_CONFIG

```
typedef struct TPML_GPIO_CONFIG TPML_GPIO_CONFIG;
```

29.8.6 Functions Documentation

29.8.6.1 function TPM2_Startup

```
WOLFTPM_API TPM_RC TPM2_Startup(
    Startup_In * in
)
```

29.8.6.2 function TPM2_Shutdown

```
WOLFTPM_API TPM_RC TPM2_Shutdown(
    Shutdown_In * in
)
```

29.8.6.3 function TPM2_GetCapability

```
WOLFTPM_API TPM_RC TPM2_GetCapability(
    GetCapability_In * in,
    GetCapability_Out * out
)
```

29.8.6.4 function TPM2_SetCapability

```
WOLFTPM_API TPM_RC TPM2_SetCapability(
    SetCapability_In * in
)
```

29.8.6.5 function TPM2_SelfTest

```
WOLFTPM_API TPM_RC TPM2_SelfTest(
    SelfTest_In * in
)
```

29.8.6.6 function TPM2_IncrementalSelfTest

```
WOLFTPM_API TPM_RC TPM2_IncrementalSelfTest(
    IncrementalSelfTest_In * in,
    IncrementalSelfTest_Out * out
)
```

29.8.6.7 function TPM2_GetTestResult

```
WOLFTPM_API TPM_RC TPM2_GetTestResult(
    GetTestResult_Out * out
)
```

29.8.6.8 function TPM2_GetRandom

```
WOLFTPM_API TPM_RC TPM2_GetRandom(
    GetRandom_In * in,
    GetRandom_Out * out
)
```

29.8.6.9 function TPM2_StirRandom

```
WOLFTPM_API TPM_RC TPM2_StirRandom(
    StirRandom_In * in
)
```

29.8.6.10 function TPM2_PCR_Read

```
WOLFTPM_API TPM_RC TPM2_PCR_Read(
    PCR_Read_In * in,
    PCR_Read_Out * out
)
```

29.8.6.11 function TPM2_PCR_Extend

```
WOLFTPM_API TPM_RC TPM2_PCR_Extend(
    PCR_Extend_In * in
)
```

29.8.6.12 function TPM2_Create

```
WOLFTPM_API TPM_RC TPM2_Create(
    Create_In * in,
    Create_Out * out
)
```

29.8.6.13 function TPM2_CreateLoaded

```
WOLFTPM_API TPM_RC TPM2_CreateLoaded(
    CreateLoaded_In * in,
    CreateLoaded_Out * out
)
```

29.8.6.14 function TPM2_CreatePrimary

```
WOLFTPM_API TPM_RC TPM2_CreatePrimary(
    CreatePrimary_In * in,
    CreatePrimary_Out * out
)
```

29.8.6.15 function TPM2_Load

```
WOLFTPM_API TPM_RC TPM2_Load(
    Load_In * in,
    Load_Out * out
)
```

29.8.6.16 function TPM2_FlushContext

```
WOLFTPM_API TPM_RC TPM2_FlushContext(
    FlushContext_In * in
)
```

29.8.6.17 function TPM2_Unseal

```
WOLFTPM_API TPM_RC TPM2_Unseal(
    Unseal_In * in,
    Unseal_Out * out
)
```

29.8.6.18 function TPM2_StartAuthSession

```
WOLFTPM_API TPM_RC TPM2_StartAuthSession(
    StartAuthSession_In * in,
    StartAuthSession_Out * out
)
```

29.8.6.19 function TPM2_PolicyRestart

```
WOLFTPM_API TPM_RC TPM2_PolicyRestart(
    PolicyRestart_In * in
)
```

29.8.6.20 function TPM2_LoadExternal

```
WOLFTPM_API TPM_RC TPM2_LoadExternal(
    LoadExternal_In * in,
    LoadExternal_Out * out
)
```

29.8.6.21 function TPM2_ReadPublic

```
WOLFTPM_API TPM_RC TPM2_ReadPublic(
    ReadPublic_In * in,
    ReadPublic_Out * out
)
```

29.8.6.22 function TPM2_ActivateCredential

```
WOLFTPM_API TPM_RC TPM2_ActivateCredential(
    ActivateCredential_In * in,
    ActivateCredential_Out * out
)
```

29.8.6.23 function TPM2_MakeCredential

```
WOLFTPM_API TPM_RC TPM2_MakeCredential(
    MakeCredential_In * in,
    MakeCredential_Out * out
)
```

29.8.6.24 function TPM2_ObjectChangeAuth

```
WOLFTPM_API TPM_RC TPM2_ObjectChangeAuth(
    ObjectChangeAuth_In * in,
    ObjectChangeAuth_Out * out
)
```

29.8.6.25 function TPM2_Duplicate

```
WOLFTPM_API TPM_RC TPM2_Duplicate(
    Duplicate_In * in,
    Duplicate_Out * out
)
```

29.8.6.26 function TPM2_Rewrap

```
WOLFTPM_API TPM_RC TPM2_Rewrap(
    Rewrap_In * in,
    Rewrap_Out * out
)
```

29.8.6.27 function TPM2_Import

```
WOLFTPM_API TPM_RC TPM2_Import(
    Import_In * in,
    Import_Out * out
)
```

29.8.6.28 function TPM2_RSA_Encrypt

```
WOLFTPM_API TPM_RC TPM2_RSA_Encrypt(
    RSA_Encrypt_In * in,
    RSA_Encrypt_Out * out
)
```

29.8.6.29 function TPM2_RSA_Decrypt

```
WOLFTPM_API TPM_RC TPM2_RSA_Decrypt(
    RSA_Decrypt_In * in,
    RSA_Decrypt_Out * out
)
```

29.8.6.30 function TPM2_ECDH_KeyGen

```
WOLFTPM_API TPM_RC TPM2_ECDH_KeyGen(
    ECDH_KeyGen_In * in,
    ECDH_KeyGen_Out * out
)
```

29.8.6.31 function TPM2_ECDH_ZGen

```
WOLFTPM_API TPM_RC TPM2_ECDH_ZGen(
    ECDH_ZGen_In * in,
    ECDH_ZGen_Out * out
)
```

29.8.6.32 function TPM2_ECC_Parameters

```
WOLFTPM_API TPM_RC TPM2_ECC_Parameters(
    ECC_Parameters_In * in,
    ECC_Parameters_Out * out
)
```

29.8.6.33 function TPM2_ZGen_2Phase

```
WOLFTPM_API TPM_RC TPM2_ZGen_2Phase(
    ZGen_2Phase_In * in,
    ZGen_2Phase_Out * out
)
```

29.8.6.34 function TPM2_EncryptDecrypt

```
WOLFTPM_API TPM_RC TPM2_EncryptDecrypt(
    EncryptDecrypt_In * in,
    EncryptDecrypt_Out * out
)
```

29.8.6.35 function TPM2_EncryptDecrypt2

```
WOLFTPM_API TPM_RC TPM2_EncryptDecrypt2(
    EncryptDecrypt2_In * in,
    EncryptDecrypt2_Out * out
)
```

29.8.6.36 function TPM2_Hash

```
WOLFTPM_API TPM_RC TPM2_Hash(
    Hash_In * in,
    Hash_Out * out
)
```

29.8.6.37 function TPM2_HMAC

```
WOLFTPM_API TPM_RC TPM2_HMAC(
    HMAC_In * in,
    HMAC_Out * out
)
```

29.8.6.38 function TPM2_HMAC_Start

```
WOLFTPM_API TPM_RC TPM2_HMAC_Start(
    HMAC_Start_In * in,
    HMAC_Start_Out * out
)
```

29.8.6.39 function TPM2_HashSequenceStart

```
WOLFTPM_API TPM_RC TPM2_HashSequenceStart(
    HashSequenceStart_In * in,
    HashSequenceStart_Out * out
)
```

29.8.6.40 function TPM2_SequenceUpdate

```
WOLFTPM_API TPM_RC TPM2_SequenceUpdate(
    SequenceUpdate_In * in
)
```

29.8.6.41 function TPM2_SequenceComplete

```
WOLFTPM_API TPM_RC TPM2_SequenceComplete(
    SequenceComplete_In * in,
    SequenceComplete_Out * out
)
```

29.8.6.42 function TPM2_EventSequenceComplete

```
WOLFTPM_API TPM_RC TPM2_EventSequenceComplete(
    EventSequenceComplete_In * in,
    EventSequenceComplete_Out * out
)
```

29.8.6.43 function TPM2_Certify

```
WOLFTPM_API TPM_RC TPM2_Certify(
    Certify_In * in,
    Certify_Out * out
)
```

29.8.6.44 function TPM2_CertifyCreation

```
WOLFTPM_API TPM_RC TPM2_CertifyCreation(
    CertifyCreation_In * in,
    CertifyCreation_Out * out
)
```

29.8.6.45 function TPM2_Quote

```
WOLFTPM_API TPM_RC TPM2_Quote(
    Quote_In * in,
    Quote_Out * out
)
```

29.8.6.46 function TPM2_GetSessionAuditDigest

```
WOLFTPM_API TPM_RC TPM2_GetSessionAuditDigest(
    GetSessionAuditDigest_In * in,
    GetSessionAuditDigest_Out * out
)
```

29.8.6.47 function TPM2_GetCommandAuditDigest

```
WOLFTPM_API TPM_RC TPM2_GetCommandAuditDigest(
    GetCommandAuditDigest_In * in,
    GetCommandAuditDigest_Out * out
)
```

29.8.6.48 function TPM2_GetTime

```
WOLFTPM_API TPM_RC TPM2_GetTime(
    GetTime_In * in,
    GetTime_Out * out
)
```

29.8.6.49 function TPM2_Commit

```
WOLFTPM_API TPM_RC TPM2_Commit(
    Commit_In * in,
    Commit_Out * out
)
```

29.8.6.50 function TPM2_EC_Ephemeral

```
WOLFTPM_API TPM_RC TPM2_EC_Ephemeral(
    EC_Ephemeral_In * in,
    EC_Ephemeral_Out * out
)
```

29.8.6.51 function TPM2_VerifySignature

```
WOLFTPM_API TPM_RC TPM2_VerifySignature(
    VerifySignature_In * in,
    VerifySignature_Out * out
)
```

29.8.6.52 function TPM2_Sign

```
WOLFTPM_API TPM_RC TPM2_Sign(
    Sign_In * in,
    Sign_Out * out
)
```

29.8.6.53 function TPM2_SignSequenceStart

```
WOLFTPM_API TPM_RC TPM2_SignSequenceStart(
    SignSequenceStart_In * in,
    SignSequenceStart_Out * out
)
```

29.8.6.54 function TPM2_VerifySequenceStart

```
WOLFTPM_API TPM_RC TPM2_VerifySequenceStart(
    VerifySequenceStart_In * in,
    VerifySequenceStart_Out * out
)
```

29.8.6.55 function TPM2_SignSequenceComplete

```
WOLFTPM_API TPM_RC TPM2_SignSequenceComplete(
    SignSequenceComplete_In * in,
    SignSequenceComplete_Out * out
)
```

29.8.6.56 function TPM2_VerifySequenceComplete

```
WOLFTPM_API TPM_RC TPM2_VerifySequenceComplete(
    VerifySequenceComplete_In * in,
    VerifySequenceComplete_Out * out
)
```

29.8.6.57 function TPM2_SignDigest

```
WOLFTPM_API TPM_RC TPM2_SignDigest(
    SignDigest_In * in,
    SignDigest_Out * out
)
```

29.8.6.58 function TPM2_VerifyDigestSignature

```
WOLFTPM_API TPM_RC TPM2_VerifyDigestSignature(
    VerifyDigestSignature_In * in,
    VerifyDigestSignature_Out * out
)
```

29.8.6.59 function TPM2_Encapsulate

```
WOLFTPM_API TPM_RC TPM2_Encapsulate(
    Encapsulate_In * in,
    Encapsulate_Out * out
)
```

29.8.6.60 function TPM2_Decapsulate

```
WOLFTPM_API TPM_RC TPM2_Decapsulate(
    Decapsulate_In * in,
    Decapsulate_Out * out
)
```

29.8.6.61 function TPM2_SetCommandCodeAuditStatus

```
WOLFTPM_API TPM_RC TPM2_SetCommandCodeAuditStatus(
    SetCommandCodeAuditStatus_In * in
)
```

29.8.6.62 function TPM2_PCR_Event

```
WOLFTPM_API TPM_RC TPM2_PCR_Event(
    PCR_Event_In * in,
    PCR_Event_Out * out
)
```

29.8.6.63 function TPM2_PCR_Allocate

```
WOLFTPM_API TPM_RC TPM2_PCR_Allocate(
    PCR_Allocate_In * in,
    PCR_Allocate_Out * out
)
```

Set which PCR banks the TPM allocates.

Parameters:

- **in** pointer to a PCR_Allocate_In struct
- **out** pointer to a PCR_Allocate_Out struct

See:

- wolfTPM2_AllocatePCRBanks
- TPM2_IsPcrBankAllocated

Return:

- TPM_RC_SUCCESS: the TPM processed the request - check out->allocationSuccess, which is NO when the TPM lacks the space
- TPM_RC_HIERARCHY: the platform hierarchy is disabled

- BAD_FUNC_ARG: check the provided arguments, or no session is set

Note:

- The selection REPLACES the current allocation - banks not named in it are deallocated. Algorithms the TPM does not implement are silently ignored (TPM 2.0 Part 3 22.5), so a selection naming only unimplemented algorithms can leave the TPM with no PCR banks at all. Prefer wolfTPM2_AllocatePCRBanks, which pre-checks each algorithm and refuses an empty result.
- The change takes effect at the next Startup(CLEAR), not on return.
- Requires an active session (TPM2_SetAuthPassword or wolfTPM2_SetAuthSession on slot 0) and the platform hierarchy; without one this returns BAD_FUNC_ARG rather than a TPM response code.

29.8.6.64 function TPM2_PCR_SetAuthPolicy

```
WOLFTPM_API TPM_RC TPM2_PCR_SetAuthPolicy(
    PCR_SetAuthPolicy_In * in
)
```

29.8.6.65 function TPM2_PCR_SetAuthValue

```
WOLFTPM_API TPM_RC TPM2_PCR_SetAuthValue(
    PCR_SetAuthValue_In * in
)
```

29.8.6.66 function TPM2_PCR_Reset

```
WOLFTPM_API TPM_RC TPM2_PCR_Reset(
    PCR_Reset_In * in
)
```

29.8.6.67 function TPM2_PolicySigned

```
WOLFTPM_API TPM_RC TPM2_PolicySigned(
    PolicySigned_In * in,
    PolicySigned_Out * out
)
```

29.8.6.68 function TPM2_PolicySecret

```
WOLFTPM_API TPM_RC TPM2_PolicySecret(
    PolicySecret_In * in,
    PolicySecret_Out * out
)
```

29.8.6.69 function TPM2_PolicyTicket

```
WOLFTPM_API TPM_RC TPM2_PolicyTicket(
    PolicyTicket_In * in
)
```

29.8.6.70 function TPM2_PolicyOR

```
WOLFTPM_API TPM_RC TPM2_PolicyOR(
    PolicyOR_In * in
)
```

29.8.6.71 function TPM2_PolicyPCR

```
WOLFTPM_API TPM_RC TPM2_PolicyPCR(
    PolicyPCR_In * in
)
```

29.8.6.72 function TPM2_PolicyTransportSPDM

```
WOLFTPM_API TPM_RC TPM2_PolicyTransportSPDM(
    PolicyTransportSPDM_In * in
)
```

29.8.6.73 function TPM2_PolicyLocality

```
WOLFTPM_API TPM_RC TPM2_PolicyLocality(
    PolicyLocality_In * in
)
```

29.8.6.74 function TPM2_PolicyNV

```
WOLFTPM_API TPM_RC TPM2_PolicyNV(
    PolicyNV_In * in
)
```

29.8.6.75 function TPM2_PolicyCounterTimer

```
WOLFTPM_API TPM_RC TPM2_PolicyCounterTimer(
    PolicyCounterTimer_In * in
)
```

29.8.6.76 function TPM2_PolicyCommandCode

```
WOLFTPM_API TPM_RC TPM2_PolicyCommandCode(
    PolicyCommandCode_In * in
)
```

29.8.6.77 function TPM2_PolicyPhysicalPresence

```
WOLFTPM_API TPM_RC TPM2_PolicyPhysicalPresence(
    PolicyPhysicalPresence_In * in
)
```

29.8.6.78 function TPM2_PolicyCpHash

```
WOLFTPM_API TPM_RC TPM2_PolicyCpHash(
    PolicyCpHash_In * in
)
```

29.8.6.79 function TPM2_PolicyNameHash

```
WOLFTPM_API TPM_RC TPM2_PolicyNameHash(
    PolicyNameHash_In * in
)
```

29.8.6.80 function TPM2_PolicyDuplicationSelect

```
WOLFTPM_API TPM_RC TPM2_PolicyDuplicationSelect(
    PolicyDuplicationSelect_In * in
)
```

29.8.6.81 function TPM2_PolicyAuthorize

```
WOLFTPM_API TPM_RC TPM2_PolicyAuthorize(
    PolicyAuthorize_In * in
)
```

29.8.6.82 function TPM2_PolicyAuthValue

```
WOLFTPM_API TPM_RC TPM2_PolicyAuthValue(
    PolicyAuthValue_In * in
)
```

29.8.6.83 function TPM2_PolicyPassword

```
WOLFTPM_API TPM_RC TPM2_PolicyPassword(
    PolicyPassword_In * in
)
```

29.8.6.84 function TPM2_PolicyGetDigest

```
WOLFTPM_API TPM_RC TPM2_PolicyGetDigest(
    PolicyGetDigest_In * in,
    PolicyGetDigest_Out * out
)
```

29.8.6.85 function TPM2_PolicyNvWritten

```
WOLFTPM_API TPM_RC TPM2_PolicyNvWritten(
    PolicyNvWritten_In * in
)
```

29.8.6.86 function TPM2_PolicyTemplate

```
WOLFTPM_API TPM_RC TPM2_PolicyTemplate(
    PolicyTemplate_In * in
)
```

29.8.6.87 function TPM2_PolicyAuthorizeNV

```
WOLFTPM_API TPM_RC TPM2_PolicyAuthorizeNV(
    PolicyAuthorizeNV_In * in
)
```

29.8.6.88 function _TPM_Hash_Start

```
WOLFTPM_API void _TPM_Hash_Start(
    void
)
```

29.8.6.89 function _TPM_Hash_Data

```
WOLFTPM_API void _TPM_Hash_Data(
    UINT32 dataSize,
    BYTE * data
)
```

29.8.6.90 function _TPM_Hash_End

```
WOLFTPM_API void _TPM_Hash_End(
    void
)
```

29.8.6.91 function TPM2_HierarchyControl

```
WOLFTPM_API TPM_RC TPM2_HierarchyControl(
    HierarchyControl_In * in
)
```

29.8.6.92 function TPM2_SetPrimaryPolicy

```
WOLFTPM_API TPM_RC TPM2_SetPrimaryPolicy(
    SetPrimaryPolicy_In * in
)
```

29.8.6.93 function TPM2_ChangePPS

```
WOLFTPM_API TPM_RC TPM2_ChangePPS(
    ChangePPS_In * in
)
```

29.8.6.94 function TPM2_ChangeEPS

```
WOLFTPM_API TPM_RC TPM2_ChangeEPS(
    ChangeEPS_In * in
)
```

29.8.6.95 function TPM2_Clear

```
WOLFTPM_API TPM_RC TPM2_Clear(
    Clear_In * in
)
```

29.8.6.96 function TPM2_ClearControl

```
WOLFTPM_API TPM_RC TPM2_ClearControl(
    ClearControl_In * in
)
```

29.8.6.97 function TPM2_HierarchyChangeAuth

```
WOLFTPM_API TPM_RC TPM2_HierarchyChangeAuth(
    HierarchyChangeAuth_In * in
)
```

29.8.6.98 function TPM2_DictionaryAttackLockReset

```
WOLFTPM_API TPM_RC TPM2_DictionaryAttackLockReset(
    DictionaryAttackLockReset_In * in
)
```

29.8.6.99 function TPM2_DictionaryAttackParameters

```
WOLFTPM_API TPM_RC TPM2_DictionaryAttackParameters(
    DictionaryAttackParameters_In * in
)
```

29.8.6.100 function TPM2_PP_Commands

```
WOLFTPM_API TPM_RC TPM2_PP_Commands(
    PP_Commands_In * in
)
```

29.8.6.101 function TPM2_SetAlgorithmSet

```
WOLFTPM_API TPM_RC TPM2_SetAlgorithmSet(
    SetAlgorithmSet_In * in
)
```

29.8.6.102 function TPM2_FieldUpgradeStart

```
WOLFTPM_API TPM_RC TPM2_FieldUpgradeStart(
    FieldUpgradeStart_In * in
)
```

29.8.6.103 function TPM2_FieldUpgradeData

```
WOLFTPM_API TPM_RC TPM2_FieldUpgradeData(
    FieldUpgradeData_In * in,
    FieldUpgradeData_Out * out
)
```

29.8.6.104 function TPM2_FirmwareRead

```
WOLFTPM_API TPM_RC TPM2_FirmwareRead(
    FirmwareRead_In * in,
    FirmwareRead_Out * out
)
```

29.8.6.105 function TPM2_ContextSave

```
WOLFTPM_API TPM_RC TPM2_ContextSave(
    ContextSave_In * in,
    ContextSave_Out * out
)
```

29.8.6.106 function TPM2_ContextLoad

```
WOLFTPM_API TPM_RC TPM2_ContextLoad(
    ContextLoad_In * in,
    ContextLoad_Out * out
)
```

29.8.6.107 function TPM2_EvictControl

```
WOLFTPM_API TPM_RC TPM2_EvictControl(
    EvictControl_In * in
)
```

29.8.6.108 function TPM2_ReadClock

```
WOLFTPM_API TPM_RC TPM2_ReadClock(
    ReadClock_Out * out
)
```

29.8.6.109 function TPM2_ClockSet

```
WOLFTPM_API TPM_RC TPM2_ClockSet(
    ClockSet_In * in
)
```

29.8.6.110 function TPM2_ClockRateAdjust

```
WOLFTPM_API TPM_RC TPM2_ClockRateAdjust(
    ClockRateAdjust_In * in
)
```

29.8.6.111 function TPM2_TestParms

```
WOLFTPM_API TPM_RC TPM2_TestParms(
    TestParms_In * in
)
```

29.8.6.112 function TPM2_NV_DefineSpace

```
WOLFTPM_API TPM_RC TPM2_NV_DefineSpace(
    NV_DefineSpace_In * in
)
```

29.8.6.113 function TPM2_NV_UndefineSpace

```
WOLFTPM_API TPM_RC TPM2_NV_UndefineSpace(
    NV_UndefineSpace_In * in
)
```

29.8.6.114 function TPM2_NV_UndefineSpaceSpecial

```
WOLFTPM_API TPM_RC TPM2_NV_UndefineSpaceSpecial(
    NV_UndefineSpaceSpecial_In * in
)
```

29.8.6.115 function TPM2_NV_ReadPublic

```
WOLFTPM_API TPM_RC TPM2_NV_ReadPublic(
    NV_ReadPublic_In * in,
    NV_ReadPublic_Out * out
)
```

29.8.6.116 function TPM2_NV_Write

```
WOLFTPM_API TPM_RC TPM2_NV_Write(
    NV_Write_In * in
)
```

29.8.6.117 function TPM2_NV_Increment

```
WOLFTPM_API TPM_RC TPM2_NV_Increment(
    NV_Increment_In * in
)
```

29.8.6.118 function TPM2_NV_Extend

```
WOLFTPM_API TPM_RC TPM2_NV_Extend(
    NV_Extend_In * in
)
```

29.8.6.119 function TPM2_NV_SetBits

```
WOLFTPM_API TPM_RC TPM2_NV_SetBits(
    NV_SetBits_In * in
)
```

29.8.6.120 function TPM2_NV_WriteLock

```
WOLFTPM_API TPM_RC TPM2_NV_WriteLock(
    NV_WriteLock_In * in
)
```

29.8.6.121 function TPM2_NV_GlobalWriteLock

```
WOLFTPM_API TPM_RC TPM2_NV_GlobalWriteLock(
    NV_GlobalWriteLock_In * in
)
```

29.8.6.122 function TPM2_NV_Read

```
WOLFTPM_API TPM_RC TPM2_NV_Read(
    NV_Read_In * in,
    NV_Read_Out * out
)
```

29.8.6.123 function TPM2_NV_ReadLock

```
WOLFTPM_API TPM_RC TPM2_NV_ReadLock(
    NV_ReadLock_In * in
)
```

29.8.6.124 function TPM2_NV_ChangeAuth

```
WOLFTPM_API TPM_RC TPM2_NV_ChangeAuth(
    NV_ChangeAuth_In * in
)
```

29.8.6.125 function TPM2_NV_Certify

```
WOLFTPM_API TPM_RC TPM2_NV_Certify(
    NV_Certify_In * in,
    NV_Certify_Out * out
)
```

29.8.6.126 function TPM2_SetCommandSet

```
WOLFTPM_API int TPM2_SetCommandSet(
    SetCommandSet_In * in
)
```

29.8.6.127 function TPM2_SetMode

```
WOLFTPM_API int TPM2_SetMode(
    SetMode_In * in
)
```

29.8.6.128 function TPM2_GetRandom2

```
WOLFTPM_API TPM_RC TPM2_GetRandom2(
    GetRandom2_In * in,
    GetRandom2_Out * out
)
```

29.8.6.129 function TPM2_GetProductInfo

```
WOLFTPM_API TPM_RC TPM2_GetProductInfo(
    uint8_t * info,
    uint16_t size
)
```

29.8.6.130 function TPM2_IFX_FieldUpgradeStart

```
WOLFTPM_API int TPM2_IFX_FieldUpgradeStart(
    TPM_HANDLE sessionHandle,
    uint8_t * data,
    uint32_t size
)
```

29.8.6.131 function TPM2_IFX_FieldUpgradeCommand

```
WOLFTPM_API int TPM2_IFX_FieldUpgradeCommand(
    TPM_CC cc,
    uint8_t * data,
    uint32_t size
)
```

29.8.6.132 function TPM2_ST33_FieldUpgradeStart

```
WOLFTPM_API int TPM2_ST33_FieldUpgradeStart(
    TPM_HANDLE sessionHandle,
    uint8_t * data,
    uint32_t size
)
```

29.8.6.133 function TPM2_ST33_FieldUpgradeStart_ex

```
WOLFTPM_API int TPM2_ST33_FieldUpgradeStart_ex(
    TPM_HANDLE sessionHandle,
    TPM_CC cc,
    uint8_t * data,
    uint32_t size
)
```

29.8.6.134 function TPM2_ST33_FieldUpgradeCommand

```
WOLFTPM_API int TPM2_ST33_FieldUpgradeCommand(
    TPM_CC cc,
    uint8_t * data,
    uint32_t size
)
```

29.8.6.135 function TPM2_GPIO_Config

```
WOLFTPM_API int TPM2_GPIO_Config(
    GpioConfig_In * in
)
```

29.8.6.136 function TPM2_NTC2_PreConfig

```
WOLFTPM_API int TPM2_NTC2_PreConfig(
    NTC2_PreConfig_In * in
)
```

29.8.6.137 function TPM2_NTC2_GetConfig

```
WOLFTPM_API int TPM2_NTC2_GetConfig(
    NTC2_GetConfig_Out * out
)
```

29.8.6.138 function TPM2_Nations_IdentityKeySet

```
WOLFTPM_API int TPM2_Nations_IdentityKeySet(
    Nations_IdentityKeySet_In * in
)
```

29.8.6.139 function TPM2_Init

```
WOLFTPM_API TPM_RC TPM2_Init(
    TPM2_CTX * ctx,
    TPM2HalIoCb ioCb,
```

```
    void * userCtx
)
```

Initializes a TPM with HAL IO callback and user supplied context. When using wolfTPM with –enable-devtpm or –enable-swtpm configuration, the ioCb and userCtx are not used.

Parameters:

- **ctx** pointer to a TPM2_CTX struct
- **ioCb** pointer to TPM2HalIoCb (HAL IO) callback function
- **userCtx** pointer to the user's context that will be stored as a member of the ctx struct

See:

- TPM2_Startup
- TPM2_GetRCString
- TPM2_Init_minimal
- TPM2_Init_ex
- wolfTPM2_Init

Return:

- TPM_RC_SUCCESS: successful
- TPM_RC_FAILURE: general error (possibly IO)
- BAD_FUNC_ARG check arguments provided

Note: TPM2_Init_minimal() with both ioCb and userCtx set to NULL. In other modes, the ioCb shall be set in order to use TIS. Example ioCB for baremetal and RTOS applications are provided in hal/tpm_io.c

Example

```
int rc;
TPM2_CTX tpm2Ctx;

rc = TPM2_Init(&tpm2Ctx, TPM2_IoCb, userCtx);
if (rc != TPM_RC_SUCCESS) {
    // TPM2_Init failed
}
```

29.8.6.140 function TPM2_Init_ex

```
WOLFTPM_API TPM_RC TPM2_Init_ex(
    TPM2_CTX * ctx,
    TPM2HalIoCb ioCb,
    void * userCtx,
    int timeoutTries
)
```

Initializes a TPM with timeoutTries, HAL IO callback and user supplied context.

Parameters:

- **ctx** pointer to a TPM2_CTX struct
- **ioCb** pointer to TPM2HalIoCb (HAL IO) callback function
- **userCtx** pointer to the user's context that will be stored as a member of the ctx struct
- **timeoutTries** specifies the number of attempts to confirm that TPM2 startup has completed

See:

- TPM2_GetRCString
- TPM2_Init_minimal
- TPM2_Init

- wolfTPM2_Init_ex

Return:

- TPM_RC_SUCCESS: successful
- TPM_RC_FAILURE: general error (possibly IO)
- BAD_FUNC_ARG check arguments provided

Note: It is recommended to use TPM2_Init instead of using TPM2_Init_ex directly.

29.8.6.141 function TPM2_Init_minimal

```
WOLFTPM_API TPM_RC TPM2_Init_minimal(
    TPM2_CTX * ctx
)
```

Initializes a TPM and sets the wolfTPM2 context that will be used. This function is typically used for rich operating systems, like Windows.

Parameters:

- **ctx** pointer to a TPM2_CTX struct

See:

- TPM2_GetRCString
- TPM2_Init

Return:

- TPM_RC_SUCCESS: successful
- TPM_RC_FAILURE: general error (possibly IO)
- BAD_FUNC_ARG check arguments provided

Note: It is recommended to use TPM2_Init instead of using TPM2_Init_minimal directly.

29.8.6.142 function TPM2_Cleanup

```
WOLFTPM_API TPM_RC TPM2_Cleanup(
    TPM2_CTX * ctx
)
```

Deinitializes a TPM and wolfcrypt (if it was initialized)

Parameters:

- **ctx** pointer to a TPM2_CTX struct

See:

- TPM2_GetRCString
- TPM2_Init
- wolfTPM2_Cleanup

Return:

- TPM_RC_SUCCESS: successful
- TPM_RC_FAILURE: could not acquire the lock on the wolfTPM2 context
- BAD_FUNC_ARG: the TPM2 device structure is a NULL pointer

Example

```
int rc;
TPM2_CTX tpm2Ctx;

rc = TPM2_Cleanup(&tpm2Ctx->dev);
if (rc != TPM_RC_SUCCESS) {
    // TPM2_Cleanup failed
}
```

29.8.6.143 function TPM2_ChipStartup

```
WOLFTPM_API TPM_RC TPM2_ChipStartup(
    TPM2_CTX * ctx,
    int timeoutTries
)
```

Makes sure the TPM2 startup has completed and extracts the TPM device information.

Parameters:

- **ctx** pointer to a TPM2_CTX struct
- **timeoutTries** specifies the number of attempts to check if TPM2 startup has completed

See:

- TPM2_GetRCString
- TPM2_TIS_StartupWait
- TPM2_TIS_RequestLocality
- TPM2_TIS_GetInfo
- TPM2_Init_ex

Return:

- TPM_RC_SUCCESS: successful
- TPM_RC_FAILURE: general error (possibly IO)
- BAD_FUNC_ARG: check the provided arguments
- TPM_RC_TIMEOUT: timeout occurred

Note: This function is used in TPM2_Init_ex

29.8.6.144 function TPM2_SetHalIoCb

```
WOLFTPM_API TPM_RC TPM2_SetHalIoCb(
    TPM2_CTX * ctx,
    TPM2HalIoCb ioCb,
    void * userCtx
)
```

Sets the user's context and IO callbacks needed for TPM communication.

Parameters:

- **ctx** pointer to a TPM2_CTX struct
- **ioCb** pointer to TPM2HalIoCb (HAL IO) callback function
- **userCtx** pointer to the user's context that will be stored as a member of the ctx struct

See:

- TPM2_GetRCString
- TPM2_Init
- wolfTPM2_Init

Return:

- TPM_RC_SUCCESS: successful
- TPM_RC_FAILURE: could not acquire the lock on the wolfTPM2 context
- BAD_FUNC_ARG: the TPM2 device structure is a NULL pointer

Note: SetHalIoCb will fail if built with devtpm or swtpm as the callback is not used for TPM. For other configuration builds, ioCb must be set to a non-NULL function pointer and userCtx is optional.

Typically, TPM2_Init or wolfTPM2_Init are used to set the HAL IO.

29.8.6.145 function TPM2_SetCommandRetries

```
WOLFTPM_API TPM_RC TPM2_SetCommandRetries(
    TPM2_CTX * ctx,
    int retries
)
```

Sets the number of times a command is transparently resubmitted on TPM_RC_RETRY.

Parameters:

- **ctx** pointer to a TPM2_CTX struct
- **retries** number of additional resubmit attempts on TPM_RC_RETRY (0 disables)

See: TPM2_GetCommandRetries

Return:

- TPM_RC_SUCCESS: successful
- BAD_FUNC_ARG: the TPM2 context is NULL or retries is negative

The TPM returns TPM_RC_RETRY when momentarily busy (for example persisting the daUsed flag on first auth use of a non-noDA key). Disabled by default (WOLFTPM_MAX_RETRIES is 0); pass a count > 0 to enable transparent resubmit. A count of 0 returns TPM_RC_RETRY to the caller.

29.8.6.146 function TPM2_GetCommandRetries

```
WOLFTPM_API int TPM2_GetCommandRetries(
    TPM2_CTX * ctx
)
```

Returns the number of times a command is transparently resubmitted on TPM_RC_RETRY.

Parameters:

- **ctx** pointer to a TPM2_CTX struct

See: TPM2_SetCommandRetries

Return:

- the configured retry count on success
- BAD_FUNC_ARG: the TPM2 context is a NULL pointer

Note: Callers must synchronize concurrent calls to this function and TPM2_SetCommandRetries.

29.8.6.147 function TPM2_SetSessionAuth

```
WOLFTPM_API TPM_RC TPM2_SetSessionAuth(
    TPM2_AUTH_SESSION * session
)
```

Sets the structure holding the TPM Authorizations.

Parameters:

- **session** pointer to an array of type TPM2_AUTH_SESSION

See:

- TPM2_GetRCString
- TPM2_Init
- wolfTPM2_Init

Return:

- TPM_RC_SUCCESS: successful
- TPM_RC_FAILURE: could not acquire the lock on the wolfTPM2 context
- BAD_FUNC_ARG: the TPM2 context structure is a NULL pointer

Rarely used, because TPM2_Init functions and wolfTPM2_Init perform this initialization as well TPM 2.0 Commands can have up to three authorization slots, therefore it is recommended to supply an array of size MAX_SESSION_NUM to TPM2_SetSessionAuth(see example below).

Example

```
int rc;
TPM2_AUTH_SESSION session[MAX_SESSION_NUM];

XMEMSET(session, 0, sizeof(session));
session[0].sessionHandle = TPM_RS_PW;

rc = TPM2_SetSessionAuth(session);
if (rc != TPM_RC_SUCCESS) {
    // TPM2_SetSessionAuth failed
}
```

29.8.6.148 function TPM2_GetSessionAuthCount

```
WOLFTPM_API int TPM2_GetSessionAuthCount(
    TPM2_CTX * ctx
)
```

Determine the number of currently set TPM Authorizations.

Parameters:

- **ctx** pointer to a TPM2_CTX struct

See:

- TPM2_CTX
- TPM2_AUTH_SESSION

Return:

- the number of active TPM Authorizations (between one and three)
- BAD_FUNC_ARG: check the arguments provided for a NULL pointer

Example

```
int authCount;
TPM2_CTX tpm2Ctx;

authCount = TPM2_GetSessionAuthCount(&tpm2Ctx);
```

```
if (authCount == BAD_FUNC_ARG) {
    // TPM2_GetSessionAuthCount failed
}
```

29.8.6.149 function TPM2_SetActiveCtx

```
WOLFTPM_API void TPM2_SetActiveCtx(
    TPM2_CTX * ctx
)
```

Sets a new TPM2 context for use.

Parameters:

- **ctx** pointer to a TPM2_CTX struct

See:

- TPM2_CTX
- TPM2_AUTH_SESSION

Example

```
TPM2_CTX tpm2Ctx;

TPM2_SetActiveCtx(&tpm2Ctx);
```

29.8.6.150 function TPM2_GetActiveCtx

```
WOLFTPM_API TPM2_CTX * TPM2_GetActiveCtx(
    void
)
```

Provides a pointer to the TPM2 context in use.

See:

- TPM2_CTX
- TPM2_AUTH_SESSION

Return: ctx pointer to a TPM2_CTX struct

Example

```
TPM2_CTX *tpm2Ctx;

tpm2Ctx = TPM2_GetActiveCtx();
```

29.8.6.151 function TPM2_GetHashDigestSize

```
WOLFTPM_API int TPM2_GetHashDigestSize(
    TPMI_ALG_HASH hashAlg
)
```

Determine the size in bytes of a TPM 2.0 hash digest.

Parameters:

- **hashAlg** a valid TPM 2.0 hash type

Return:

- the size of a TPM 2.0 hash digest as number of bytes

- 0 if hash type is invalid

Example

```
int digestSize = 0;
TPMI_ALG_HASH hashAlg = TPM_ALG_SHA256;

digestSize = TPM2_GetHashDigestSize(hashAlg);
if (digestSize > 0) {
    //digestSize contains a valid value
}
```

29.8.6.152 function TPM2_GetHashType

```
WOLFTPM_API int TPM2_GetHashType(
    TPMI_ALG_HASH hashAlg
)
```

Translate a TPM2 hash type to its corresponding wolfcrypt hash type.

Parameters:

- **hashAlg** a valid TPM 2.0 hash type

Return:

- a value specifying a hash type to use with wolfcrypt
- 0 if hash type is invalid

Example

```
int wc_hashType;
TPMI_ALG_HASH hashAlg = TPM_ALG_SHA256;

wc_hashType = TPM2_GetHashDigestSize(hashAlg);
if (wc_hashType > 0) {
    //wc_hashType contains a valid wolfcrypt hash type
}
```

29.8.6.153 function TPM2_GetTpmHashType

```
WOLFTPM_API TPMI_ALG_HASH TPM2_GetTpmHashType(
    int hashType
)
```

Translate a wolfCrypt hash type to TPM2 hash type.

Parameters:

- **hashType** a wolfCrypt hash type

Return:

- a TPM2 hash type (TPM_ALG_*)
- TPM_ALG_ERROR when wolfCrypt hash type is invalid or not found

Example

```
int wc_hashType = WC_HASH_TYPE_SHA256;
TPMI_ALG_HASH hashAlg;

hashAlg = TPM2_GetHashDigestSize(wc_hashType);
```

```
if (hashAlg != TPM_ALG_ERROR) {
    //hashAlg contains a valid TPM2 hash type
}
```

29.8.6.154 function TPM2_GetNonce

```
WOLFTPM_API int TPM2_GetNonce(
    byte * nonceBuf,
    int nonceSz
)
```

Generate a fresh nonce of random numbers.

Parameters:

- **nonceBuf** pointer to a BYTE buffer
- **nonceSz** size of the nonce in bytes

Return:

- TPM_RC_SUCCESS: successful
- TPM_RC_FAILURE: generic failure (TPM IO issue or wolfcrypt configuration)
- BAD_FUNC_ARG: check the provided arguments

Note: Can use the TPM random number generator if WOLFTPM2_USE_HW_RNG is defined. WARNING: Using WOLFTPM2_USE_HW_RNG exposes nonces to physical bus tampering (SPI/I2C). An attacker with bus access can force a known nonce, which can significantly weaken session security, especially for unsalted sessions. Use the default local wolfCrypt RNG unless the bus is physically secure.

Example

```
int rc, nonceSize = 32;
BYTE freshNonce[32];

rc = TPM2_GetNonce(freshNonce, nonceSize);
if (rc != TPM_RC_SUCCESS) {
    //TPM2_GetNonce failed
}
```

29.8.6.155 function TPM2_GetNonceNoLock

```
WOLFTPM_LOCAL int TPM2_GetNonceNoLock(
    byte * nonceBuf,
    int nonceSz
)
```

29.8.6.156 function TPM2_SetupPCRSel

```
WOLFTPM_API void TPM2_SetupPCRSel(
    TPML_PCR_SELECTION * pcr,
    TPM_ALG_ID alg,
    int pcrIndex
)
```

Helper function to prepare a correct PCR selection For example, when preparing to create a TPM2_Quote.

Parameters:

- **pcr** pointer to a structure of type TPML_PCR_SELECTION. Note: Caller must zeroize/memset(0)

- **alg** value of type TPM_ALG_ID specifying the type of hash algorithm used
- **pcrIndex** value between 0 and 23 specifying the PCR register for use

See:

- TPM2_PCR_Read
- TPM2_PCR_Extend
- TPM2_PCR_Reset
- TPM2_Quote

Example

```
int pcrIndex = 16; // This is a PCR register for DEBUG & testing purposes
PCR_Read_In pcrRead;
XMEMSET(&pcrRead, 0, sizeof(pcrRead));
TPM2_SetupPCRSel(&pcrRead.pcrSelectionIn, TPM_ALG_SHA256, pcrIndex);
```

29.8.6.157 function TPM2_SetupPCRSelArray

```
WOLFTPM_API void TPM2_SetupPCRSelArray(
    TPML_PCR_SELECTION * pcr,
    TPM_ALG_ID alg,
    byte * pcrArray,
    word32 pcrArraySz
)
```

Helper function to prepare a correct PCR selection with multiple indices For example, when preparing to create a TPM2_Quote.

Parameters:

- **pcr** pointer to a structure of type TPML_PCR_SELECTION. Note: Caller must zeroize/memset(0)
- **alg** value of type TPM_ALG_ID specifying the type of hash algorithm used
- **pcrArray** array of values between 0 and 23 specifying the PCR register for use
- **pcrArraySz** length of the pcrArray

See:

- TPM2_PCR_Read
- TPM2_PCR_Extend
- TPM2_PCR_Reset
- TPM2_Quote

Example

```
PCR_Read_In pcrRead;
byte   pcrArray[PCR_SELECT_MAX];
word32 pcrArraySz = 0;

XMEMSET(&pcrRead, 0, sizeof(pcrRead));
XMEMSET(pcrArray, 0, sizeof(pcrArray));
pcrArray[pcrArraySz++] = 16; // This is a PCR register for DEBUG & testing
 ↪  purposes

TPM2_SetupPCRSelArray(&pcrRead.pcrSelectionIn, TPM_ALG_SHA256, pcrArray,
 ↪  pcrArraySz);
```

29.8.6.158 function TPM2_IsAlgSupported

```
WOLFTPM_API int TPM2_IsAlgSupported(
    TPM_ALG_ID alg,
    int * isSupported
)
```

Report whether the TPM implements a given algorithm.

Parameters:

- **alg** the algorithm identifier to test (for example TPM_ALG_SHA512)
- **isSupported** output, set to 1 if implemented by the TPM, else 0

See: TPM2_IsPcrBankAllocated

Return:

- TPM_RC_SUCCESS: query completed; *isSupported is 1 or 0
- BAD_FUNC_ARG: isSupported is NULL

Note: Queries TPM_CAP_ALGS. Fails closed: *isSupported is 0 on any error, so a query failure cannot be mistaken for "supported".

29.8.6.159 function TPM2_IsPcrBankAllocated

```
WOLFTPM_API int TPM2_IsPcrBankAllocated(
    TPM_ALG_ID hashAlg,
    int pcrIndex,
    int * isAllocated
)
```

Report whether a PCR index is allocated in a bank of a given hash.

Parameters:

- **hashAlg** the PCR bank hash algorithm (for example TPM_ALG_SHA256)
- **pcrIndex** the PCR index to test
- **isAllocated** output, set to 1 if allocated in that bank, else 0

See:

- TPM2_SetupPCRSel
- TPM2_IsAlgSupported

Return:

- TPM_RC_SUCCESS: query completed; *isAllocated is 1 or 0
- BAD_FUNC_ARG: isAllocated is NULL or pcrIndex is negative

Note: Queries TPM_CAP_PCRS. Implementing a hash and allocating a bank for it are separate: a TPM may offer SHA-1 while allocating no SHA-1 bank, and a selection naming an unallocated bank is rejected. Ask before building a selection with TPM2_SetupPCRSel(). Fails closed: isAllocated is 0 on any error.

29.8.6.160 function TPM2_GetRCString

```
WOLFTPM_API const char * TPM2_GetRCString(
    int rc
)
```

Get a human readable string for any TPM 2.0 return code.

Parameters:

- **rc** integer value representing a TPM return code

Return: pointer to a string constant

Example

```
int rc;

rc = wolfTPM2_Init(&dev, TPM2_IoCb, userCtx);
if (rc != TPM_RC_SUCCESS) {
    printf("wolfTPM2_Init failed 0x%x: %s\n", rc, TPM2_GetRCString(rc));
    return rc;
}
```

29.8.6.161 function TPM2_GetAlgName

```
WOLFTPM_API const char * TPM2_GetAlgName(
    TPM_ALG_ID alg
)
```

Get a human readable string for any TPM 2.0 algorithm.

Parameters:

- **alg** value of type TPM_ALG_ID specifying a valid TPM 2.0 algorithm

Return: pointer to a string constant

Example

```
int paramEncAlg = TPM_ALG_CFB;

printf("\tUse Parameter Encryption: %s\n", TPM2_GetAlgName(paramEncAlg));
```

29.8.6.162 function TPM2_GetAlgId

```
WOLFTPM_API TPM_ALG_ID TPM2_GetAlgId(
    const char * name
)
```

Translates a TPM algorithm name to its TPM algorithm ID.

Parameters:

- **name** a pointer to a string constant specifying a valid TPM algorithm name

Return:

- a TPM algorithm ID
- TPM_ALG_ERROR if invalid algorithm name

Example

```
TPM_ALG_ID alg = TPM2_GetAlgId("SHA256");
if (alg == TPM_ALG_ERROR) {
    //TPM2_GetAlgId failed
}
```

29.8.6.163 function TPM2_GetHierarchyDesc

```
WOLFTPM_API const char * TPM2_GetHierarchyDesc(
    TPMI_RH_HIERARCHY_AUTH authHandle
)
```

Get readable string for TPM 2.0 hierarchy.

Parameters:

- **authHandle** value of type TPMI_RH_HIERARCHY_AUTH specifying a valid TPM 2.0 hierarchy

Return: pointer to a string constant

Example

```
TPMI_RH_HIERARCHY_AUTH authHandle = TPM_RH_OWNER;

printf("\tHierarchy: %s\n", TPM2_GetHierarchyDesc(authHandle));
```

29.8.6.164 function TPM2_GetCurveSize

```
WOLFTPM_API int TPM2_GetCurveSize(
    TPM_ECC_CURVE curveID
)
```

Determine the size in bytes of any TPM ECC Curve.

Parameters:

- **curveID** value of type TPM_ECC_CURVE

Return:

- 0 in case of invalid curve type
- integer value representing the number of bytes

Example

```
int bytes;
TPM_ECC_CURVE curve = TPM_ECC_NIST_P256;

bytes = TPM2_GetCurveSize(curve);
if (bytes == 0) {
    //TPM2_GetCurveSize failed
}
```

29.8.6.165 function TPM2_GetCurveHashAlg

```
WOLFTPM_API TPM_ALG_ID TPM2_GetCurveHashAlg(
    TPM_ECC_CURVE curveID
)
```

Determine the recommended message digest for a TPM ECC Curve.

Parameters:

- **curveID** value of type TPM_ECC_CURVE

See: TPM2_GetCurveSize

Return: TPM_ALG_ID of the hash paired to the curve size (P256->SHA256, P384->SHA384, P521->SHA512). Defaults to TPM_ALG_SHA256.

29.8.6.166 function TPM2_GetTpmCurve

```
WOLFTPM_API int TPM2_GetTpmCurve(
    int curveID
)
```

Translate a wolfcrypt curve type to its corresponding TPM curve type.

Parameters:

- **curveID** pointer to a BYTE buffer

See: TPM2_GetWolfCurve

Return:

- integer value representing a wolfcrypt curve type
- ECC_CURVE_OID_E in case of invalid curve type

Example

```
int tpmCurve;
int wc_curve = ECC_SECP256R1;

tpmCurve = TPM2_GetTpmCurve(curve);
\in this case tpmCurve will be TPM_ECC_NIST_P256
if (tpmCurve = ECC_CURVE_OID_E) {
    //TPM2_GetTpmCurve failed
}
```

29.8.6.167 function TPM2_GetWolfCurve

```
WOLFTPM_API int TPM2_GetWolfCurve(
    int curve_id
)
```

Translate a TPM curve type to its corresponding wolfcrypt curve type.

Parameters:

- **curve_id** pointer to a BYTE buffer

See: TPM2_GetTpmCurve

Return:

- integer value representing a TPM curve type
- -1 or ECC_CURVE_OID_E in case of invalid curve type

Example

```
int tpmCurve = TPM_ECC_NIST_P256;
int wc_curve;

wc_curve = TPM2_GetWolfCurve(tpmCurve);
\in this case tpmCurve will be ECC_SECP256R1
if (wc_curve = ECC_CURVE_OID_E || wc_curve == -1) {
    //TPM2_GetWolfCurve failed
}
```

29.8.6.168 function TPM2_ParseAttest

```
WOLFTPM_API int TPM2_ParseAttest(
    const TPM2B_ATTEST * in,
    TPMS_ATTEST * out
)
```

Parses TPM2B_ATTEST structure.

Parameters:

- **in** pointer to a structure of a TPM2B_ATTEST type
- **out** pointer to a structure of a TPMS_ATTEST type

Return:

- TPM_RC_SUCCESS: successful
- BAD_FUNC_ARG: check the provided arguments
- TPM_RC_VALUE: invalid attestation magic or type
- TPM_RC_SIZE: malformed or truncated attestation data

Note: This is public API of the helper function TPM2_Packet_ParseAttest

Example

```
TPM2B_ATTEST in; //for example, as part of a TPM2_Quote
TPMS_ATTEST out

rc = TPM2_GetNonce(&in, &out);
if (rc != TPM_RC_SUCCESS) {
    //TPM2_ParseAttest failed
}
```

29.8.6.169 function TPM2_HashNvPublic

```
WOLFTPM_API int TPM2_HashNvPublic(
    TPMS_NV_PUBLIC * nvPublic,
    byte * buffer,
    UINT16 * size
)
```

Computes fresh NV Index name based on a nvPublic structure.

Parameters:

- **nvPublic**
- **buffer** pointer to a structure of a TPMS_ATTEST type
- **size** pointer to a variable of UINT16 type to store the size of the nvIndex

Return:

- TPM_RC_SUCCESS: successful
- negative integer value in case of an error
- BAD_FUNC_ARG: check the provided arguments
- NOT_COMPILED_IN: check if wolfcrypt is enabled

Example

```
TPMS_NV_PUBLIC nvPublic;
BYTE buffer[TPM_MAX_DIGEST_SIZE];
UINT16 size;
```

```
rc = TPM2_HashNvPublic(&nvPublic, &buffer, &size);
if (rc != TPM_RC_SUCCESS) {
    //TPM2_HashNvPublic failed
}
```

29.8.6.170 function TPM2_AppendPublic

```
WOLFTPM_API int TPM2_AppendPublic(
    byte * buf,
    word32 size,
    int * sizeUsed,
    TPM2B_PUBLIC * pub
)
```

Populates TPM2B_PUBLIC structure based on a user provided buffer.

Parameters:

- **buf** pointer to a user buffer
- **size** integer value of word32 type, specifying the size of the user buffer
- **sizeUsed** pointer to an integer variable, stores the used size of pub->buffer
- **pub** pointer to an empty structure of TPM2B_PUBLIC type

See: TPM2_ParsePublic

Return:

- TPM_RC_SUCCESS: successful
- TPM_RC_FAILURE: insufficient buffer size
- BAD_FUNC_ARG: check the provided arguments

Note: Public API of the helper function TPM2_Packet_AppendPublic

Example

```
TPM2B_PUBLIC pub; //empty
int sizeUsed, rc;
BYTE buffer[sizeof(TPM2B_PUBLIC)];
word32 size = sizeof(buffer);

rc = TPM2_AppendPublic(&buffer, size, &sizeUsed, &pub);
if (rc != TPM_RC_SUCCESS) {
    //TPM2_AppendPublic failed
}
```

29.8.6.171 function TPM2_ParsePublic

```
WOLFTPM_API int TPM2_ParsePublic(
    TPM2B_PUBLIC * pub,
    byte * buf,
    word32 size,
    int * sizeUsed
)
```

Parses TPM2B_PUBLIC structure and stores in a user provided buffer.

Parameters:

- **pub** pointer to a populated structure of TPM2B_PUBLIC type
- **buf** pointer to an empty user buffer

- **size** integer value of word32 type, specifying the available size of the user buffer
- **sizeUsed** pointer to an integer variable, stores the used size of the user buffer

See: TPM2_AppendPublic

Return:

- TPM_RC_SUCCESS: successful
- TPM_RC_FAILURE: insufficient buffer size
- BAD_FUNC_ARG: check the provided arguments

Note: Public API of the helper function TPM2_Packet_ParsePublic

Example

```
TPM2B_PUBLIC pub; //populated
int sizeUsed, rc;
BYTE buffer[sizeof(TPM2B_PUBLIC)];
word32 size = sizeof(buffer);

rc = TPM2_ParsePublic(&pub, buffer, size, &sizeUsed);
if (rc != TPM_RC_SUCCESS) {
    //TPM2_ParsePublic failed
}
```

29.8.6.172 function TPM2_GetName

```
WOLFTPM_LOCAL int TPM2_GetName(
    TPM2_CTX * ctx,
    UINT32 handleValue,
    int handleCnt,
    int idx,
    TPM2B_NAME * name
)
```

Provides the Name of a TPM object.

Parameters:

- **ctx** pointer to a TPM2 context
- **handleValue** value of UINT32 type, specifying a valid TPM handle
- **handleCnt** total number of handles used in the current TPM command/session
- **idx** index value, between one and three, specifying a valid TPM Authorization session
- **name** pointer to an empty structure of TPM2B_NAME type

Return:

- TPM_RC_SUCCESS: successful
- BAD_FUNC_ARG: check the provided arguments

Note: The object is reference by its TPM handle and session index

Example

```
int rc;
UINT32 handleValue = TRANSIENT_FIRST;
handleCount = 1;
sessionIdx = 0;
TPM2B_NAME name;

rc = TPM2_GetName(ctx, handleValue, handleCount, sessionIdx, &name);
```

```
if (rc != TPM_RC_SUCCESS) {
    //TPM2_GetName failed
}
```

29.8.6.173 function TPM2_GetWolfRng

```
WOLFTPM_API int TPM2_GetWolfRng(
    WC_RNG ** rng
)
```

29.8.6.174 function TPM2_GetVendorID

```
WOLFTPM_API UINT16 TPM2_GetVendorID(
    void
)
```

Provides the vendorID of the active TPM2 context.

See:

- TPM2_GetCapabilities
- TPM2_Init

Return:

- integer value of UINT16 type, specifying the vendor ID
- 0 if TPM2 context is invalid or NULL

Note: Depends on correctly read TPM device info during TPM Init

Example

```
TPM2_CTX *tpm2Ctx;

tpm2Ctx = TPM2_GetActiveCtx();
```

29.8.6.175 function TPM2_GetCapsFipsStr

```
WOLFTPM_API const char * TPM2_GetCapsFipsStr(
    int fips140_3,
    int fips140_2
)
```

Returns a human-readable FIPS mode string from FIPS capability bits: “140-3” when fips140_3 is set, “140-2” when fips140_2 is set, otherwise “none”. Used for capability reporting in examples and tools.

Parameters:

- **fips140_3** non-zero if the TPM reports FIPS 140-3 mode
- **fips140_2** non-zero if the TPM reports FIPS 140-2 mode

Return: Non-NULL pointer to a static string. Never NULL.

29.8.6.176 function TPM2_ForceZero

```
WOLFTPM_LOCAL void TPM2_ForceZero(
    void * mem,
    word32 len
)
```

Forcefully zero a memory region. Unlike memset, this call is not subject to dead-store elimination, so it is safe to use for erasing sensitive material (keys, intermediate HMAC state, etc.) just before the memory goes out of scope.

Parameters:

- **mem** pointer to the buffer to zero (may be NULL; call is a no-op)
- **len** number of bytes to zero

See: TPM2_ConstantCompare

29.8.6.177 function TPM2_ConstantCompare

```
WOLFTPM_TEST_API int TPM2_ConstantCompare(
    const byte * a,
    const byte * b,
    word32 len
)
```

Constant-time memory comparison. Returns 0 on byte-for-byte match and non-zero on mismatch. Timing is independent of where the first differing byte is located, which is required when comparing HMAC digests or authentication values.

Parameters:

- **a** first buffer
- **b** second buffer
- **len** number of bytes to compare

Return:

- 0 if len bytes of a and b are identical
- non-zero on any mismatch

29.8.6.178 function TPM2_PrintBin

```
WOLFTPM_API void TPM2_PrintBin(
    const byte * buffer,
    word32 length
)
```

Helper function to print a binary buffer in a formatted way.

Parameters:

- **buffer** pointer to a buffer of BYTE type
- **length** integer value of word32 type, containing the size of the buffer

See:

- TPM2_PrintAuth
- TPM2_PrintPublicArea

Note: Requires DEBUG_WOLFTPM to be defined

Example

```
BYTE buffer[] = {0x01,0x02,0x03,0x04};
word32 length = sizeof(buffer);

TPM2_PrintBin(buffer, length);
```

29.8.6.179 function TPM2_PrintAuth

```
WOLFTPM_API void TPM2_PrintAuth(
    const TPMS_AUTH_COMMAND * authCmd
)
```

Helper function to print a structure of TPMS_AUTH_COMMAND type in a human readable way.

Parameters:

- **authCmd** pointer to a populated structure of TPMS_AUTH_COMMAND type

See:

- TPM2_PrintBin
- TPM2_PrintPublicArea

Note: Requires DEBUG_WOLFTPM to be defined

Example

```
TPMS_AUTH_COMMAND authCmd; //for example, part of a TPM Authorization session

TPM2_PrintAuth(&authCmd);
```

29.8.6.180 function TPM2_PrintPublicArea

```
WOLFTPM_API void TPM2_PrintPublicArea(
    const TPM2B_PUBLIC * pub
)
```

Helper function to print a structure of TPM2B_PUBLIC type in a human readable way.

Parameters:

- **pub** pointer to a populated structure of TPM2B_PUBLIC type

See:

- TPM2_PrintBin
- TPM2_PrintAuth
- TPM2_Create
- TPM2_ReadPublic

Note: Requires DEBUG_WOLFTPM to be defined

Example

```
TPM2B_PUBLIC pub; //for example, part of the output of a successful TPM2_Create

TPM2_PrintPublicArea(&pub);
```

29.8.7 Attributes Documentation

29.8.7.1 variable C

```
C {
#endif
```

```
typedef UINT32 TPM_ALGORITHM_ID;
```

29.8.7.2 variable TPM_20_EK_AUTH_POLICY

```
static const BYTE[] TPM_20_EK_AUTH_POLICY = {
    0x83, 0x71, 0x97, 0x67, 0x44, 0x84, 0xB3, 0xF8,
    0x1A, 0x90, 0xCC, 0x8D, 0x46, 0xA5, 0xD7, 0x24,
    0xFD, 0x52, 0xD7, 0x6E, 0x06, 0x52, 0x0B, 0x64,
    0xF2, 0xA1, 0xDA, 0x1B, 0x33, 0x14, 0x69, 0xAA
};
```

29.8.7.3 variable TPM_20_EK_AUTH_POLICY_SHA256

```
static const BYTE[] TPM_20_EK_AUTH_POLICY_SHA256 = {
    0xCA, 0x3D, 0x0A, 0x99, 0xA2, 0xB9, 0x39, 0x06,
    0xF7, 0xA3, 0x34, 0x24, 0x14, 0xEF, 0xCF, 0xB3,
    0xA3, 0x85, 0xD4, 0x4C, 0xD1, 0xFD, 0x45, 0x90,
    0x89, 0xD1, 0x9B, 0x50, 0x71, 0xC0, 0xB7, 0xA0
};
```

29.8.7.4 variable TPM_20_EK_AUTH_POLICY_SHA384

```
static const BYTE[] TPM_20_EK_AUTH_POLICY_SHA384 = {
    0xB2, 0x6E, 0x7D, 0x28, 0xD1, 0x1A, 0x50, 0xBC,
    0x53, 0xD8, 0x82, 0xBC, 0xF5, 0xFD, 0x3A, 0x1A,
    0x07, 0x41, 0x48, 0xBB, 0x35, 0xD3, 0xB4, 0xE4,
    0xCB, 0x1C, 0x0A, 0xD9, 0xBD, 0xE4, 0x19, 0xCA,
    0xCB, 0x47, 0xBA, 0x09, 0x69, 0x96, 0x46, 0x15,
    0x0F, 0x9F, 0xC0, 0x00, 0xF3, 0xF8, 0x0E, 0x12
};
```

29.8.7.5 variable TPM_20_EK_AUTH_POLICY_SHA512

```
static const BYTE[] TPM_20_EK_AUTH_POLICY_SHA512 = {
    0xB8, 0x22, 0x1C, 0xA6, 0x9E, 0x85, 0x50, 0xA4,
    0x91, 0x4D, 0xE3, 0xFA, 0xA6, 0xA1, 0x8C, 0x07,
    0x2C, 0xC0, 0x12, 0x08, 0x07, 0x3A, 0x92, 0x8D,
    0x5D, 0x66, 0xD5, 0x9E, 0xF7, 0x9E, 0x49, 0xA4,
    0x29, 0xC4, 0x1A, 0x6B, 0x26, 0x95, 0x71, 0xD5,
    0x7E, 0xDB, 0x25, 0xFB, 0xDB, 0x18, 0x38, 0x42,
    0x56, 0x08, 0xB4, 0x13, 0xCD, 0x61, 0x6A, 0x5F,
    0x6D, 0xB5, 0xB6, 0x07, 0x1A, 0xF9, 0x9B, 0xEA
};
```

29.8.7.6 variable TPM_20_IDEVID_POLICY

```
static const BYTE[] TPM_20_IDEVID_POLICY = {
    0xAD, 0x6B, 0x3A, 0x22, 0x84, 0xFD, 0x69, 0x8A,
    0x07, 0x10, 0xBF, 0x5C, 0xC1, 0xB9, 0xBD, 0xF1,
```

```
    0x5E, 0x25, 0x32, 0xE3, 0xF6, 0x01, 0xFA, 0x4B,
    0x93, 0xA6, 0xA8, 0xFA, 0x8D, 0xE5, 0x79, 0xEA
};
```

29.8.7.7 variable TPM_20_IAK_POLICY

```
static const BYTE[] TPM_20_IAK_POLICY = {
    0x54, 0x37, 0x18, 0x23, 0x26, 0xE4, 0x14, 0xFC,
    0xA7, 0x97, 0xD5, 0xF1, 0x74, 0x61, 0x5A, 0x16,
    0x41, 0xF6, 0x12, 0x55, 0x79, 0x7C, 0x3A, 0x2B,
    0x22, 0xC2, 0x1D, 0x12, 0x0B, 0x2D, 0x1E, 0x07
};
```

29.8.7.8 variable TPM_20_IDEVID_POLICY_SHA384

```
static const BYTE[] TPM_20_IDEVID_POLICY_SHA384 = {
    0x4D, 0xB1, 0xAA, 0x83, 0x6D, 0x0B, 0x56, 0x15,
    0xDF, 0x6E, 0xE5, 0x3A, 0x40, 0xEF, 0x70, 0xC6,
    0x1C, 0x21, 0x7F, 0x43, 0x03, 0xD4, 0x46, 0x95,
    0x92, 0x59, 0x72, 0xBC, 0x92, 0x70, 0x06, 0xCF,
    0xA5, 0xCB, 0xDF, 0x6D, 0xC1, 0x8C, 0x4D, 0xBE,
    0x32, 0x9B, 0x2F, 0x15, 0x42, 0xC3, 0xDD, 0x33
};
```

29.8.7.9 variable TPM_20_IAK_POLICY_SHA384

```
static const BYTE[] TPM_20_IAK_POLICY_SHA384 = {
    0x12, 0x9D, 0x94, 0xEB, 0xF8, 0x45, 0x56, 0x65,
    0x2C, 0x6E, 0xEF, 0x43, 0xBB, 0xB7, 0x57, 0x51,
    0x2A, 0xC8, 0x7E, 0x52, 0xBE, 0x7B, 0x34, 0x9C,
    0xA6, 0xCE, 0x4D, 0x82, 0x6F, 0x74, 0x9F, 0xCF,
    0x67, 0x2F, 0x51, 0x71, 0x6C, 0x5C, 0xBB, 0x60,
    0x5F, 0x31, 0x3B, 0xF3, 0x45, 0xAA, 0xB3, 0x12
};
```

29.8.7.10 variable TPM_20_IDEVID_POLICY_SHA512

```
static const BYTE[] TPM_20_IDEVID_POLICY_SHA512 = {
    0x7D, 0xD7, 0x50, 0x0F, 0xD6, 0xC1, 0xB9, 0x4F,
    0x97, 0xA6, 0xAF, 0x91, 0x0D, 0xA1, 0x47, 0x30,
    0x1E, 0xF2, 0x8F, 0x66, 0x2F, 0xEE, 0x06, 0xF2,
    0x25, 0xA4, 0xCC, 0xAD, 0xDA, 0x3B, 0x4E, 0x6B,
    0x38, 0xE6, 0x6B, 0x2F, 0x3A, 0xD5, 0xDE, 0xE1,
    0xA0, 0x50, 0x3C, 0xD2, 0xDA, 0xED, 0xB1, 0xE6,
    0x8C, 0xFE, 0x4F, 0x84, 0xB0, 0x3A, 0x8C, 0xD2,
    0x2B, 0xB6, 0xA9, 0x76, 0xF0, 0x71, 0xA7, 0x2F
};
```

29.8.7.11 variable TPM_20_IAK_POLICY_SHA512

```
static const BYTE[] TPM_20_IAK_POLICY_SHA512 = {
    0x80, 0x60, 0xD1, 0xFB, 0x31, 0x71, 0x6A, 0x29,
    0xE4, 0x8A, 0x6E, 0x5F, 0xEC, 0xE0, 0x88, 0xBC,
    0xFC, 0x1B, 0x27, 0x8F, 0xC1, 0x62, 0x25, 0x5E,
```

```
    0x81, 0xC3, 0xEC, 0xA3, 0x54, 0x4C, 0xD4, 0x4A,
    0xF9, 0x44, 0x10, 0xC3, 0x71, 0x5D, 0x56, 0x1C,
    0xCC, 0xD9, 0xE3, 0x9A, 0x6C, 0xB2, 0x64, 0x6D,
    0x43, 0x53, 0x5B, 0xB5, 0x4E, 0xA8, 0x87, 0x10,
    0xDE, 0xB5, 0xF7, 0x83, 0x6B, 0xD9, 0xB5, 0x86
};
```

29.8.8 Source code

```
/* tpm2.h
 *
 * Copyright (C) 2006-2026 wolfSSL Inc.
 *
 * This file is part of wolfTPM.
 *
 * wolfTPM is free software; you can redistribute it and/or modify
 * it under the terms of the GNU General Public License as published by
 * the Free Software Foundation; either version 3 of the License, or
 * (at your option) any later version.
 *
 * wolfTPM is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
 * GNU General Public License for more details.
 *
 * You should have received a copy of the GNU General Public License
 * along with this program; if not, write to the Free Software
 * Foundation, Inc., 51 Franklin Street, Fifth Floor, Boston, MA 02110-1335, USA
 */

#ifndef __TPM2_H__
#define __TPM2_H__

#include <wolftpm/tpm2_types.h>

#ifdef __cplusplus
    extern "C" {
#endif


/* ------------------------------------------------------------------------------
 ↪  */
/* TYPES */
/* ------------------------------------------------------------------------------
 ↪  */

typedef UINT32 TPM_ALGORITHM_ID;
typedef UINT32 TPM_MODIFIER_INDICATOR;
typedef UINT32 TPM_AUTHORIZATION_SIZE;
typedef UINT32 TPM_PARAMETER_SIZE;
typedef UINT16 TPM_KEY_SIZE;
typedef UINT16 TPM_KEY_BITS;
typedef UINT32 TPM_GENERATED;
```

```
/* ---------------------------------------------------------------------------
 ↪  */
/* ENUMERATIONS */
/* ---------------------------------------------------------------------------
 ↪  */

#define TPM_SPEC_FAMILY       0x322E3000
#define TPM_SPEC_LEVEL        0
#ifdef WOLFTPM_V185
    #define TPM_SPEC_VERSION  185
    #define TPM_SPEC_YEAR     0
    #define TPM_SPEC_ERRATA   0
#else
    #define TPM_SPEC_VERSION      138
    #define TPM_SPEC_YEAR         2016
    #define TPM_SPEC_DAY_OF_YEAR  273
#endif

#define TPM_GENERATED_VALUE   0xff544347U
#define TPM_MAX_DERIVATION_BITS 8192U

typedef enum {
    TPM_ALG_ERROR           = 0x0000,
    TPM_ALG_FIRST           = 0x0001,
    TPM_ALG_RSA             = TPM_ALG_FIRST,
    TPM_ALG_SHA             = 0x0004,
    TPM_ALG_SHA1            = TPM_ALG_SHA,
    TPM_ALG_HMAC            = 0x0005,
    TPM_ALG_AES             = 0x0006,
    TPM_ALG_MGF1            = 0x0007,
    TPM_ALG_KEYEDHASH       = 0x0008,
    TPM_ALG_XOR             = 0x000A,
    TPM_ALG_SHA256          = 0x000B,
    TPM_ALG_SHA384          = 0x000C,
    TPM_ALG_SHA512          = 0x000D,
    TPM_ALG_NULL            = 0x0010,
    TPM_ALG_SM3_256         = 0x0012,
    TPM_ALG_SM4             = 0x0013,
    TPM_ALG_RSASSA          = 0x0014,
    TPM_ALG_RSAES           = 0x0015,
    TPM_ALG_RSAPSS          = 0x0016,
    TPM_ALG_OAEP            = 0x0017,
    TPM_ALG_ECDSA           = 0x0018,
    TPM_ALG_ECDH            = 0x0019,
    TPM_ALG_ECDAA           = 0x001A,
    TPM_ALG_SM2             = 0x001B,
    TPM_ALG_ECSCHNORR       = 0x001C,
    TPM_ALG_ECMQV           = 0x001D,
    TPM_ALG_HKDF            = 0x001F, /* IETF RFC 5869, v1.85 */
    TPM_ALG_KDF1_SP800_56A  = 0x0020,
    TPM_ALG_KDF2            = 0x0021,
    TPM_ALG_KDF1_SP800_108  = 0x0022,
    TPM_ALG_ECC             = 0x0023,
    TPM_ALG_SYMCIPHER       = 0x0025,
```

```
    TPM_ALG_CAMELLIA        = 0x0026,
    TPM_ALG_SHA3_256        = 0x0027,
    TPM_ALG_SHA3_384        = 0x0028,
    TPM_ALG_SHA3_512        = 0x0029,
    TPM_ALG_SHAKE128        = 0x002A,
    TPM_ALG_SHAKE256        = 0x002B,
    TPM_ALG_CTR             = 0x0040,
    TPM_ALG_OFB             = 0x0041,
    TPM_ALG_CBC             = 0x0042,
    TPM_ALG_CFB             = 0x0043,
    TPM_ALG_ECB             = 0x0044,
#ifdef WOLFTPM_PQC
    /* Post-Quantum Algorithms - TPM 2.0 Library v185 */
    TPM_ALG_MLKEM           = 0x00A0,
    TPM_ALG_MLDSA           = 0x00A1,
    TPM_ALG_HASH_MLDSA      = 0x00A2,
#endif
} TPM_ALG_ID_T;
typedef UINT16 TPM_ALG_ID;

typedef enum {
    TPM_ECC_NONE        = 0x0000,
    TPM_ECC_NIST_P192   = 0x0001,
    TPM_ECC_NIST_P224   = 0x0002,
    TPM_ECC_NIST_P256   = 0x0003,
    TPM_ECC_NIST_P384   = 0x0004,
    TPM_ECC_NIST_P521   = 0x0005,
    TPM_ECC_BN_P256     = 0x0010,
    TPM_ECC_BN_P638     = 0x0011,
    TPM_ECC_SM2_P256    = 0x0020,
    TPM_ECC_BP_P256_R1  = 0x0030,
    TPM_ECC_BP_P384_R1  = 0x0031,
    TPM_ECC_BP_P512_R1  = 0x0032,
} TPM_ECC_CURVE_T;
typedef UINT16 TPM_ECC_CURVE;

#ifdef WOLFTPM_PQC
/* ML-KEM Parameter Sets (TCG Algorithm Registry v2.0) */
typedef UINT16 TPMI_MLKEM_PARAMETER_SET;
#define TPM_MLKEM_NONE              0x0000
#define TPM_MLKEM_512               0x0001
#define TPM_MLKEM_768               0x0002
#define TPM_MLKEM_1024              0x0003

/* ML-DSA Parameter Sets (TCG Algorithm Registry v2.0) */
typedef UINT16 TPMI_MLDSA_PARAMETER_SET;
#define TPM_MLDSA_NONE              0x0000
#define TPM_MLDSA_44                0x0001
#define TPM_MLDSA_65                0x0002
#define TPM_MLDSA_87                0x0003
#endif /* WOLFTPM_PQC */

/* Command Codes */
typedef enum {
```

```
    TPM_CC_FIRST                    = 0x0000011F,
    TPM_CC_NV_UndefineSpaceSpecial  = TPM_CC_FIRST,
    TPM_CC_EvictControl             = 0x00000120,
    TPM_CC_HierarchyControl         = 0x00000121,
    TPM_CC_NV_UndefineSpace         = 0x00000122,
    TPM_CC_ChangeEPS                = 0x00000124,
    TPM_CC_ChangePPS                = 0x00000125,
    TPM_CC_Clear                    = 0x00000126,
    TPM_CC_ClearControl             = 0x00000127,
    TPM_CC_ClockSet                 = 0x00000128,
    TPM_CC_HierarchyChangeAuth      = 0x00000129,
    TPM_CC_NV_DefineSpace           = 0x0000012A,
    TPM_CC_PCR_Allocate             = 0x0000012B,
    TPM_CC_PCR_SetAuthPolicy        = 0x0000012C,
    TPM_CC_PP_Commands              = 0x0000012D,
    TPM_CC_SetPrimaryPolicy         = 0x0000012E,
    TPM_CC_FieldUpgradeStart        = 0x0000012F,
    TPM_CC_ClockRateAdjust          = 0x00000130,
    TPM_CC_CreatePrimary            = 0x00000131,
    TPM_CC_NV_GlobalWriteLock       = 0x00000132,
    TPM_CC_GetCommandAuditDigest    = 0x00000133,
    TPM_CC_NV_Increment             = 0x00000134,
    TPM_CC_NV_SetBits               = 0x00000135,
    TPM_CC_NV_Extend                = 0x00000136,
    TPM_CC_NV_Write                 = 0x00000137,
    TPM_CC_NV_WriteLock             = 0x00000138,
    TPM_CC_DictionaryAttackLockReset = 0x00000139,
    TPM_CC_DictionaryAttackParameters = 0x0000013A,
    TPM_CC_NV_ChangeAuth            = 0x0000013B,
    TPM_CC_PCR_Event                = 0x0000013C,
    TPM_CC_PCR_Reset                = 0x0000013D,
    TPM_CC_SequenceComplete         = 0x0000013E,
    TPM_CC_SetAlgorithmSet          = 0x0000013F,
    TPM_CC_SetCommandCodeAuditStatus = 0x00000140,
    TPM_CC_FieldUpgradeData         = 0x00000141,
    TPM_CC_IncrementalSelfTest      = 0x00000142,
    TPM_CC_SelfTest                 = 0x00000143,
    TPM_CC_Startup                  = 0x00000144,
    TPM_CC_Shutdown                 = 0x00000145,
    TPM_CC_StirRandom               = 0x00000146,
    TPM_CC_ActivateCredential       = 0x00000147,
    TPM_CC_Certify                  = 0x00000148,
    TPM_CC_PolicyNV                 = 0x00000149,
    TPM_CC_CertifyCreation          = 0x0000014A,
    TPM_CC_Duplicate                = 0x0000014B,
    TPM_CC_GetTime                  = 0x0000014C,
    TPM_CC_GetSessionAuditDigest    = 0x0000014D,
    TPM_CC_NV_Read                  = 0x0000014E,
    TPM_CC_NV_ReadLock              = 0x0000014F,
    TPM_CC_ObjectChangeAuth         = 0x00000150,
    TPM_CC_PolicySecret             = 0x00000151,
    TPM_CC_Rewrap                   = 0x00000152,
    TPM_CC_Create                   = 0x00000153,
    TPM_CC_ECDH_ZGen                = 0x00000154,
```

```
    TPM_CC_HMAC                     = 0x00000155,
    TPM_CC_Import                   = 0x00000156,
    TPM_CC_Load                     = 0x00000157,
    TPM_CC_Quote                    = 0x00000158,
    TPM_CC_RSA_Decrypt              = 0x00000159,
    TPM_CC_HMAC_Start               = 0x0000015B,
    TPM_CC_SequenceUpdate           = 0x0000015C,
    TPM_CC_Sign                     = 0x0000015D,
    TPM_CC_Unseal                   = 0x0000015E,
    TPM_CC_PolicySigned             = 0x00000160,
    TPM_CC_ContextLoad              = 0x00000161,
    TPM_CC_ContextSave              = 0x00000162,
    TPM_CC_ECDH_KeyGen              = 0x00000163,
    TPM_CC_EncryptDecrypt           = 0x00000164,
    TPM_CC_FlushContext             = 0x00000165,
    TPM_CC_LoadExternal             = 0x00000167,
    TPM_CC_MakeCredential           = 0x00000168,
    TPM_CC_NV_ReadPublic            = 0x00000169,
    TPM_CC_PolicyAuthorize          = 0x0000016A,
    TPM_CC_PolicyAuthValue          = 0x0000016B,
    TPM_CC_PolicyCommandCode        = 0x0000016C,
    TPM_CC_PolicyCounterTimer       = 0x0000016D,
    TPM_CC_PolicyCpHash             = 0x0000016E,
    TPM_CC_PolicyLocality           = 0x0000016F,
    TPM_CC_PolicyNameHash           = 0x00000170,
    TPM_CC_PolicyOR                 = 0x00000171,
    TPM_CC_PolicyTicket             = 0x00000172,
    TPM_CC_ReadPublic               = 0x00000173,
    TPM_CC_RSA_Encrypt              = 0x00000174,
    TPM_CC_StartAuthSession         = 0x00000176,
    TPM_CC_VerifySignature          = 0x00000177,
    TPM_CC_ECC_Parameters           = 0x00000178,
    TPM_CC_FirmwareRead             = 0x00000179,
    TPM_CC_GetCapability            = 0x0000017A,
    TPM_CC_GetRandom                = 0x0000017B,
    TPM_CC_GetTestResult            = 0x0000017C,
    TPM_CC_Hash                     = 0x0000017D,
    TPM_CC_PCR_Read                 = 0x0000017E,
    TPM_CC_PolicyPCR                = 0x0000017F,
    TPM_CC_PolicyRestart            = 0x00000180,
    TPM_CC_ReadClock                = 0x00000181,
    TPM_CC_PCR_Extend               = 0x00000182,
    TPM_CC_PCR_SetAuthValue         = 0x00000183,
    TPM_CC_NV_Certify               = 0x00000184,
    TPM_CC_EventSequenceComplete    = 0x00000185,
    TPM_CC_HashSequenceStart        = 0x00000186,
    TPM_CC_PolicyPhysicalPresence   = 0x00000187,
    TPM_CC_PolicyDuplicationSelect  = 0x00000188,
    TPM_CC_PolicyGetDigest          = 0x00000189,
    TPM_CC_TestParms                = 0x0000018A,
    TPM_CC_Commit                   = 0x0000018B,
    TPM_CC_PolicyPassword           = 0x0000018C,
    TPM_CC_ZGen_2Phase              = 0x0000018D,
    TPM_CC_EC_Ephemeral             = 0x0000018E,
```

```
    TPM_CC_PolicyNvWritten          = 0x0000018F,
    TPM_CC_PolicyTemplate           = 0x00000190,
    TPM_CC_CreateLoaded             = 0x00000191,
    TPM_CC_PolicyAuthorizeNV        = 0x00000192,
    TPM_CC_EncryptDecrypt2          = 0x00000193,
    /* TPM 2.0 Library v1.85: settable feature configuration. Used by some
     * Infineon parts to enable/lock SPDM-PSK and other features. */
    TPM_CC_SetCapability            = 0x0000019F,
#ifdef WOLFTPM_SPDM
    TPM_CC_PolicyTransportSPDM      = 0x000001A1,
#endif
#ifdef WOLFTPM_PQC
    /* Post-Quantum Cryptography Commands - TPM 2.0 Library v185 */
    TPM_CC_VerifySequenceComplete   = 0x000001A3,
    TPM_CC_SignSequenceComplete     = 0x000001A4,
    TPM_CC_VerifyDigestSignature    = 0x000001A5,
    TPM_CC_SignDigest               = 0x000001A6,
    TPM_CC_Encapsulate              = 0x000001A7,
    TPM_CC_Decapsulate              = 0x000001A8,
    TPM_CC_VerifySequenceStart      = 0x000001A9,
    TPM_CC_SignSequenceStart        = 0x000001AA,
    TPM_CC_LAST                     = TPM_CC_SignSequenceStart,
#elif defined(WOLFTPM_SPDM)
    TPM_CC_LAST                     = TPM_CC_PolicyTransportSPDM,
#else
    TPM_CC_LAST                     = TPM_CC_SetCapability,
#endif

    CC_VEND                         = 0x20000000,
    TPM_CC_Vendor_TCG_Test          = CC_VEND + 0x0000,
#if defined(WOLFTPM_ST33) || defined(WOLFTPM_AUTODETECT)
    TPM_CC_SetMode                  = CC_VEND + 0x0307,
    TPM_CC_SetCommandSet            = CC_VEND + 0x0309,
    TPM_CC_GetRandom2               = CC_VEND + 0x030E,
#endif
#ifdef WOLFTPM_ST33
    TPM_CC_RestoreEK                = CC_VEND + 0x030A,
    TPM_CC_SetCommandSetLock        = CC_VEND + 0x030B,
    TPM_CC_GPIO_Config              = CC_VEND + 0x030F,
#endif
#if defined(WOLFTPM_NUVOTON) || defined(WOLFTPM_AUTODETECT)
    TPM_CC_NTC2_PreConfig           = CC_VEND + 0x0211,
    TPM_CC_NTC2_GetConfig           = CC_VEND + 0x0213,
#endif
#if defined(WOLFTPM_NATIONS) || defined(WOLFTPM_AUTODETECT)
    TPM_CC_Nations_IdentityKeySet   = 0x20000708,
#endif
#ifdef WOLFTPM_IFX_FW_UPGRADE
    TPM_CC_FieldUpgradeStartVendor    = CC_VEND + 0x12F,
    TPM_CC_FieldUpgradeAbandonVendor  = CC_VEND + 0x130,
    TPM_CC_FieldUpgradeManifestVendor = CC_VEND + 0x131,
    TPM_CC_FieldUpgradeDataVendor     = CC_VEND + 0x132,
    TPM_CC_FieldUpgradeFinalizeVendor = CC_VEND + 0x133,
#endif
```

```
#if defined(WOLFTPM_ST33) || defined(WOLFTPM_AUTODETECT)
    /* ST33 Firmware Update Vendor Command Codes
     * Verified from ST reference implementation (TPM_FU_STM_KTPM_LMS.c):
     * - Start: 0x2000030C, Data: 0x2000030D
     * Note: Abandon still uses placeholder - may need verification
     * These are only used by the ST33KTPM class of firmware. Other ST33
     * firmware performs the field upgrade with the standard
     * TPM_CC_FieldUpgradeStart/Data codes instead; the library queries
     * TPM_CAP_COMMANDS to find out which pair a part implements.
     * Note the abandon/finalize codes alias TPM_CC_GetRandom2 (0x2000030E)
     * and TPM_CC_GPIO_Config (0x2000030F); only the firmware upgrade mode the
     * TPM is in tells the two meanings apart. */
    TPM_CC_FieldUpgradeStartVendor_ST33    = CC_VEND + 0x030C,
    TPM_CC_FieldUpgradeAbandonVendor_ST33  = CC_VEND + 0x030E, /*
    ↪  Abandon/cancel */
    TPM_CC_FieldUpgradeDataVendor_ST33     = CC_VEND + 0x030D,
    TPM_CC_FieldUpgradeFinalizeVendor_ST33 = CC_VEND + 0x030F, /* Finalize if
    ↪  needed */
#endif
} TPM_CC_T;
typedef UINT32 TPM_CC;

/* Response Codes */
typedef enum {
    TPM_RC_SUCCESS  = 0x000,
    TPM_RC_BAD_TAG  = 0x01E,

    RC_VER1 = 0x100,
    TPM_RC_INITIALIZE           = RC_VER1 + 0x000,
    TPM_RC_FAILURE              = RC_VER1 + 0x001,
    TPM_RC_SEQUENCE             = RC_VER1 + 0x003,
    TPM_RC_PRIVATE              = RC_VER1 + 0x00B,
    TPM_RC_HMAC                 = RC_VER1 + 0x019,
    TPM_RC_DISABLED             = RC_VER1 + 0x020,
    TPM_RC_EXCLUSIVE            = RC_VER1 + 0x021,
    TPM_RC_AUTH_TYPE            = RC_VER1 + 0x024,
    TPM_RC_AUTH_MISSING         = RC_VER1 + 0x025,
    TPM_RC_POLICY               = RC_VER1 + 0x026,
    TPM_RC_PCR                  = RC_VER1 + 0x027,
    TPM_RC_PCR_CHANGED          = RC_VER1 + 0x028,
    TPM_RC_UPGRADE              = RC_VER1 + 0x02D,
    TPM_RC_TOO_MANY_CONTEXTS    = RC_VER1 + 0x02E,
    TPM_RC_AUTH_UNAVAILABLE     = RC_VER1 + 0x02F,
    TPM_RC_REBOOT               = RC_VER1 + 0x030,
    TPM_RC_UNBALANCED           = RC_VER1 + 0x031,
    TPM_RC_COMMAND_SIZE         = RC_VER1 + 0x042,
    TPM_RC_COMMAND_CODE         = RC_VER1 + 0x043,
    TPM_RC_AUTHSIZE             = RC_VER1 + 0x044,
    TPM_RC_AUTH_CONTEXT         = RC_VER1 + 0x045,
    TPM_RC_NV_RANGE             = RC_VER1 + 0x046,
    TPM_RC_NV_SIZE              = RC_VER1 + 0x047,
    TPM_RC_NV_LOCKED            = RC_VER1 + 0x048,
    TPM_RC_NV_AUTHORIZATION     = RC_VER1 + 0x049,
    TPM_RC_NV_UNINITIALIZED     = RC_VER1 + 0x04A,
```

```
    TPM_RC_NV_SPACE             = RC_VER1 + 0x04B,
    TPM_RC_NV_DEFINED           = RC_VER1 + 0x04C,
    TPM_RC_BAD_CONTEXT          = RC_VER1 + 0x050,
    TPM_RC_CPHASH               = RC_VER1 + 0x051,
    TPM_RC_PARENT               = RC_VER1 + 0x052,
    TPM_RC_NEEDS_TEST           = RC_VER1 + 0x053,
    TPM_RC_NO_RESULT            = RC_VER1 + 0x054,
    TPM_RC_SENSITIVE            = RC_VER1 + 0x055,
    RC_MAX_FM0                  = RC_VER1 + 0x07F,

    RC_FMT1 = 0x080,
    TPM_RC_ASYMMETRIC       = RC_FMT1 + 0x001,
    TPM_RC_ATTRIBUTES       = RC_FMT1 + 0x002,
    TPM_RC_HASH             = RC_FMT1 + 0x003,
    TPM_RC_VALUE            = RC_FMT1 + 0x004,
    TPM_RC_HIERARCHY        = RC_FMT1 + 0x005,
    TPM_RC_KEY_SIZE         = RC_FMT1 + 0x007,
    TPM_RC_MGF              = RC_FMT1 + 0x008,
    TPM_RC_MODE             = RC_FMT1 + 0x009,
    TPM_RC_TYPE             = RC_FMT1 + 0x00A,
    TPM_RC_HANDLE           = RC_FMT1 + 0x00B,
    TPM_RC_KDF              = RC_FMT1 + 0x00C,
    TPM_RC_RANGE            = RC_FMT1 + 0x00D,
    TPM_RC_AUTH_FAIL        = RC_FMT1 + 0x00E,
    TPM_RC_NONCE            = RC_FMT1 + 0x00F,
    TPM_RC_PP               = RC_FMT1 + 0x010,
    TPM_RC_SCHEME           = RC_FMT1 + 0x012,
    TPM_RC_SIZE             = RC_FMT1 + 0x015,
    TPM_RC_SYMMETRIC        = RC_FMT1 + 0x016,
    TPM_RC_TAG              = RC_FMT1 + 0x017,
    TPM_RC_SELECTOR         = RC_FMT1 + 0x018,
    TPM_RC_INSUFFICIENT     = RC_FMT1 + 0x01A,
    TPM_RC_SIGNATURE        = RC_FMT1 + 0x01B,
    TPM_RC_KEY              = RC_FMT1 + 0x01C,
    TPM_RC_POLICY_FAIL      = RC_FMT1 + 0x01D,
    TPM_RC_INTEGRITY        = RC_FMT1 + 0x01F,
    TPM_RC_TICKET           = RC_FMT1 + 0x020,
    TPM_RC_RESERVED_BITS    = RC_FMT1 + 0x021,
    TPM_RC_BAD_AUTH         = RC_FMT1 + 0x022,
    TPM_RC_EXPIRED          = RC_FMT1 + 0x023,
    TPM_RC_POLICY_CC        = RC_FMT1 + 0x024,
    TPM_RC_BINDING          = RC_FMT1 + 0x025,
    TPM_RC_CURVE            = RC_FMT1 + 0x026,
    TPM_RC_ECC_POINT        = RC_FMT1 + 0x027,
#ifdef WOLFTPM_V185
    /* Part 2 v1.85 Sec.6.6.3 Table 17 firmware/SVN-limited codes */
    TPM_RC_FW_LIMITED         = RC_FMT1 + 0x028,
    TPM_RC_SVN_LIMITED        = RC_FMT1 + 0x029,
#endif
    /* TCG Part 2 Sec.6.6.3 Table 17 -- present since v1.16, not v1.85 */
    TPM_RC_PARMS              = RC_FMT1 + 0x02A,
#ifdef WOLFTPM_PQC
    /* v185 rc4 Part 2 Sec.6.6.3 Table 17 additions */
    TPM_RC_EXT_MU             = RC_FMT1 + 0x02B,
```

```
    TPM_RC_ONE_SHOT_SIGNATURE = RC_FMT1 + 0x02C,
    TPM_RC_SIGN_CONTEXT_KEY   = RC_FMT1 + 0x02D,
#endif
#if defined(WOLFTPM_V185) || defined(WOLFTPM_SPDM)
    /* Part 2 v1.85 Sec.6.6.3 Table 17 channel protocol codes */
    TPM_RC_CHANNEL            = RC_FMT1 + 0x030,
    TPM_RC_CHANNEL_KEY        = RC_FMT1 + 0x031,
#endif
    RC_MAX_FMT1             = RC_FMT1 + 0x03F,

    RC_WARN = 0x900,
    TPM_RC_CONTEXT_GAP      = RC_WARN + 0x001,
    TPM_RC_OBJECT_MEMORY    = RC_WARN + 0x002,
    TPM_RC_SESSION_MEMORY   = RC_WARN + 0x003,
    TPM_RC_MEMORY           = RC_WARN + 0x004,
    TPM_RC_SESSION_HANDLES  = RC_WARN + 0x005,
    TPM_RC_OBJECT_HANDLES   = RC_WARN + 0x006,
    TPM_RC_LOCALITY         = RC_WARN + 0x007,
    TPM_RC_YIELDED          = RC_WARN + 0x008,
    TPM_RC_CANCELED         = RC_WARN + 0x009,
    TPM_RC_TESTING          = RC_WARN + 0x00A,
    TPM_RC_REFERENCE_H0     = RC_WARN + 0x010,
    TPM_RC_REFERENCE_H1     = RC_WARN + 0x011,
    TPM_RC_REFERENCE_H2     = RC_WARN + 0x012,
    TPM_RC_REFERENCE_H3     = RC_WARN + 0x013,
    TPM_RC_REFERENCE_H4     = RC_WARN + 0x014,
    TPM_RC_REFERENCE_H5     = RC_WARN + 0x015,
    TPM_RC_REFERENCE_H6     = RC_WARN + 0x016,
    TPM_RC_REFERENCE_S0     = RC_WARN + 0x018,
    TPM_RC_REFERENCE_S1     = RC_WARN + 0x019,
    TPM_RC_REFERENCE_S2     = RC_WARN + 0x01A,
    TPM_RC_REFERENCE_S3     = RC_WARN + 0x01B,
    TPM_RC_REFERENCE_S4     = RC_WARN + 0x01C,
    TPM_RC_REFERENCE_S5     = RC_WARN + 0x01D,
    TPM_RC_REFERENCE_S6     = RC_WARN + 0x01E,
    TPM_RC_NV_RATE          = RC_WARN + 0x020,
    TPM_RC_LOCKOUT          = RC_WARN + 0x021,
    TPM_RC_RETRY            = RC_WARN + 0x022,
    TPM_RC_NV_UNAVAILABLE   = RC_WARN + 0x023,
    RC_MAX_WARN             = RC_WARN + 0x03F,

    TPM_RC_NOT_USED         = RC_WARN + 0x07F,

    TPM_RC_H        = 0x000,
    TPM_RC_P        = 0x040,
    TPM_RC_S        = 0x800,
    TPM_RC_1        = 0x100,
    TPM_RC_2        = 0x200,
    TPM_RC_3        = 0x300,
    TPM_RC_4        = 0x400,
    TPM_RC_5        = 0x500,
    TPM_RC_6        = 0x600,
    TPM_RC_7        = 0x700,
    TPM_RC_8        = 0x800,
```

```
    TPM_RC_9        = 0x900,
    TPM_RC_A        = 0xA00,
    TPM_RC_B        = 0xB00,
    TPM_RC_C        = 0xC00,
    TPM_RC_D        = 0xD00,
    TPM_RC_E        = 0xE00,
    TPM_RC_F        = 0xF00,
    TPM_RC_N_MASK   = 0xF00,

    /* use negative codes for internal errors */
    TPM_RC_TIMEOUT = -100,
} TPM_RC_T;
typedef INT32 TPM_RC; /* type is unsigned 16-bits, but internally use signed
 ↪  32-bit */

typedef enum {
    TPM_CLOCK_COARSE_SLOWER = -3,
    TPM_CLOCK_MEDIUM_SLOWER = -2,
    TPM_CLOCK_FINE_SLOWER   = -1,
    TPM_CLOCK_NO_CHANGE     = 0,
    TPM_CLOCK_FINE_FASTER   = 1,
    TPM_CLOCK_MEDIUM_FASTER = 2,
    TPM_CLOCK_COARSE_FASTER = 3,
} TPM_CLOCK_ADJUST_T;
typedef UINT8 TPM_CLOCK_ADJUST;

/* EA Arithmetic Operands */
typedef enum {
    TPM_EO_EQ           = 0x0000,
    TPM_EO_NEQ          = 0x0001,
    TPM_EO_SIGNED_GT    = 0x0002,
    TPM_EO_UNSIGNED_GT  = 0x0003,
    TPM_EO_SIGNED_LT    = 0x0004,
    TPM_EO_UNSIGNED_LT  = 0x0005,
    TPM_EO_SIGNED_GE    = 0x0006,
    TPM_EO_UNSIGNED_GE  = 0x0007,
    TPM_EO_SIGNED_LE    = 0x0008,
    TPM_EO_UNSIGNED_LE  = 0x0009,
    TPM_EO_BITSET       = 0x000A,
    TPM_EO_BITCLEAR     = 0x000B,
} TPM_EO_T;
typedef UINT16 TPM_EO;

/* Structure Tags */
typedef enum {
    TPM_ST_RSP_COMMAND          = 0x00C4,
    TPM_ST_NULL                 = 0X8000,
    TPM_ST_NO_SESSIONS          = 0x8001,
    TPM_ST_SESSIONS             = 0x8002,
    TPM_ST_ATTEST_NV            = 0x8014,
    TPM_ST_ATTEST_COMMAND_AUDIT = 0x8015,
    TPM_ST_ATTEST_SESSION_AUDIT = 0x8016,
    TPM_ST_ATTEST_CERTIFY       = 0x8017,
    TPM_ST_ATTEST_QUOTE         = 0x8018,
```

```
    TPM_ST_ATTEST_TIME          = 0x8019,
    TPM_ST_ATTEST_CREATION      = 0x801A,
    TPM_ST_ATTEST_NV_DIGEST     = 0x801C,
    TPM_ST_CREATION             = 0x8021,
    TPM_ST_VERIFIED             = 0x8022,
    TPM_ST_AUTH_SECRET          = 0x8023,
    TPM_ST_HASHCHECK            = 0x8024,
    TPM_ST_AUTH_SIGNED          = 0x8025,
#ifdef WOLFTPM_PQC
    TPM_ST_MESSAGE_VERIFIED     = 0x8026,
    TPM_ST_DIGEST_VERIFIED      = 0x8027,
#endif
    TPM_ST_FU_MANIFEST          = 0x8029,
} TPM_ST_T;
typedef UINT16 TPM_ST;

/* Session Type */
typedef enum {
    TPM_SE_HMAC     = 0x00,
    TPM_SE_POLICY   = 0x01,
    TPM_SE_TRIAL    = 0x03,
} TPM_SE_T;
typedef UINT8 TPM_SE;


/* Startup Type */
typedef enum {
    TPM_SU_CLEAR = 0x0000,
    TPM_SU_STATE = 0x0001,
} TPM_SU_T;
typedef UINT16 TPM_SU;

/* Capabilities */
typedef enum {
    TPM_CAP_FIRST           = 0x00000000,
    TPM_CAP_ALGS            = TPM_CAP_FIRST,
    TPM_CAP_HANDLES         = 0x00000001,
    TPM_CAP_COMMANDS        = 0x00000002,
    TPM_CAP_PP_COMMANDS     = 0x00000003,
    TPM_CAP_AUDIT_COMMANDS  = 0x00000004,
    TPM_CAP_PCRS            = 0x00000005,
    TPM_CAP_TPM_PROPERTIES  = 0x00000006,
    TPM_CAP_PCR_PROPERTIES  = 0x00000007,
    TPM_CAP_ECC_CURVES      = 0x00000008,
    TPM_CAP_AUTH_POLICIES   = 0x00000009,
    TPM_CAP_ACT             = 0x0000000A,
#if defined(WOLFTPM_NATIONS) || defined(WOLFTPM_AUTODETECT) || \
    defined(WOLFTPM_SPDM)
    TPM_CAP_PUB_KEYS        = 0x0000000B, /* TPM 184: SPDM identity keys */
    TPM_CAP_SPDM_SESSION_INFO = 0x0000000C, /* TPM 184: SPDM session info */
    TPM_CAP_LAST            = TPM_CAP_SPDM_SESSION_INFO,
#else
    TPM_CAP_LAST            = TPM_CAP_ACT,
#endif
```

```
    TPM_CAP_VENDOR_PROPERTY = 0x00000100,
} TPM_CAP_T;
typedef UINT32 TPM_CAP;

/* Settable feature configuration (TPM 2.0 Library v1.85). The first byte
 * (0x81) encodes the authorization (Platform), the second (0xC9) the vendor
 * (Infineon). Used with TPM2_SetCapability / TPM2_GetCapability. Kept out of
 * TPM_CAP_T because ISO C limits enumerators to the range of int. */
#define TPM_CAP_CONFIGS 0x81C90001U

/* TPM_PT_CONFIG: property identifiers for settable feature configuration,
 * used inside TPMS_TAGGED_CONFIG (TPM_CAP_CONFIGS). */
typedef enum {
    TPM_PT_CONFIG_ENCRYPTDECRYPT2 = 0x00000000,
    TPM_PT_CONFIG_CHANGEEPS       = 0x00000001,
    TPM_PT_CONFIG_TPMID_NV        = 0x00000002,
    TPM_PT_CONFIG_SPDM_PSK        = 0x00000003,
} TPM_PT_CONFIG_T;
typedef UINT32 TPM_PT_CONFIG;

#ifdef WOLFTPM_SPDM
/* TCG SPDM Binding for Secure Communication v1.0 Constants */

/* TCG SPDM Binding Message Tags */
#define SPDM_TAG_CLEAR      0x8101  /* Clear (unencrypted) SPDM message */
#define SPDM_TAG_SECURED    0x8201  /* Secured (encrypted) SPDM message */

/* SPDM Protocol Version */
#define SPDM_VERSION_1_3    0x13    /* SPDM v1.3 */

/* SPDM protocol message codes, response codes, and error codes are
 * defined in <wolftpm/spdm/spdm_types.h> (the authoritative source).
 * Include that header directly if you need SPDM protocol constants. */

/* SPDM Vendor Defined Codes (8-byte ASCII, used as VdCode in VENDOR_DEFINED) */
#define SPDM_VDCODE_TPM2_CMD   "TPM2_CMD"  /* TPM command over SPDM session */
#define SPDM_VDCODE_GET_PUBK   "GET_PUBK"  /* Get TPM's SPDM-Identity pub key */
#define SPDM_VDCODE_GIVE_PUB   "GIVE_PUB"  /* Give host's SPDM-Identity pub key
 ↪  */
#define SPDM_VDCODE_SPDMONLY   "SPDMONLY"  /* Lock/unlock SPDM-only mode */
#define SPDM_VDCODE_GET_STS    "GET_STS_"  /* Get SPDM status */

/* SPDM Vendor Defined Code Length */
#define SPDM_VDCODE_LEN        8

/* SPDM Session Constants (Nuvoton NPCT7xx) */
#define SPDM_CONNECTION_ID      0       /* Single connection */
#define SPDM_RSP_SESSION_ID     0xAEAD  /* Responder session ID */
#define SPDM_REQ_SESSION_ID     0x0001  /* Default requester session ID */

/* SPDM FIPS Indicator (TCG binding) */
#define SPDM_FIPS_NON_FIPS     0x00
#define SPDM_FIPS_APPROVED     0x01
```

```
/* SPDM Algorithm Set B (192-bit security strength) */
#define SPDM_ALG_ECDSA_P384    0x0003  /* Signing algorithm */
#define SPDM_ALG_SHA384        0x0002  /* Hash algorithm */
#define SPDM_ALG_ECDHE_P384    0x0003  /* Key exchange algorithm */
#define SPDM_ALG_AES256_GCM    0x0002  /* AEAD algorithm */

/* SPDM-Identity NV Indices */
#define SPDM_NV_INDEX_TPM_KEY  0x01C20110  /* TPM SPDM-Identity key */
#define SPDM_NV_INDEX_REQ_KEY  0x01C20111  /* Requester SPDM-Identity key */

/* NTC2 PreConfig CFG_H Bit Definitions for SPDM */
#define NTC2_CFG_H_SPDM_ENABLE_BIT    1  /* Bit position in CFG_H */
#define NTC2_CFG_H_SPDM_ENABLE        0x00  /* SPDM enabled (bit 1 = 0) */
#define NTC2_CFG_H_SPDM_DISABLE       0x02  /* SPDM disabled (bit 1 = 1) */

/* SPDM ONLY mode sub-commands */
#define SPDM_ONLY_LOCK         0x01
#define SPDM_ONLY_UNLOCK       0x00

/* SPDM Message Sizes */
#define SPDM_MAX_MSG_SIZE       4096

/* TCG SPDM Binding Header Size (per TCG SPDM Binding Spec):
 * tag(2/BE) + size(4/BE) + connectionHandle(4/BE) + fipsIndicator(2/BE) +
 * reserved(4) = 16 bytes */
#define SPDM_TCG_BINDING_HEADER_SIZE  16

/* SPDM Secured Message Header Size (per DSP0277):
 * sessionId(4/LE) + sequenceNumber(8/LE) + length(2/LE) = 14 bytes
 * where length = size of encrypted data + MAC */
#define SPDM_SECURED_MSG_HEADER_SIZE  14

#endif /* WOLFTPM_SPDM */

/* Property Tag */
typedef enum {
    TPM_PT_NONE    = 0x00000000,
    PT_GROUP       = 0x00000100,

    PT_FIXED = PT_GROUP * 1,
    TPM_PT_FAMILY_INDICATOR     = PT_FIXED + 0,
    TPM_PT_LEVEL                = PT_FIXED + 1,
    TPM_PT_REVISION             = PT_FIXED + 2,
    TPM_PT_DAY_OF_YEAR          = PT_FIXED + 3,
    TPM_PT_YEAR                 = PT_FIXED + 4,
    TPM_PT_MANUFACTURER         = PT_FIXED + 5,
    TPM_PT_VENDOR_STRING_1      = PT_FIXED + 6,
    TPM_PT_VENDOR_STRING_2      = PT_FIXED + 7,
    TPM_PT_VENDOR_STRING_3      = PT_FIXED + 8,
    TPM_PT_VENDOR_STRING_4      = PT_FIXED + 9,
    TPM_PT_VENDOR_TPM_TYPE      = PT_FIXED + 10,
    TPM_PT_FIRMWARE_VERSION_1   = PT_FIXED + 11,
    TPM_PT_FIRMWARE_VERSION_2   = PT_FIXED + 12,
```

```
    TPM_PT_INPUT_BUFFER          = PT_FIXED + 13,
    TPM_PT_HR_TRANSIENT_MIN      = PT_FIXED + 14,
    TPM_PT_HR_PERSISTENT_MIN     = PT_FIXED + 15,
    TPM_PT_HR_LOADED_MIN         = PT_FIXED + 16,
    TPM_PT_ACTIVE_SESSIONS_MAX   = PT_FIXED + 17,
    TPM_PT_PCR_COUNT             = PT_FIXED + 18,
    TPM_PT_PCR_SELECT_MIN        = PT_FIXED + 19,
    TPM_PT_CONTEXT_GAP_MAX       = PT_FIXED + 20,
    TPM_PT_NV_COUNTERS_MAX       = PT_FIXED + 22,
    TPM_PT_NV_INDEX_MAX          = PT_FIXED + 23,
    TPM_PT_MEMORY                = PT_FIXED + 24,
    TPM_PT_CLOCK_UPDATE          = PT_FIXED + 25,
    TPM_PT_CONTEXT_HASH          = PT_FIXED + 26,
    TPM_PT_CONTEXT_SYM           = PT_FIXED + 27,
    TPM_PT_CONTEXT_SYM_SIZE      = PT_FIXED + 28,
    TPM_PT_ORDERLY_COUNT         = PT_FIXED + 29,
    TPM_PT_MAX_COMMAND_SIZE      = PT_FIXED + 30,
    TPM_PT_MAX_RESPONSE_SIZE     = PT_FIXED + 31,
    TPM_PT_MAX_DIGEST            = PT_FIXED + 32,
    TPM_PT_MAX_OBJECT_CONTEXT    = PT_FIXED + 33,
    TPM_PT_MAX_SESSION_CONTEXT   = PT_FIXED + 34,
    TPM_PT_PS_FAMILY_INDICATOR   = PT_FIXED + 35,
    TPM_PT_PS_LEVEL              = PT_FIXED + 36,
    TPM_PT_PS_REVISION           = PT_FIXED + 37,
    TPM_PT_PS_DAY_OF_YEAR        = PT_FIXED + 38,
    TPM_PT_PS_YEAR               = PT_FIXED + 39,
    TPM_PT_SPLIT_MAX             = PT_FIXED + 40,
    TPM_PT_TOTAL_COMMANDS        = PT_FIXED + 41,
    TPM_PT_LIBRARY_COMMANDS      = PT_FIXED + 42,
    TPM_PT_VENDOR_COMMANDS       = PT_FIXED + 43,
    TPM_PT_NV_BUFFER_MAX         = PT_FIXED + 44,
    TPM_PT_MODES                 = PT_FIXED + 45,
    TPM_PT_MAX_CAP_BUFFER        = PT_FIXED + 46,
#ifdef WOLFTPM_V185
    /* v185 rc4 Part 2 Sec.6.13 Table 27 */
    TPM_PT_FIRMWARE_SVN          = PT_FIXED + 47,
    TPM_PT_FIRMWARE_MAX_SVN      = PT_FIXED + 48,
    TPM_PT_ML_PARAMETER_SETS     = PT_FIXED + 49,
#endif

    PT_VAR = PT_GROUP * 2,
    TPM_PT_PERMANENT             = PT_VAR + 0,
    TPM_PT_STARTUP_CLEAR         = PT_VAR + 1,
    TPM_PT_HR_NV_INDEX           = PT_VAR + 2,
    TPM_PT_HR_LOADED             = PT_VAR + 3,
    TPM_PT_HR_LOADED_AVAIL       = PT_VAR + 4,
    TPM_PT_HR_ACTIVE             = PT_VAR + 5,
    TPM_PT_HR_ACTIVE_AVAIL       = PT_VAR + 6,
    TPM_PT_HR_TRANSIENT_AVAIL    = PT_VAR + 7,
    TPM_PT_HR_PERSISTENT         = PT_VAR + 8,
    TPM_PT_HR_PERSISTENT_AVAIL   = PT_VAR + 9,
    TPM_PT_NV_COUNTERS           = PT_VAR + 10,
    TPM_PT_NV_COUNTERS_AVAIL     = PT_VAR + 11,
    TPM_PT_ALGORITHM_SET         = PT_VAR + 12,
```

```
    TPM_PT_LOADED_CURVES        = PT_VAR + 13,
    TPM_PT_LOCKOUT_COUNTER      = PT_VAR + 14,
    TPM_PT_MAX_AUTH_FAIL        = PT_VAR + 15,
    TPM_PT_LOCKOUT_INTERVAL     = PT_VAR + 16,
    TPM_PT_LOCKOUT_RECOVERY     = PT_VAR + 17,
    TPM_PT_NV_WRITE_RECOVERY    = PT_VAR + 18,
    TPM_PT_AUDIT_COUNTER_0      = PT_VAR + 19,
    TPM_PT_AUDIT_COUNTER_1      = PT_VAR + 20,
} TPM_PT_T;
typedef UINT32 TPM_PT;

/* Smallest TPM_PT_INPUT_BUFFER a conformant TPM may report (TCG Part 2), so a
 * parameter at or below it fits without reading the capability. */
#define TPM_MIN_INPUT_BUFFER 1024

/* PCR Property Tag */
typedef enum {
    TPM_PT_PCR_FIRST        = 0x00000000,
    TPM_PT_PCR_SAVE         = TPM_PT_PCR_FIRST,
    TPM_PT_PCR_EXTEND_L0    = 0x00000001,
    TPM_PT_PCR_RESET_L0     = 0x00000002,
    TPM_PT_PCR_EXTEND_L1    = 0x00000003,
    TPM_PT_PCR_RESET_L1     = 0x00000004,
    TPM_PT_PCR_EXTEND_L2    = 0x00000005,
    TPM_PT_PCR_RESET_L2     = 0x00000006,
    TPM_PT_PCR_EXTEND_L3    = 0x00000007,
    TPM_PT_PCR_RESET_L3     = 0x00000008,
    TPM_PT_PCR_EXTEND_L4    = 0x00000009,
    TPM_PT_PCR_RESET_L4     = 0x0000000A,
    TPM_PT_PCR_NO_INCREMENT = 0x00000011,
    TPM_PT_PCR_DRTM_RESET   = 0x00000012,
    TPM_PT_PCR_POLICY       = 0x00000013,
    TPM_PT_PCR_AUTH         = 0x00000014,
    TPM_PT_PCR_LAST         = TPM_PT_PCR_AUTH,
} TPM_PT_PCR_T;
typedef UINT32 TPM_PT_PCR;

/* Platform Specific */
typedef enum {
    TPM_PS_MAIN             = 0x00000000,
    TPM_PS_PC               = 0x00000001,
    TPM_PS_PDA              = 0x00000002,
    TPM_PS_CELL_PHONE       = 0x00000003,
    TPM_PS_SERVER           = 0x00000004,
    TPM_PS_PERIPHERAL       = 0x00000005,
    TPM_PS_TSS              = 0x00000006,
    TPM_PS_STORAGE          = 0x00000007,
    TPM_PS_AUTHENTICATION   = 0x00000008,
    TPM_PS_EMBEDDED         = 0x00000009,
    TPM_PS_HARDCOPY         = 0x0000000A,
    TPM_PS_INFRASTRUCTURE   = 0x0000000B,
    TPM_PS_VIRTUALIZATION   = 0x0000000C,
    TPM_PS_TNC              = 0x0000000D,
    TPM_PS_MULTI_TENANT     = 0x0000000E,
```

```
    TPM_PS_TC               = 0x0000000F,
} TPM_PS_T;
typedef UINT32 TPM_PS;


/* HANDLES */
typedef UINT32 TPM_HANDLE;

/* Handle Types */
typedef enum {
    TPM_HT_PCR              = 0x00,
    TPM_HT_NV_INDEX         = 0x01,
    TPM_HT_HMAC_SESSION     = 0x02,
    TPM_HT_LOADED_SESSION   = 0x02,
    TPM_HT_POLICY_SESSION   = 0x03,
    TPM_HT_ACTIVE_SESSION   = 0x03,
    TPM_HT_PERMANENT        = 0x40,
    TPM_HT_TRANSIENT        = 0x80,
    TPM_HT_PERSISTENT       = 0x81,
} TPM_HT_T;
typedef UINT8 TPM_HT;

/* Permanent Handles */
typedef enum {
    TPM_RH_FIRST        = 0x40000000,
    TPM_RH_SRK          = TPM_RH_FIRST,
    TPM_RH_OWNER        = 0x40000001,
    TPM_RH_REVOKE       = 0x40000002,
    TPM_RH_TRANSPORT    = 0x40000003,
    TPM_RH_OPERATOR     = 0x40000004,
    TPM_RH_ADMIN        = 0x40000005,
    TPM_RH_EK           = 0x40000006,
    TPM_RH_NULL         = 0x40000007,
    TPM_RH_UNASSIGNED   = 0x40000008,
    TPM_RS_PW           = 0x40000009,
    TPM_RH_LOCKOUT      = 0x4000000A,
    TPM_RH_ENDORSEMENT  = 0x4000000B,
    TPM_RH_PLATFORM     = 0x4000000C,
    TPM_RH_PLATFORM_NV  = 0x4000000D,
    TPM_RH_AUTH_00      = 0x40000010,
    TPM_RH_AUTH_FF      = 0x4000010F,
    TPM_RH_LAST         = TPM_RH_AUTH_FF,
} TPM_RH_T;
typedef UINT32 TPM_RH;

/* Handle Value Constants */
/* Using defines, not "enum TPM_HC_T" to avoid pedantic error:
 * "ISO C restricts enumerator values to range of 'int'"
 */
#define HR_HANDLE_MASK       0x00FFFFFFUL
#define HR_RANGE_MASK        0xFF000000UL
#define HR_SHIFT             24
#define HR_PCR               ((UINT32)TPM_HT_PCR << HR_SHIFT)
#define HR_HMAC_SESSION      ((UINT32)TPM_HT_HMAC_SESSION << HR_SHIFT)
```

```
#define HR_POLICY_SESSION    ((UINT32)TPM_HT_POLICY_SESSION << HR_SHIFT)
#define HR_TRANSIENT         ((UINT32)TPM_HT_TRANSIENT << HR_SHIFT)
#define HR_PERSISTENT        ((UINT32)TPM_HT_PERSISTENT << HR_SHIFT)
#define HR_NV_INDEX          ((UINT32)TPM_HT_NV_INDEX << HR_SHIFT)
#define HR_PERMANENT         ((UINT32)TPM_HT_PERMANENT << HR_SHIFT)
#define PCR_FIRST            (HR_PCR + 0)
#define PCR_LAST             (PCR_FIRST + IMPLEMENTATION_PCR-1)
#define HMAC_SESSION_FIRST   (HR_HMAC_SESSION + 0)
#define HMAC_SESSION_LAST    (HMAC_SESSION_FIRST+MAX_ACTIVE_SESSIONS-1)
#define LOADED_SESSION_FIRST HMAC_SESSION_FIRST
#define LOADED_SESSION_LAST  HMAC_SESSION_LAST
#define POLICY_SESSION_FIRST (HR_POLICY_SESSION + 0)
#define POLICY_SESSION_LAST  (POLICY_SESSION_FIRST+MAX_ACTIVE_SESSIONS-1)
#define TRANSIENT_FIRST      (HR_TRANSIENT + 0)
#define ACTIVE_SESSION_FIRST POLICY_SESSION_FIRST
#define ACTIVE_SESSION_LAST  POLICY_SESSION_LAST
#define TRANSIENT_LAST       (TRANSIENT_FIRST+MAX_LOADED_OBJECTS-1)
#define PERSISTENT_FIRST     (HR_PERSISTENT + 0)
#define PERSISTENT_LAST      (PERSISTENT_FIRST + 0x00FFFFFFUL)
#define PLATFORM_PERSISTENT  (PERSISTENT_FIRST + 0x00800000UL)
#define NV_INDEX_FIRST       (HR_NV_INDEX + 0)
#define NV_INDEX_LAST        (NV_INDEX_FIRST + 0x00FFFFFFUL)
#define PERMANENT_FIRST      TPM_RH_FIRST
#define PERMANENT_LAST       TPM_RH_LAST
typedef UINT32 TPM_HC;


/* Attributes */
typedef UINT32 TPMA_ALGORITHM;
enum TPMA_ALGORITHM_mask {
    TPMA_ALGORITHM_asymmetric = 0x00000001,
    TPMA_ALGORITHM_symmetric  = 0x00000002,
    TPMA_ALGORITHM_hash       = 0x00000004,
    TPMA_ALGORITHM_object     = 0x00000008,
    TPMA_ALGORITHM_signing    = 0x00000100,
    TPMA_ALGORITHM_encrypting = 0x00000200,
    TPMA_ALGORITHM_method     = 0x00000400,
};

typedef UINT32 TPMA_OBJECT;
enum TPMA_OBJECT_mask {
    TPMA_OBJECT_fixedTPM             = 0x00000002,
    TPMA_OBJECT_stClear              = 0x00000004,
    TPMA_OBJECT_fixedParent          = 0x00000010,
    TPMA_OBJECT_sensitiveDataOrigin = 0x00000020,
    TPMA_OBJECT_userWithAuth         = 0x00000040,
    TPMA_OBJECT_adminWithPolicy      = 0x00000080,
    TPMA_OBJECT_noDA                 = 0x00000400,
    TPMA_OBJECT_encryptedDuplication= 0x00000800,
    TPMA_OBJECT_restricted           = 0x00010000,
    TPMA_OBJECT_decrypt              = 0x00020000,
    TPMA_OBJECT_sign                 = 0x00040000,
#ifndef WOLFTPM_V185
    /* Deprecated alias. Earlier versions of this header labeled bit 9
```

```
     * as derivedDataOrigin, which does not appear in the TCG spec.
     * In v1.85 builds bit 9 is exposed only as svnLimited. */
    TPMA_OBJECT_derivedDataOrigin   = 0x00000200,
#endif
#ifdef WOLFTPM_V185
    /* Part 2 v1.85 Sec.8.3.2 Table 36 bits 8 and 9: firmwareLimited /
     * svnLimited mark keys whose lifetime is bound to the firmware
     * version / SVN of the TPM that produced them. */
    TPMA_OBJECT_firmwareLimited     = 0x00000100,
    TPMA_OBJECT_svnLimited          = 0x00000200,
    /* Part 2 v1.85 Sec.8.3.3 (bit 19): x509sign restricts the digests this
     * key can sign so the signature is suitable for use as an X.509
     * Certificate signature. Part 3 Sec.20.6.1 / Sec.20.7.1 mandate
     * TPM_RC_ATTRIBUTES if SET on a key passed to TPM2_SignSequenceComplete
     * or TPM2_SignDigest. */
    TPMA_OBJECT_x509sign            = 0x00080000,
#endif
};

typedef BYTE TPMA_SESSION;
enum TPMA_SESSION_mask {
    TPMA_SESSION_continueSession    = 0x01,
    TPMA_SESSION_auditExclusive     = 0x02,
    TPMA_SESSION_auditReset         = 0x04,
    TPMA_SESSION_decrypt            = 0x20,
    TPMA_SESSION_encrypt            = 0x40,
    TPMA_SESSION_audit              = 0x80,
};

typedef BYTE TPMA_LOCALITY;
enum TPMA_LOCALITY_mask {
    TPM_LOC_ZERO = 0x01,
    TPM_LOC_ONE = 0x02,
    TPM_LOC_TWO = 0x04,
    TPM_LOC_THREE = 0x08,
    TPM_LOC_FOUR = 0x10,
};

typedef UINT32 TPMA_PERMANENT;
enum TPMA_PERMANENT_mask {
    TPMA_PERMANENT_ownerAuthSet         = 0x00000001,
    TPMA_PERMANENT_endorsementAuthSet   = 0x00000002,
    TPMA_PERMANENT_lockoutAuthSet       = 0x00000004,
    TPMA_PERMANENT_disableClear         = 0x00000100,
    TPMA_PERMANENT_inLockout            = 0x00000200,
    TPMA_PERMANENT_tpmGeneratedEPS      = 0x00000400,
};

typedef UINT32 TPMA_STARTUP_CLEAR;
/* Using defines, not "enum TPMA_STARTUP_CLEAR_mask" to avoid pedantic error:
 * "ISO C restricts enumerator values to range of 'int'"
 */
#define TPMA_STARTUP_CLEAR_phEnable   0x00000001UL
#define TPMA_STARTUP_CLEAR_shEnable   0x00000002UL
```

```
#define TPMA_STARTUP_CLEAR_ehEnable   0x00000004UL
#define TPMA_STARTUP_CLEAR_phEnableNV 0x00000008UL
#define TPMA_STARTUP_CLEAR_orderly    0x80000000UL

typedef UINT32 TPMA_MEMORY;
enum TPMA_MEMORY_mask {
    TPMA_MEMORY_sharedRAM           = 0x00000001,
    TPMA_MEMORY_sharedNV            = 0x00000002,
    TPMA_MEMORY_objectCopiedToRam   = 0x00000004,
};

typedef UINT32 TPMA_CC;
enum TPMA_CC_mask {
    TPMA_CC_commandIndex = 0x0000FFFF,
    TPMA_CC_nv           = 0x00400000,
    TPMA_CC_extensive    = 0x00800000,
    TPMA_CC_flushed      = 0x01000000,
    TPMA_CC_cHandles     = 0x0E000000,
    TPMA_CC_rHandle      = 0x10000000,
    TPMA_CC_V            = 0x20000000,
};

/* TPM 2.0 Library Spec Part 2: bitfield returned from
 * TPM2_GetCapability(TPM_CAP_TPM_PROPERTIES, TPM_PT_MODES)
 * indicating FIPS compliance level. Bit 0 (FIPS_140_2) is defined
 * since v1.38; bit 1 (FIPS_140_3) was added in v1.83. Bits 2..31
 * are reserved. Decoded unconditionally so any host build can
 * interpret either flag from any TPM. */
typedef UINT32 TPMA_MODES;
enum TPMA_MODES_mask {
    TPMA_MODES_FIPS_140_2 = 0x00000001,
    TPMA_MODES_FIPS_140_3 = 0x00000002,
};

#ifdef WOLFTPM_PQC
/* v185 rc4 Part 2 Sec.8.13 Table 46 - bitfield returned from
 * TPM2_GetCapability(TPM_CAP_TPM_PROPERTIES, TPM_PT_ML_PARAMETER_SETS)
 * indicating which ML-KEM/ML-DSA parameter sets the TPM supports. */
typedef UINT32 TPMA_ML_PARAMETER_SET;
enum TPMA_ML_PARAMETER_SET_mask {
    TPMA_ML_PARAMETER_SET_mlKem_512   = 0x00000001,
    TPMA_ML_PARAMETER_SET_mlKem_768   = 0x00000002,
    TPMA_ML_PARAMETER_SET_mlKem_1024  = 0x00000004,
    TPMA_ML_PARAMETER_SET_mlDsa_44    = 0x00000008,
    TPMA_ML_PARAMETER_SET_mlDsa_65    = 0x00000010,
    TPMA_ML_PARAMETER_SET_mlDsa_87    = 0x00000020,
    TPMA_ML_PARAMETER_SET_extMu       = 0x00000040,
};
#endif


/* Interface Types */

typedef BYTE TPMI_YES_NO;
```

```
typedef TPM_HANDLE TPMI_DH_OBJECT;
typedef TPM_HANDLE TPMI_DH_PARENT;
typedef TPM_HANDLE TPMI_DH_PERSISTENT;
typedef TPM_HANDLE TPMI_DH_ENTITY;
typedef TPM_HANDLE TPMI_DH_PCR;
typedef TPM_HANDLE TPMI_SH_AUTH_SESSION;
typedef TPM_HANDLE TPMI_SH_HMAC;
typedef TPM_HANDLE TPMI_SH_POLICY;
typedef TPM_HANDLE TPMI_DH_CONTEXT;
typedef TPM_HANDLE TPMI_RH_HIERARCHY;
typedef TPM_HANDLE TPMI_RH_ENABLES;
typedef TPM_HANDLE TPMI_RH_HIERARCHY_AUTH;
typedef TPM_HANDLE TPMI_RH_PLATFORM;
typedef TPM_HANDLE TPMI_RH_OWNER;
typedef TPM_HANDLE TPMI_RH_ENDORSEMENT;
typedef TPM_HANDLE TPMI_RH_PROVISION;
typedef TPM_HANDLE TPMI_RH_CLEAR;
typedef TPM_HANDLE TPMI_RH_NV_AUTH;
typedef TPM_HANDLE TPMI_RH_LOCKOUT;
typedef TPM_HANDLE TPMI_RH_NV_INDEX;
#ifdef WOLFTPM_SPDM
typedef TPM_HANDLE TPMI_DH_AC;  /* Authenticated Controller handle */
#endif

typedef TPM_ALG_ID TPMI_ALG_HASH;
typedef TPM_ALG_ID TPMI_ALG_ASYM;
typedef TPM_ALG_ID TPMI_ALG_SYM;
typedef TPM_ALG_ID TPMI_ALG_SYM_OBJECT;
typedef TPM_ALG_ID TPMI_ALG_SYM_MODE;
typedef TPM_ALG_ID TPMI_ALG_KDF;
typedef TPM_ALG_ID TPMI_ALG_SIG_SCHEME;
typedef TPM_ALG_ID TPMI_ECC_KEY_EXCHANGE;

typedef TPM_ST TPMI_ST_COMMAND_TAG;


/* Structures */

typedef struct TPMS_ALGORITHM_DESCRIPTION {
    TPM_ALG_ID alg;
    TPMA_ALGORITHM attributes;
} TPMS_ALGORITHM_DESCRIPTION;


typedef union TPMU_HA {
    BYTE sha512[TPM_SHA512_DIGEST_SIZE];
    BYTE sha384[TPM_SHA384_DIGEST_SIZE];
    BYTE sha256[TPM_SHA256_DIGEST_SIZE];
    BYTE sha224[TPM_SHA224_DIGEST_SIZE];
    BYTE sha[TPM_SHA_DIGEST_SIZE];
    BYTE md5[TPM_MD5_DIGEST_SIZE];
    BYTE H[TPM_MAX_DIGEST_SIZE];
} TPMU_HA;
```

```
typedef struct TPMT_HA {
    TPMI_ALG_HASH hashAlg;
    TPMU_HA digest;
} TPMT_HA;

typedef struct TPM2B_DIGEST {
    UINT16 size;
    BYTE buffer[sizeof(TPMU_HA)];
} TPM2B_DIGEST;

typedef struct TPM2B_DATA {
    UINT16 size;
    BYTE buffer[sizeof(TPMT_HA)];
} TPM2B_DATA;

typedef TPM2B_DIGEST TPM2B_NONCE;
typedef TPM2B_DIGEST TPM2B_AUTH;
typedef TPM2B_DIGEST TPM2B_OPERAND;

typedef struct TPM2B_EVENT {
    UINT16 size;
    BYTE buffer[1024];
} TPM2B_EVENT;

typedef struct TPM2B_MAX_BUFFER {
    UINT16 size;
    BYTE buffer[MAX_DIGEST_BUFFER];
} TPM2B_MAX_BUFFER;

typedef struct TPM2B_MAX_NV_BUFFER {
    UINT16 size;
    BYTE buffer[MAX_NV_BUFFER_SIZE];
} TPM2B_MAX_NV_BUFFER;


typedef TPM2B_DIGEST TPM2B_TIMEOUT;

typedef struct TPM2B_IV {
    UINT16 size;
    BYTE buffer[MAX_SYM_BLOCK_SIZE];
} TPM2B_IV;

#ifdef WOLFTPM_PQC
/* Post-Quantum Cryptography (PQC) Types */
typedef struct TPM2B_SIGNATURE_CTX {
    UINT16 size;
    BYTE buffer[MAX_SIGNATURE_CTX_SIZE];
} TPM2B_SIGNATURE_CTX;

/* v185 rc4 Part 2 Sec.11.3.9 Table 221 — TPM2B_SIGNATURE_HINT carries the
 * encoded R value for EdDSA sequences; for ML-DSA and other schemes the
 * TPM requires size == 0. Used as a parameter on TPM2_VerifySequenceStart. */
typedef struct TPM2B_SIGNATURE_HINT {
    UINT16 size;
```

```
    BYTE buffer[MAX_SIGNATURE_HINT_SIZE];
} TPM2B_SIGNATURE_HINT;

typedef struct TPM2B_KEM_CIPHERTEXT {
    UINT16 size;
    BYTE buffer[MAX_KEM_CIPHERTEXT_SIZE];
} TPM2B_KEM_CIPHERTEXT;

typedef struct TPM2B_SHARED_SECRET {
    UINT16 size;
    BYTE buffer[MAX_SHARED_SECRET_SIZE];
} TPM2B_SHARED_SECRET;

/* TPM2B_PUBLIC_KEY_MLDSA - TCG v185 RC4 Table 209 */
typedef struct TPM2B_PUBLIC_KEY_MLDSA {
    UINT16 size;
    BYTE buffer[MAX_MLDSA_PUB_SIZE];
} TPM2B_PUBLIC_KEY_MLDSA;

/* TPM2B_PRIVATE_KEY_MLDSA - TCG v185 RC4 Table 210 */
typedef struct TPM2B_PRIVATE_KEY_MLDSA {
    UINT16 size;           /* shall be 32 */
    BYTE buffer[MAX_MLDSA_PRIV_SEED_SIZE];  /* 32-byte private seed Xi */
} TPM2B_PRIVATE_KEY_MLDSA;

/* TPM2B_PUBLIC_KEY_MLKEM - TCG v185 RC4 Table 205 */
typedef struct TPM2B_PUBLIC_KEY_MLKEM {
    UINT16 size;
    BYTE buffer[MAX_MLKEM_PUB_SIZE];
} TPM2B_PUBLIC_KEY_MLKEM;

/* TPM2B_PRIVATE_KEY_MLKEM - TCG v185 RC4 Table 206 */
typedef struct TPM2B_PRIVATE_KEY_MLKEM {
    UINT16 size;           /* shall be 64 */
    BYTE buffer[MAX_MLKEM_PRIV_SEED_SIZE];  /* 64-byte private seed (d||z) */
} TPM2B_PRIVATE_KEY_MLKEM;

/* TPM2B_MLDSA_SIGNATURE - ML-DSA signature (up to 4627 bytes for ML-DSA-87) */
typedef struct TPM2B_MLDSA_SIGNATURE {
    UINT16 size;
    BYTE buffer[MAX_MLDSA_SIG_SIZE];
} TPM2B_MLDSA_SIGNATURE;
#endif /* WOLFTPM_PQC */


/* Names */
typedef union TPMU_NAME {
    TPMT_HA digest;
    TPM_HANDLE handle;
} TPMU_NAME;

typedef struct TPM2B_NAME {
    UINT16 size;
    BYTE name[sizeof(TPMU_NAME)];
```

```
} TPM2B_NAME;


/* PCR */

typedef struct TPMS_PCR_SELECT {
    BYTE sizeofSelect;
    BYTE pcrSelect[PCR_SELECT_MIN];
} TPMS_PCR_SELECT;


typedef struct TPMS_PCR_SELECTION {
    TPMI_ALG_HASH hash;
    BYTE sizeofSelect;
    BYTE pcrSelect[PCR_SELECT_MAX];
} TPMS_PCR_SELECTION;


/* Tickets */

typedef struct TPMT_TK_CREATION {
    TPM_ST tag;
    TPMI_RH_HIERARCHY hierarchy;
    TPM2B_DIGEST digest;
} TPMT_TK_CREATION;

typedef struct TPMT_TK_VERIFIED {
    TPM_ST tag;
    TPMI_RH_HIERARCHY hierarchy;
#ifdef WOLFTPM_PQC
    /* v185 rc4 Part 2 Sec.10.6.5 Table 112 / Sec.10.6.4 Table 110 –
     ↪  [tag]metadata.
     * Empty on the wire for TPM_ST_VERIFIED and TPM_ST_MESSAGE_VERIFIED.
     * For TPM_ST_DIGEST_VERIFIED carries the TPM_ALG_ID (hash/XOF used).
     * For ML-DSA external-mu wolfTPM emits TPM_ALG_NULL here (hash-less
     * mu-direct path; interpretation pending Part 4 v185 publication).
     * Spec note: field formerly named `digest` was renamed to `hmac` in
     * v185 to reduce ambiguity; we retain `digest` for wolfTPM API stability
     * since the rename is editorial and does not affect wire bytes. */
    TPM_ALG_ID metaAlg;
#endif
    TPM2B_DIGEST digest;
} TPMT_TK_VERIFIED;

typedef struct TPMT_TK_AUTH {
    TPM_ST tag;
    TPMI_RH_HIERARCHY hierarchy;
    TPM2B_DIGEST digest;
} TPMT_TK_AUTH;

typedef struct TPMT_TK_HASHCHECK {
    TPM_ST tag;
    TPMI_RH_HIERARCHY hierarchy;
    TPM2B_DIGEST digest;
```

```
} TPMT_TK_HASHCHECK;

typedef struct TPMS_ALG_PROPERTY {
    TPM_ALG_ID alg;
    TPMA_ALGORITHM algProperties;
} TPMS_ALG_PROPERTY;

typedef struct TPMS_TAGGED_PROPERTY {
    TPM_PT property;
    UINT32 value;
} TPMS_TAGGED_PROPERTY;

typedef struct TPMS_TAGGED_PCR_SELECT {
    TPM_PT_PCR tag;
    BYTE sizeofSelect;
    BYTE pcrSelect[PCR_SELECT_MAX];
} TPMS_TAGGED_PCR_SELECT;

typedef struct TPMS_TAGGED_POLICY {
    TPM_HANDLE handle;
    TPMT_HA policyHash;
} TPMS_TAGGED_POLICY;

/* Settable feature configuration entry (TPM 2.0 Library v1.85).
 * Used with TPM_CAP_CONFIGS via TPM2_SetCapability / TPM2_GetCapability. */
typedef struct TPMS_TAGGED_CONFIG {
    TPM_PT_CONFIG configuration; /* TPM_PT_CONFIG_* property identifier */
    UINT8 enable;                /* if SET, configuration is enabled */
    UINT8 lock;                  /* if SET, configuration is lifetime locked */
} TPMS_TAGGED_CONFIG;


/* Lists */

typedef struct TPML_CC {
    UINT32 count;
    TPM_CC commandCodes[MAX_CAP_CC];
} TPML_CC;

typedef struct TPML_CCA {
    UINT32 count;
    TPMA_CC commandAttributes[MAX_CAP_CC];
} TPML_CCA;

typedef struct TPML_ALG {
    UINT32 count;
    TPM_ALG_ID algorithms[MAX_ALG_LIST_SIZE];
} TPML_ALG;

typedef struct TPML_HANDLE {
    UINT32 count;
    TPM_HANDLE handle[MAX_CAP_HANDLES];
} TPML_HANDLE;
```

```
typedef struct TPML_DIGEST {
    UINT32 count;
    TPM2B_DIGEST digests[8];
} TPML_DIGEST;

typedef struct TPML_DIGEST_VALUES {
    UINT32 count;
    TPMT_HA digests[HASH_COUNT];
} TPML_DIGEST_VALUES;

typedef struct TPML_PCR_SELECTION {
    UINT32 count;
    TPMS_PCR_SELECTION pcrSelections[HASH_COUNT];
} TPML_PCR_SELECTION;

typedef struct TPML_ALG_PROPERTY {
    UINT32 count;
    TPMS_ALG_PROPERTY algProperties[MAX_CAP_ALGS];
} TPML_ALG_PROPERTY;

typedef struct TPML_TAGGED_TPM_PROPERTY {
    UINT32 count;
    TPMS_TAGGED_PROPERTY tpmProperty[MAX_TPM_PROPERTIES];
} TPML_TAGGED_TPM_PROPERTY;

typedef struct TPML_TAGGED_PCR_PROPERTY {
    UINT32 count;
    TPMS_TAGGED_PCR_SELECT pcrProperty[MAX_PCR_PROPERTIES];
} TPML_TAGGED_PCR_PROPERTY;

typedef struct TPML_ECC_CURVE {
    UINT32 count;
    TPM_ECC_CURVE eccCurves[MAX_ECC_CURVES];
} TPML_ECC_CURVE;

typedef struct TPML_TAGGED_POLICY {
    UINT32 count;
    TPMS_TAGGED_POLICY policies[MAX_TAGGED_POLICIES];
} TPML_TAGGED_POLICY;

#ifndef MAX_TAGGED_CONFIGS
#define MAX_TAGGED_CONFIGS 8
#endif
/* List of settable feature configurations (TPM_CAP_CONFIGS, v1.85) */
typedef struct TPML_TAGGED_CONFIG {
    UINT32 count;
    TPMS_TAGGED_CONFIG configs[MAX_TAGGED_CONFIGS];
} TPML_TAGGED_CONFIG;

/* Authenticated Countdown Timers (ACT): Added v1.59 */
typedef enum {
    TPMA_ACT_signaled         = 0x00000001,
    TPMA_ACT_preserveSignaled = 0x00000002,
} TPMA_ACT_T;
```

```
typedef UINT32 TPMA_ACT;

typedef struct TPMS_ACT_DATA {
    TPM_HANDLE handle;
    UINT32     timeout;
    TPMA_ACT   attributes;
} TPMS_ACT_DATA;

typedef struct TPML_ACT_DATA {
    UINT32        count;
    TPMS_ACT_DATA actData[MAX_ACT_DATA];
} TPML_ACT_DATA;

#ifdef WOLFTPM_SPDM
/* SPDM session information (TPM 2.0 Library v1.84) */
typedef struct TPMS_SPDM_SESSION_INFO {
    TPM2B_NAME reqKeyName;
    TPM2B_NAME tpmKeyName;
} TPMS_SPDM_SESSION_INFO;

/* Spec-derived list capacity (TPM 2.0 Library v1.84): as many entries as fit
 * in the capability buffer. MAX_SPDM_SESS_INFO is kept as a compatibility
 * alias for the spec name MAX_SPDM_SESSION_INFO. */
#ifndef MAX_SPDM_SESSION_INFO
#define MAX_SPDM_SESSION_INFO \
    (MAX_CAP_DATA / (UINT32)sizeof(TPMS_SPDM_SESSION_INFO))
#endif
#ifndef MAX_SPDM_SESS_INFO
#define MAX_SPDM_SESS_INFO MAX_SPDM_SESSION_INFO
#endif
typedef struct TPML_SPDM_SESSION_INFO {
    UINT32 count;
    TPMS_SPDM_SESSION_INFO spdmSessionInfo[MAX_SPDM_SESS_INFO];
} TPML_SPDM_SESSION_INFO;
#endif

/* Capabilities Structures */

typedef union TPMU_CAPABILITIES {
    TPML_ALG_PROPERTY algorithms; /* TPM_CAP_ALGS */
    TPML_HANDLE handles; /* TPM_CAP_HANDLES */
    TPML_CCA command; /* TPM_CAP_COMMANDS */
    TPML_CC ppCommands; /* TPM_CAP_PP_COMMANDS */
    TPML_CC auditCommands; /* TPM_CAP_AUDIT_COMMANDS */
    TPML_PCR_SELECTION assignedPCR; /* TPM_CAP_PCRS */
    TPML_TAGGED_TPM_PROPERTY tpmProperties; /* TPM_CAP_TPM_PROPERTIES */
    TPML_TAGGED_PCR_PROPERTY pcrProperties; /* TPM_CAP_PCR_PROPERTIES */
    TPML_ECC_CURVE eccCurves; /* TPM_CAP_ECC_CURVES */
    TPML_TAGGED_POLICY authPolicies; /* TPM_CAP_AUTH_POLICIES */
    TPML_ACT_DATA actData; /* TPM_CAP_ACT - added v1.57 */
#ifdef WOLFTPM_SPDM
    TPML_SPDM_SESSION_INFO spdmSessionInfo; /* TPM_CAP_SPDM_SESSION_INFO -
 ↪  v1.84 */
#endif
```

```
    TPML_TAGGED_CONFIG tpmConfigs; /* TPM_CAP_CONFIGS - added v1.85 */
    TPM2B_MAX_BUFFER vendor;
} TPMU_CAPABILITIES;

typedef struct TPMS_CAPABILITY_DATA {
    TPM_CAP capability;
    TPMU_CAPABILITIES data;
} TPMS_CAPABILITY_DATA;

typedef struct TPMS_CLOCK_INFO {
    UINT64 clock;
    UINT32 resetCount;
    UINT32 restartCount;
    TPMI_YES_NO safe;
} TPMS_CLOCK_INFO;

typedef struct TPMS_TIME_INFO {
    UINT64 time;
    TPMS_CLOCK_INFO clockInfo;
} TPMS_TIME_INFO;

typedef struct TPMS_TIME_ATTEST_INFO {
    TPMS_TIME_INFO time;
    UINT64 firmwareVersion;
} TPMS_TIME_ATTEST_INFO;

typedef struct TPMS_CERTIFY_INFO {
    TPM2B_NAME name;
    TPM2B_NAME qualifiedName;
} TPMS_CERTIFY_INFO;

typedef struct TPMS_QUOTE_INFO {
    TPML_PCR_SELECTION pcrSelect;
    TPM2B_DIGEST pcrDigest;
} TPMS_QUOTE_INFO;

typedef struct TPMS_COMMAND_AUDIT_INFO {
    UINT64 auditCounter;
    TPM_ALG_ID digestAlg;
    TPM2B_DIGEST auditDigest;
    TPM2B_DIGEST commandDigest;
} TPMS_COMMAND_AUDIT_INFO;

typedef struct TPMS_SESSION_AUDIT_INFO {
    TPMI_YES_NO exclusiveSession;
    TPM2B_DIGEST sessionDigest;
} TPMS_SESSION_AUDIT_INFO;

typedef struct TPMS_CREATION_INFO {
    TPM2B_NAME objectName;
    TPM2B_DIGEST creationHash;
} TPMS_CREATION_INFO;

typedef struct TPMS_NV_CERTIFY_INFO {
```

```
    TPM2B_NAME indexName;
    UINT16 offset;
    TPM2B_MAX_NV_BUFFER nvContents;
} TPMS_NV_CERTIFY_INFO;

typedef struct TPMS_NV_DIGEST_CERTIFY_INFO {
    TPM2B_NAME indexName;
    TPM2B_DIGEST nvDigest;
} TPMS_NV_DIGEST_CERTIFY_INFO;


typedef TPM_ST TPMI_ST_ATTEST;
typedef union TPMU_ATTEST {
    TPMS_CERTIFY_INFO       certify;        /* TPM_ST_ATTEST_CERTIFY */
    TPMS_CREATION_INFO      creation;       /* TPM_ST_ATTEST_CREATION */
    TPMS_QUOTE_INFO         quote;          /* TPM_ST_ATTEST_QUOTE */
    TPMS_COMMAND_AUDIT_INFO commandAudit;   /* TPM_ST_ATTEST_COMMAND_AUDIT */
    TPMS_SESSION_AUDIT_INFO sessionAudit;   /* TPM_ST_ATTEST_SESSION_AUDIT */
    TPMS_TIME_ATTEST_INFO   time;           /* TPM_ST_ATTEST_TIME */
    TPMS_NV_CERTIFY_INFO    nv;             /* TPM_ST_ATTEST_NV */
    TPMS_NV_DIGEST_CERTIFY_INFO nvDigest;   /* TPM_ST_ATTEST_NV_DIGEST */
} TPMU_ATTEST;

typedef struct TPMS_ATTEST {
    TPM_GENERATED magic;
    TPMI_ST_ATTEST type;
    TPM2B_NAME qualifiedSigner;
    TPM2B_DATA extraData;
    TPMS_CLOCK_INFO clockInfo;
    UINT64 firmwareVersion;
    TPMU_ATTEST attested;
} TPMS_ATTEST;

typedef struct TPM2B_ATTEST {
    UINT16 size;
    BYTE attestationData[sizeof(TPMS_ATTEST)];
} TPM2B_ATTEST;


/* Algorithm Parameters and Structures */

/* Symmetric */
typedef TPM_KEY_BITS TPMI_AES_KEY_BITS;

typedef union TPMU_SYM_KEY_BITS {
    TPMI_AES_KEY_BITS aes;
    TPM_KEY_BITS sym;
    TPMI_ALG_HASH xorr;
} TPMU_SYM_KEY_BITS;

typedef union TPMU_SYM_MODE {
    TPMI_ALG_SYM_MODE aes;
    TPMI_ALG_SYM_MODE sym;
} TPMU_SYM_MODE;
```

```
typedef struct TPMT_SYM_DEF {
    TPMI_ALG_SYM algorithm;
    TPMU_SYM_KEY_BITS keyBits;
    TPMU_SYM_MODE mode;
    /*TPMU_SYM_DETAILS details;*/ /* not used */
} TPMT_SYM_DEF;

typedef TPMT_SYM_DEF TPMT_SYM_DEF_OBJECT;

typedef struct TPM2B_SYM_KEY {
    UINT16 size;
    BYTE buffer[MAX_SYM_KEY_BYTES];
} TPM2B_SYM_KEY;

typedef struct TPMS_SYMCIPHER_PARMS {
    TPMT_SYM_DEF_OBJECT sym;
} TPMS_SYMCIPHER_PARMS;

typedef struct TPM2B_LABEL {
    UINT16 size;
    BYTE buffer[LABEL_MAX_BUFFER];
} TPM2B_LABEL;

typedef struct TPMS_DERIVE {
    TPM2B_LABEL label;
    TPM2B_LABEL context;
} TPMS_DERIVE;

typedef struct TPM2B_DERIVE {
    UINT16 size;
    BYTE buffer[sizeof(TPMS_DERIVE)];
} TPM2B_DERIVE;

typedef union TPMU_SENSITIVE_CREATE {
    BYTE create[MAX_SYM_DATA];
    TPMS_DERIVE derive;
} TPMU_SENSITIVE_CREATE;

typedef struct TPM2B_SENSITIVE_DATA {
    UINT16 size;
    BYTE buffer[sizeof(TPMU_SENSITIVE_CREATE)];
} TPM2B_SENSITIVE_DATA;

typedef struct TPMS_SENSITIVE_CREATE {
    TPM2B_AUTH userAuth;
    TPM2B_SENSITIVE_DATA data;
} TPMS_SENSITIVE_CREATE;

typedef struct TPM2B_SENSITIVE_CREATE {
    UINT16 size;
    TPMS_SENSITIVE_CREATE sensitive;
} TPM2B_SENSITIVE_CREATE;
```

```
typedef struct TPMS_SCHEME_HASH {
    TPMI_ALG_HASH hashAlg;
} TPMS_SCHEME_HASH;

typedef struct TPMS_SCHEME_ECDAA {
    TPMI_ALG_HASH hashAlg;
    UINT16 count;
} TPMS_SCHEME_ECDAA;

typedef TPM_ALG_ID TPMI_ALG_KEYEDHASH_SCHEME;
typedef TPMS_SCHEME_HASH TPMS_SCHEME_HMAC;

typedef struct TPMS_SCHEME_XOR {
    TPMI_ALG_HASH hashAlg;
    TPMI_ALG_KDF kdf;
} TPMS_SCHEME_XOR;

typedef union TPMU_SCHEME_KEYEDHASH {
    TPMS_SCHEME_HMAC hmac;
    TPMS_SCHEME_XOR xorr;
} TPMU_SCHEME_KEYEDHASH;

typedef struct TPMT_KEYEDHASH_SCHEME {
    TPMI_ALG_KEYEDHASH_SCHEME scheme;
    TPMU_SCHEME_KEYEDHASH details;
} TPMT_KEYEDHASH_SCHEME;


/* Asymmetric */

typedef TPMS_SCHEME_HASH  TPMS_SIG_SCHEME_RSASSA;
typedef TPMS_SCHEME_HASH  TPMS_SIG_SCHEME_RSAPSS;
typedef TPMS_SCHEME_HASH  TPMS_SIG_SCHEME_ECDSA;

typedef TPMS_SCHEME_ECDAA TPMS_SIG_SCHEME_ECDAA;


typedef union TPMU_SIG_SCHEME {
    TPMS_SIG_SCHEME_RSASSA rsassa;
    TPMS_SIG_SCHEME_RSAPSS rsapss;
    TPMS_SIG_SCHEME_ECDSA  ecdsa;
    TPMS_SIG_SCHEME_ECDAA  ecdaa;
    TPMS_SCHEME_HMAC       hmac;
    TPMS_SCHEME_HASH       any;
} TPMU_SIG_SCHEME;

typedef struct TPMT_SIG_SCHEME {
    TPMI_ALG_SIG_SCHEME scheme;
    TPMU_SIG_SCHEME details;
} TPMT_SIG_SCHEME;


/* Encryption / Key Exchange Schemes */
typedef TPMS_SCHEME_HASH TPMS_ENC_SCHEME_OAEP;
```

```
typedef TPMS_SCHEME_HASH TPMS_KEY_SCHEME_ECDH;
typedef TPMS_SCHEME_HASH TPMS_KEY_SCHEME_ECMQV;

/* Key Derivation Schemes */
typedef TPMS_SCHEME_HASH TPMS_SCHEME_MGF1;
typedef TPMS_SCHEME_HASH TPMS_SCHEME_KDF1_SP800_56A;
typedef TPMS_SCHEME_HASH TPMS_SCHEME_KDF2;
typedef TPMS_SCHEME_HASH TPMS_SCHEME_KDF1_SP800_108;

typedef union TPMU_KDF_SCHEME {
    TPMS_SCHEME_MGF1            mgf1;
    TPMS_SCHEME_KDF1_SP800_56A  kdf1_sp800_56a;
    TPMS_SCHEME_KDF2            kdf2;
    TPMS_SCHEME_KDF1_SP800_108  kdf1_sp800_108;
    TPMS_SCHEME_HASH            any;
} TPMU_KDF_SCHEME;

typedef struct TPMT_KDF_SCHEME {
    TPMI_ALG_KDF scheme;
    TPMU_KDF_SCHEME details;
} TPMT_KDF_SCHEME;

typedef TPM_ALG_ID TPMI_ALG_ASYM_SCHEME;
typedef union TPMU_ASYM_SCHEME {
    TPMS_KEY_SCHEME_ECDH    ecdh;
    TPMS_SIG_SCHEME_RSASSA  rsassa;
    TPMS_SIG_SCHEME_RSAPSS  rsapss;
    TPMS_SIG_SCHEME_ECDSA   ecdsa;
    TPMS_ENC_SCHEME_OAEP    oaep;
    TPMS_SCHEME_HASH        anySig;
} TPMU_ASYM_SCHEME;

typedef struct TPMT_ASYM_SCHEME {
    TPMI_ALG_ASYM_SCHEME scheme;
    TPMU_ASYM_SCHEME details;
} TPMT_ASYM_SCHEME;

/* RSA */
typedef TPM_ALG_ID TPMI_ALG_RSA_SCHEME;
typedef struct TPMT_RSA_SCHEME {
    TPMI_ALG_RSA_SCHEME scheme;
    TPMU_ASYM_SCHEME details;
} TPMT_RSA_SCHEME;

typedef TPM_ALG_ID TPMI_ALG_RSA_DECRYPT;
typedef struct TPMT_RSA_DECRYPT {
    TPMI_ALG_RSA_DECRYPT scheme;
    TPMU_ASYM_SCHEME details;
} TPMT_RSA_DECRYPT;

typedef struct TPM2B_PUBLIC_KEY_RSA {
    UINT16 size;
    BYTE buffer[MAX_RSA_KEY_BYTES];
} TPM2B_PUBLIC_KEY_RSA;
```

```
typedef TPM_KEY_BITS TPMI_RSA_KEY_BITS;
typedef struct TPM2B_PRIVATE_KEY_RSA {
    UINT16 size;
    BYTE buffer[MAX_RSA_KEY_BYTES/2];
} TPM2B_PRIVATE_KEY_RSA;


/* ECC */
typedef struct TPM2B_ECC_PARAMETER {
    UINT16 size;
    BYTE buffer[MAX_ECC_KEY_BYTES];
} TPM2B_ECC_PARAMETER;

typedef struct TPMS_ECC_POINT {
    TPM2B_ECC_PARAMETER x;
    TPM2B_ECC_PARAMETER y;
} TPMS_ECC_POINT;

typedef struct TPM2B_ECC_POINT {
    UINT16 size;
    TPMS_ECC_POINT point;
} TPM2B_ECC_POINT;


typedef TPM_ALG_ID TPMI_ALG_ECC_SCHEME;
typedef TPM_ECC_CURVE TPMI_ECC_CURVE;
typedef TPMT_SIG_SCHEME TPMT_ECC_SCHEME;

typedef struct TPMS_ALGORITHM_DETAIL_ECC {
    TPM_ECC_CURVE curveID;
    UINT16 keySize;
    TPMT_KDF_SCHEME kdf;
    TPMT_ECC_SCHEME sign;
    TPM2B_ECC_PARAMETER p;
    TPM2B_ECC_PARAMETER a;
    TPM2B_ECC_PARAMETER b;
    TPM2B_ECC_PARAMETER gX;
    TPM2B_ECC_PARAMETER gY;
    TPM2B_ECC_PARAMETER n;
    TPM2B_ECC_PARAMETER h;
} TPMS_ALGORITHM_DETAIL_ECC;


/* Signatures */

typedef struct TPMS_SIGNATURE_RSA {
    TPMI_ALG_HASH hash;
    TPM2B_PUBLIC_KEY_RSA sig;
} TPMS_SIGNATURE_RSA;

typedef TPMS_SIGNATURE_RSA TPMS_SIGNATURE_RSASSA;
typedef TPMS_SIGNATURE_RSA TPMS_SIGNATURE_RSAPSS;
```

```
typedef struct TPMS_SIGNATURE_ECC {
    TPMI_ALG_HASH hash;
    TPM2B_ECC_PARAMETER signatureR;
    TPM2B_ECC_PARAMETER signatureS;
} TPMS_SIGNATURE_ECC;

typedef TPMS_SIGNATURE_ECC TPMS_SIGNATURE_ECDSA;
typedef TPMS_SIGNATURE_ECC TPMS_SIGNATURE_ECDAA;

#ifdef WOLFTPM_PQC
/* v185 rc4 Part 2 Sec.11.2.7.2 Table 208 – TPMS_SIGNATURE_HASH_MLDSA carries
 * the pre-hash algorithm together with the signature bytes. Used for
 * TPM_ALG_HASH_MLDSA signatures only. */
typedef struct TPMS_SIGNATURE_HASH_MLDSA {
    TPMI_ALG_HASH hash;
    TPM2B_MLDSA_SIGNATURE signature;
} TPMS_SIGNATURE_HASH_MLDSA;
#endif /* WOLFTPM_PQC */

typedef union TPMU_SIGNATURE {
    TPMS_SIGNATURE_ECDSA ecdsa;
    TPMS_SIGNATURE_ECDAA ecdaa;
    TPMS_SIGNATURE_RSASSA rsassa;
    TPMS_SIGNATURE_RSAPSS rsapss;
    TPMT_HA hmac;
    TPMS_SCHEME_HASH any;
#ifdef WOLFTPM_PQC
    /* v185 rc4 Part 2 Sec.11.3.5 Table 217. Note: mldsa arm is TPM2B (bare
     * signature bytes with no hash field) because Pure ML-DSA does not
     * select a hash; hash_mldsa arm is TPMS (hash + signature) for the
     * pre-hashed variant. See Table 217 note. */
    TPM2B_MLDSA_SIGNATURE mldsa;
    TPMS_SIGNATURE_HASH_MLDSA hash_mldsa;
#endif /* WOLFTPM_PQC */
} TPMU_SIGNATURE;

typedef struct TPMT_SIGNATURE {
    TPMI_ALG_SIG_SCHEME sigAlg;
    TPMU_SIGNATURE signature;
} TPMT_SIGNATURE;


/* Key/Secret Exchange */

typedef union TPMU_ENCRYPTED_SECRET {
    BYTE ecc[sizeof(TPMS_ECC_POINT)];     /* TPM_ALG_ECC */
    BYTE rsa[MAX_RSA_KEY_BYTES];          /* TPM_ALG_RSA */
    BYTE symmetric[sizeof(TPM2B_DIGEST)]; /* TPM_ALG_SYMCIPHER */
    BYTE keyedHash[sizeof(TPM2B_DIGEST)]; /* TPM_ALG_KEYEDHASH */
#ifdef WOLFTPM_PQC
    BYTE mlkem[MAX_MLKEM_CT_SIZE];        /* TPM_ALG_MLKEM (v1.85 T222) */
#endif
} TPMU_ENCRYPTED_SECRET;
```

```
typedef struct TPM2B_ENCRYPTED_SECRET {
    UINT16 size;
    BYTE secret[sizeof(TPMU_ENCRYPTED_SECRET)];
} TPM2B_ENCRYPTED_SECRET;


/* Key/Object Complex */

typedef TPM_ALG_ID TPMI_ALG_PUBLIC;

typedef union TPMU_PUBLIC_ID {
    TPM2B_DIGEST keyedHash;   /* TPM_ALG_KEYEDHASH */
    TPM2B_DIGEST sym;         /* TPM_ALG_SYMCIPHER */
    TPM2B_PUBLIC_KEY_RSA rsa; /* TPM_ALG_RSA */
    TPMS_ECC_POINT ecc;       /* TPM_ALG_ECC */
    TPMS_DERIVE derive;
#ifdef WOLFTPM_PQC
    TPM2B_PUBLIC_KEY_MLDSA mldsa;  /* TPM_ALG_MLDSA or TPM_ALG_HASH_MLDSA */
    TPM2B_PUBLIC_KEY_MLKEM mlkem;  /* TPM_ALG_MLKEM */
#endif
} TPMU_PUBLIC_ID;

typedef struct TPMS_KEYEDHASH_PARMS {
    TPMT_KEYEDHASH_SCHEME scheme;
} TPMS_KEYEDHASH_PARMS;


typedef struct TPMS_ASYM_PARMS {
    TPMT_SYM_DEF_OBJECT symmetric;
    TPMT_ASYM_SCHEME scheme;
} TPMS_ASYM_PARMS;

typedef struct TPMS_RSA_PARMS {
    TPMT_SYM_DEF_OBJECT symmetric;
    TPMT_RSA_SCHEME scheme;
    TPMI_RSA_KEY_BITS keyBits;
    UINT32 exponent;
} TPMS_RSA_PARMS;

typedef struct TPMS_ECC_PARMS {
    TPMT_SYM_DEF_OBJECT symmetric;
    TPMT_ECC_SCHEME scheme;
    TPMI_ECC_CURVE curveID;
    TPMT_KDF_SCHEME kdf;
} TPMS_ECC_PARMS;

#ifdef WOLFTPM_PQC
/* TPMS_MLDSA_PARMS - TCG v185 RC4 Table 229 */
typedef struct TPMS_MLDSA_PARMS {
    TPMI_MLDSA_PARAMETER_SET parameterSet;  /* ML-DSA parameter set ID */
    TPMI_YES_NO allowExternalMu;            /* Allow
 ↪  TPM2_SignDigest/VerifyDigestSignature */
} TPMS_MLDSA_PARMS;
```

```
/* TPMS_HASH_MLDSA_PARMS - TCG v185 RC4 Table 230 (Pre-Hash ML-DSA) */
typedef struct TPMS_HASH_MLDSA_PARMS {
    TPMI_MLDSA_PARAMETER_SET parameterSet;  /* ML-DSA parameter set ID */
    TPMI_ALG_HASH hashAlg;                  /* Pre-hash function PH */
} TPMS_HASH_MLDSA_PARMS;

/* TPMS_MLKEM_PARMS - TCG v185 RC4 Table 231 */
typedef struct TPMS_MLKEM_PARMS {
    TPMT_SYM_DEF_OBJECT symmetric;          /* For restricted decryption key */
    TPMI_MLKEM_PARAMETER_SET parameterSet;  /* ML-KEM parameter set */
} TPMS_MLKEM_PARMS;
#endif /* WOLFTPM_PQC */

typedef union TPMU_PUBLIC_PARMS {
    TPMS_KEYEDHASH_PARMS keyedHashDetail;
    TPMS_SYMCIPHER_PARMS symDetail;
    TPMS_RSA_PARMS rsaDetail;
    TPMS_ECC_PARMS eccDetail;
    TPMS_ASYM_PARMS asymDetail;
#ifdef WOLFTPM_PQC
    TPMS_MLDSA_PARMS mldsaDetail;           /* TPM_ALG_MLDSA - sign only */
    TPMS_HASH_MLDSA_PARMS hash_mldsaDetail; /* TPM_ALG_HASH_MLDSA - sign only
 ↪  */
    TPMS_MLKEM_PARMS mlkemDetail;           /* TPM_ALG_MLKEM - decrypt only */
#endif
} TPMU_PUBLIC_PARMS;

typedef struct TPMT_PUBLIC_PARMS {
    TPMI_ALG_PUBLIC type;
    TPMU_PUBLIC_PARMS parameters;
} TPMT_PUBLIC_PARMS;


typedef struct TPMT_PUBLIC {
    TPMI_ALG_PUBLIC type;
    TPMI_ALG_HASH nameAlg;
    TPMA_OBJECT objectAttributes;
    TPM2B_DIGEST authPolicy;
    TPMU_PUBLIC_PARMS parameters;
    TPMU_PUBLIC_ID unique;
} TPMT_PUBLIC;

typedef struct TPM2B_PUBLIC {
    UINT16 size;
    TPMT_PUBLIC publicArea;
} TPM2B_PUBLIC;

typedef struct TPM2B_TEMPLATE {
    UINT16 size;
    BYTE buffer[sizeof(TPMT_PUBLIC)];
} TPM2B_TEMPLATE;


/* Private Structures */
```

```
typedef struct TPM2B_PRIVATE_VENDOR_SPECIFIC {
    UINT16 size;
    BYTE buffer[PRIVATE_VENDOR_SPECIFIC_BYTES];
} TPM2B_PRIVATE_VENDOR_SPECIFIC;

typedef union TPMU_SENSITIVE_COMPOSITE {
    TPM2B_PRIVATE_KEY_RSA rsa;  /* TPM_ALG_RSA */
    TPM2B_ECC_PARAMETER ecc;    /* TPM_ALG_ECC */
    TPM2B_SENSITIVE_DATA bits;  /* TPM_ALG_KEYEDHASH */
    TPM2B_SYM_KEY sym;          /* TPM_ALG_SYMCIPHER */
    TPM2B_PRIVATE_VENDOR_SPECIFIC any;
#ifdef WOLFTPM_PQC
    TPM2B_PRIVATE_KEY_MLDSA mldsa;  /* TPM_ALG_MLDSA/HASH_MLDSA - seed Xi */
    TPM2B_PRIVATE_KEY_MLKEM mlkem;  /* TPM_ALG_MLKEM - seed (d||z) */
#endif
} TPMU_SENSITIVE_COMPOSITE;


typedef struct TPMT_SENSITIVE {
    TPMI_ALG_PUBLIC sensitiveType;
    TPM2B_AUTH authValue;
    TPM2B_DIGEST seedValue;
    TPMU_SENSITIVE_COMPOSITE sensitive;
} TPMT_SENSITIVE;

typedef struct TPM2B_SENSITIVE {
    UINT16 size;
    TPMT_SENSITIVE sensitiveArea;
} TPM2B_SENSITIVE;


typedef struct TPMT_PRIVATE {
    TPM2B_DIGEST integrityOuter;
    TPM2B_DIGEST integrityInner;
    TPM2B_SENSITIVE sensitive;
} TPMT_PRIVATE;

typedef struct TPM2B_PRIVATE {
    UINT16 size;
    BYTE buffer[sizeof(TPMT_PRIVATE)];
} TPM2B_PRIVATE;


/* Identity Object */

typedef struct TPMS_ID_OBJECT {
    TPM2B_DIGEST integrityHMAC;
    TPM2B_DIGEST encIdentity;
} TPMS_ID_OBJECT;

typedef struct TPM2B_ID_OBJECT {
    UINT16 size;
    BYTE buffer[sizeof(TPMS_ID_OBJECT)];
```

```
} TPM2B_ID_OBJECT;


/* NV Storage Structures */

typedef UINT32 TPM_NV_INDEX;
/* Using defines, not "enum TPM_NV_INDEX_mask" to avoid pedantic error:
 * "ISO C restricts enumerator values to range of 'int'"
 */
#define TPM_NV_INDEX_index 0x00FFFFFFUL
#define TPM_NV_INDEX_RH_NV 0xFF000000UL


typedef enum TPM_NT {
    TPM_NT_ORDINARY = 0x0,
    TPM_NT_COUNTER  = 0x1,
    TPM_NT_BITS     = 0x2,
    TPM_NT_EXTEND   = 0x4,
    TPM_NT_PIN_FAIL = 0x8,
    TPM_NT_PIN_PASS = 0x9,
} TPM_NT;


typedef struct TPMS_NV_PIN_COUNTER_PARAMETERS {
    UINT32 pinCount;
    UINT32 pinLimit;
} TPMS_NV_PIN_COUNTER_PARAMETERS;

typedef UINT32 TPMA_NV;
/* Using defines, not "enum TPMA_NV_mask" to avoid pedantic error:
 * "ISO C restricts enumerator values to range of 'int'"
 */
#define TPMA_NV_PPWRITE        0x00000001UL
#define TPMA_NV_OWNERWRITE     0x00000002UL
#define TPMA_NV_AUTHWRITE      0x00000004UL
#define TPMA_NV_POLICYWRITE    0x00000008UL
#define TPMA_NV_TPM_NT         0x000000F0UL /* index type see TPM_NT_ */
#define TPMA_NV_POLICY_DELETE  0x00000400UL
#define TPMA_NV_WRITELOCKED    0x00000800UL
#define TPMA_NV_WRITEALL       0x00001000UL
#define TPMA_NV_WRITEDEFINE    0x00002000UL
#define TPMA_NV_WRITE_STCLEAR  0x00004000UL
#define TPMA_NV_GLOBALLOCK     0x00008000UL
#define TPMA_NV_PPREAD         0x00010000UL
#define TPMA_NV_OWNERREAD      0x00020000UL
#define TPMA_NV_AUTHREAD       0x00040000UL
#define TPMA_NV_POLICYREAD     0x00080000UL
#define TPMA_NV_NO_DA          0x02000000UL
#define TPMA_NV_ORDERLY        0x04000000UL
#define TPMA_NV_CLEAR_STCLEAR  0x08000000UL
#define TPMA_NV_READLOCKED     0x10000000UL
#define TPMA_NV_WRITTEN        0x20000000UL
#define TPMA_NV_PLATFORMCREATE 0x40000000UL
#define TPMA_NV_READ_STCLEAR   0x80000000UL
```

```
typedef struct TPMS_NV_PUBLIC {
    TPMI_RH_NV_INDEX nvIndex;
    TPMI_ALG_HASH nameAlg;
    TPMA_NV attributes;
    TPM2B_DIGEST authPolicy;
    UINT16 dataSize;
} TPMS_NV_PUBLIC;

typedef struct TPM2B_NV_PUBLIC {
    UINT16 size;
    TPMS_NV_PUBLIC nvPublic;
} TPM2B_NV_PUBLIC;


/* Context Data */

typedef struct TPM2B_CONTEXT_SENSITIVE {
    UINT16 size;
    BYTE buffer[MAX_CONTEXT_SIZE];
} TPM2B_CONTEXT_SENSITIVE;

typedef struct TPMS_CONTEXT_DATA {
    TPM2B_DIGEST integrity;
    TPM2B_CONTEXT_SENSITIVE encrypted;
} TPMS_CONTEXT_DATA;

typedef struct TPM2B_CONTEXT_DATA {
    UINT16 size;
    BYTE buffer[sizeof(TPMS_CONTEXT_DATA)];
} TPM2B_CONTEXT_DATA;

typedef struct TPMS_CONTEXT {
    UINT64 sequence;
    TPMI_DH_CONTEXT savedHandle;
    TPMI_RH_HIERARCHY hierarchy;
    TPM2B_CONTEXT_DATA contextBlob;
} TPMS_CONTEXT;


typedef struct TPMS_CREATION_DATA {
    TPML_PCR_SELECTION pcrSelect;
    TPM2B_DIGEST pcrDigest;
    TPMA_LOCALITY locality;
    TPM_ALG_ID parentNameAlg;
    TPM2B_NAME parentName;
    TPM2B_NAME parentQualifiedName;
    TPM2B_DATA outsideInfo;
} TPMS_CREATION_DATA;

typedef struct TPM2B_CREATION_DATA {
    UINT16 size;
    TPMS_CREATION_DATA creationData;
} TPM2B_CREATION_DATA;
```

```
/* Authorization Structures */
typedef struct TPMS_AUTH_COMMAND {
    TPMI_SH_AUTH_SESSION sessionHandle;
    TPM2B_NONCE nonce; /* nonceCaller */
    TPMA_SESSION sessionAttributes;
    TPM2B_AUTH hmac;
} TPMS_AUTH_COMMAND;

typedef struct TPMS_AUTH_RESPONSE {
    TPM2B_NONCE nonce;
    TPMA_SESSION sessionAttributes;
    TPM2B_AUTH hmac;
} TPMS_AUTH_RESPONSE;

/* Implementation specific authorization session information */
typedef struct TPM2_AUTH_SESSION {
    /* this section is used for TPMS_AUTH_COMMAND */
    TPMI_SH_AUTH_SESSION sessionHandle;
    TPM2B_NONCE nonceCaller;
    TPMA_SESSION sessionAttributes;
    TPM2B_AUTH hmac;

    /* additional auth data required for implementation */
    TPM2B_NONCE nonceTPM;
    TPMT_SYM_DEF symmetric;
    TPMI_ALG_HASH authHash;
    TPM2B_NAME name;
    TPM2B_AUTH auth;
    TPM2B_AUTH* bind;
    TPM2B_NAME bindName; /* bind entity Name; see TPM2_ParamEncBindKey */

    unsigned int policyAuth : 1; /* if policy auth should be used */
    unsigned int policyPass : 1;
} TPM2_AUTH_SESSION;

/* Macros to determine TPM 2.0 Session type */
#define TPM2_IS_PWD_SESSION(sessionHandle) ((sessionHandle) == TPM_RS_PW)
#define TPM2_IS_HMAC_SESSION(sessionHandle) ((sessionHandle & 0xFF000000) ==
 ↪  HMAC_SESSION_FIRST)
#define TPM2_IS_POLICY_SESSION(sessionHandle) ((sessionHandle & 0xFF000000) ==
 ↪  POLICY_SESSION_FIRST)

/* Predetermined TPM 2.0 Indexes */
#define TPM_20_TPM_MFG_NV_SPACE        ((TPM_HT_NV_INDEX << 24) | (0x00 << 22))
#define TPM_20_PLATFORM_MFG_NV_SPACE   ((TPM_HT_NV_INDEX << 24) | (0x01 << 22))
#define TPM_20_OWNER_NV_SPACE          ((TPM_HT_NV_INDEX << 24) | (0x02 << 22))
#define TPM_20_TCG_NV_SPACE            ((TPM_HT_NV_INDEX << 24) | (0x03 << 22))

/* EK (Low Range): RSA 2048 */
#define TPM2_NV_RSA_EK_CERT            (TPM_20_TCG_NV_SPACE + 0x2)
#define TPM2_NV_RSA_EK_NONCE           (TPM_20_TCG_NV_SPACE + 0x3)
#define TPM2_NV_RSA_EK_TEMPLATE        (TPM_20_TCG_NV_SPACE + 0x4)
```

```
/* EK (Low Range): ECC P256 */
#define TPM2_NV_ECC_EK_CERT             (TPM_20_TCG_NV_SPACE + 0xA)
#define TPM2_NV_ECC_EK_NONCE            (TPM_20_TCG_NV_SPACE + 0xB)
#define TPM2_NV_ECC_EK_TEMPLATE         (TPM_20_TCG_NV_SPACE + 0xC)

/* EK (High Range) */
#define TPM2_NV_EK_RSA2048              (TPM_20_TCG_NV_SPACE + 0x12)
#define TPM2_NV_EK_ECC_P256             (TPM_20_TCG_NV_SPACE + 0x14)
#define TPM2_NV_EK_ECC_P384             (TPM_20_TCG_NV_SPACE + 0x16)
#define TPM2_NV_EK_ECC_P521             (TPM_20_TCG_NV_SPACE + 0x18)
#define TPM2_NV_EK_ECC_SM2              (TPM_20_TCG_NV_SPACE + 0x1A)
#define TPM2_NV_EK_RSA3072              (TPM_20_TCG_NV_SPACE + 0x1C)
#define TPM2_NV_EK_RSA4096              (TPM_20_TCG_NV_SPACE + 0x1E)

/* EK Certificate Chains (0x100 - 0x1FF) - Not common */
#define TPM2_NV_EK_CHAIN                (TPM_20_TCG_NV_SPACE + 0x100)

/* EK Policy Indices for PolicyAuthorizeNV (0x7F01 - 0x7F04) */
#define TPM2_NV_EK_POLICY_SHA256        (TPM_20_TCG_NV_SPACE + 0x7F01)
#define TPM2_NV_EK_POLICY_SHA384        (TPM_20_TCG_NV_SPACE + 0x7F02)
#define TPM2_NV_EK_POLICY_SHA512        (TPM_20_TCG_NV_SPACE + 0x7F03)
#define TPM2_NV_EK_POLICY_SM3_256       (TPM_20_TCG_NV_SPACE + 0x7F04)

/* Predetermined TPM 2.0 Endorsement policy auth templates */
/* SHA256 (Low Range) */
static const BYTE TPM_20_EK_AUTH_POLICY[] = {
    0x83, 0x71, 0x97, 0x67, 0x44, 0x84, 0xB3, 0xF8,
    0x1A, 0x90, 0xCC, 0x8D, 0x46, 0xA5, 0xD7, 0x24,
    0xFD, 0x52, 0xD7, 0x6E, 0x06, 0x52, 0x0B, 0x64,
    0xF2, 0xA1, 0xDA, 0x1B, 0x33, 0x14, 0x69, 0xAA
};
/* SHA256 (PolicyB - High Range) */
static const BYTE TPM_20_EK_AUTH_POLICY_SHA256[] = {
    0xCA, 0x3D, 0x0A, 0x99, 0xA2, 0xB9, 0x39, 0x06,
    0xF7, 0xA3, 0x34, 0x24, 0x14, 0xEF, 0xCF, 0xB3,
    0xA3, 0x85, 0xD4, 0x4C, 0xD1, 0xFD, 0x45, 0x90,
    0x89, 0xD1, 0x9B, 0x50, 0x71, 0xC0, 0xB7, 0xA0
};
#ifdef WOLFSSL_SHA384
/* SHA384 (PolicyB - High Range) */
static const BYTE TPM_20_EK_AUTH_POLICY_SHA384[] = {
    0xB2, 0x6E, 0x7D, 0x28, 0xD1, 0x1A, 0x50, 0xBC,
    0x53, 0xD8, 0x82, 0xBC, 0xF5, 0xFD, 0x3A, 0x1A,
    0x07, 0x41, 0x48, 0xBB, 0x35, 0xD3, 0xB4, 0xE4,
    0xCB, 0x1C, 0x0A, 0xD9, 0xBD, 0xE4, 0x19, 0xCA,
    0xCB, 0x47, 0xBA, 0x09, 0x69, 0x96, 0x46, 0x15,
    0x0F, 0x9F, 0xC0, 0x00, 0xF3, 0xF8, 0x0E, 0x12
};
#endif
#ifdef WOLFSSL_SHA512
/* SHA512 (PolicyB - High Range) */
static const BYTE TPM_20_EK_AUTH_POLICY_SHA512[] = {
    0xB8, 0x22, 0x1C, 0xA6, 0x9E, 0x85, 0x50, 0xA4,
```

```
    0x91, 0x4D, 0xE3, 0xFA, 0xA6, 0xA1, 0x8C, 0x07,
    0x2C, 0xC0, 0x12, 0x08, 0x07, 0x3A, 0x92, 0x8D,
    0x5D, 0x66, 0xD5, 0x9E, 0xF7, 0x9E, 0x49, 0xA4,
    0x29, 0xC4, 0x1A, 0x6B, 0x26, 0x95, 0x71, 0xD5,
    0x7E, 0xDB, 0x25, 0xFB, 0xDB, 0x18, 0x38, 0x42,
    0x56, 0x08, 0xB4, 0x13, 0xCD, 0x61, 0x6A, 0x5F,
    0x6D, 0xB5, 0xB6, 0x07, 0x1A, 0xF9, 0x9B, 0xEA
};
#endif

#ifdef WOLFTPM_PROVISIONING
/* Precalcualted IDevID/IAK Policies */
/* PolicyOR:
 * 1: PolicyUser (section 7.3.6.1)
 * 2: PolicyCertify (section 7.3.6.2)
 * 3: PolicyActivateCredential (section 7.3.6.3)
 * 4: PolicyDelegationNV (section 7.3.6.4)*/
static const BYTE TPM_20_IDEVID_POLICY[] = {
    0xAD, 0x6B, 0x3A, 0x22, 0x84, 0xFD, 0x69, 0x8A,
    0x07, 0x10, 0xBF, 0x5C, 0xC1, 0xB9, 0xBD, 0xF1,
    0x5E, 0x25, 0x32, 0xE3, 0xF6, 0x01, 0xFA, 0x4B,
    0x93, 0xA6, 0xA8, 0xFA, 0x8D, 0xE5, 0x79, 0xEA
};
static const BYTE TPM_20_IAK_POLICY[] = {
    0x54, 0x37, 0x18, 0x23, 0x26, 0xE4, 0x14, 0xFC,
    0xA7, 0x97, 0xD5, 0xF1, 0x74, 0x61, 0x5A, 0x16,
    0x41, 0xF6, 0x12, 0x55, 0x79, 0x7C, 0x3A, 0x2B,
    0x22, 0xC2, 0x1D, 0x12, 0x0B, 0x2D, 0x1E, 0x07
};
#ifdef WOLFSSL_SHA384
static const BYTE TPM_20_IDEVID_POLICY_SHA384[] = {
    0x4D, 0xB1, 0xAA, 0x83, 0x6D, 0x0B, 0x56, 0x15,
    0xDF, 0x6E, 0xE5, 0x3A, 0x40, 0xEF, 0x70, 0xC6,
    0x1C, 0x21, 0x7F, 0x43, 0x03, 0xD4, 0x46, 0x95,
    0x92, 0x59, 0x72, 0xBC, 0x92, 0x70, 0x06, 0xCF,
    0xA5, 0xCB, 0xDF, 0x6D, 0xC1, 0x8C, 0x4D, 0xBE,
    0x32, 0x9B, 0x2F, 0x15, 0x42, 0xC3, 0xDD, 0x33
};
static const BYTE TPM_20_IAK_POLICY_SHA384[] = {
    0x12, 0x9D, 0x94, 0xEB, 0xF8, 0x45, 0x56, 0x65,
    0x2C, 0x6E, 0xEF, 0x43, 0xBB, 0xB7, 0x57, 0x51,
    0x2A, 0xC8, 0x7E, 0x52, 0xBE, 0x7B, 0x34, 0x9C,
    0xA6, 0xCE, 0x4D, 0x82, 0x6F, 0x74, 0x9F, 0xCF,
    0x67, 0x2F, 0x51, 0x71, 0x6C, 0x5C, 0xBB, 0x60,
    0x5F, 0x31, 0x3B, 0xF3, 0x45, 0xAA, 0xB3, 0x12
};
#endif
#ifdef WOLFSSL_SHA512
static const BYTE TPM_20_IDEVID_POLICY_SHA512[] = {
    0x7D, 0xD7, 0x50, 0x0F, 0xD6, 0xC1, 0xB9, 0x4F,
    0x97, 0xA6, 0xAF, 0x91, 0x0D, 0xA1, 0x47, 0x30,
    0x1E, 0xF2, 0x8F, 0x66, 0x2F, 0xEE, 0x06, 0xF2,
    0x25, 0xA4, 0xCC, 0xAD, 0xDA, 0x3B, 0x4E, 0x6B,
    0x38, 0xE6, 0x6B, 0x2F, 0x3A, 0xD5, 0xDE, 0xE1,
```

```
    0xA0, 0x50, 0x3C, 0xD2, 0xDA, 0xED, 0xB1, 0xE6,
    0x8C, 0xFE, 0x4F, 0x84, 0xB0, 0x3A, 0x8C, 0xD2,
    0x2B, 0xB6, 0xA9, 0x76, 0xF0, 0x71, 0xA7, 0x2F
};
static const BYTE TPM_20_IAK_POLICY_SHA512[] = {
    0x80, 0x60, 0xD1, 0xFB, 0x31, 0x71, 0x6A, 0x29,
    0xE4, 0x8A, 0x6E, 0x5F, 0xEC, 0xE0, 0x88, 0xBC,
    0xFC, 0x1B, 0x27, 0x8F, 0xC1, 0x62, 0x25, 0x5E,
    0x81, 0xC3, 0xEC, 0xA3, 0x54, 0x4C, 0xD4, 0x4A,
    0xF9, 0x44, 0x10, 0xC3, 0x71, 0x5D, 0x56, 0x1C,
    0xCC, 0xD9, 0xE3, 0x9A, 0x6C, 0xB2, 0x64, 0x6D,
    0x43, 0x53, 0x5B, 0xB5, 0x4E, 0xA8, 0x87, 0x10,
    0xDE, 0xB5, 0xF7, 0x83, 0x6B, 0xD9, 0xB5, 0x86
};
#endif
#endif /* WOLFTPM_PROVISIONING */

/* HAL IO Callbacks */
struct TPM2_CTX;

#ifdef WOLFTPM_SWTPM
struct wolfTPM_tcpContext {
#if defined(_WIN32) && !defined(WOLFTPM_SWTPM_UART)
    /* A Winsock SOCKET is an unsigned pointer-width handle, which an int
     * truncates on a 64 bit build. Spelled without the socket headers so
     * this one stays out of the public include path. */
    uintptr_t fd;
#else
    int fd;
#endif
};
#endif /* WOLFTPM_SWTPM */

#ifdef WOLFTPM_WINAPI
#include <tbs.h>
#include <winerror.h>

struct wolfTPM_winContext {
  TBS_HCONTEXT tbs_context;
};
/* may be needed with msys */
#ifndef TPM_E_COMMAND_BLOCKED
#define TPM_E_COMMAND_BLOCKED (0x80280400)
#endif

#endif /* WOLFTPM_WINAPI */

/* Compare a return code against a TPM_RC, masking off vendor/layer high bits
 * so a vendor-decorated code (NS350 returns 0x000b0143 for 0x143) still
 * matches. Gate on >= 0 so a propagated negative wolfCrypt error (e.g. -189)
 * is never misread as a TPM response code. */
#define WOLFTPM_RC_IS(code, rc) \
    ((code) >= 0 && (((UINT32)(code)) & 0xFFFFu) == (UINT32)(rc))
```

```
/* The TPM does not implement this command. TPM_E_COMMAND_BLOCKED is a Windows
 * HRESULT (negative), so it is matched exactly rather than masked. */
#ifdef WOLFTPM_WINAPI
    #define WOLFTPM_IS_COMMAND_UNAVAILABLE(code) \
        (WOLFTPM_RC_IS(code, TPM_RC_COMMAND_CODE) || \
         (code) == (int)TPM_E_COMMAND_BLOCKED)
#else
    #define WOLFTPM_IS_COMMAND_UNAVAILABLE(code) \
        WOLFTPM_RC_IS(code, TPM_RC_COMMAND_CODE)
#endif

/* Implemented but switched off, commonly TPM2_EncryptDecrypt for export
 * controls; answers TPM_RC_DISABLED not TPM_RC_COMMAND_CODE. */
#define WOLFTPM_IS_COMMAND_DISABLED(code) \
    WOLFTPM_RC_IS(code, TPM_RC_DISABLED)

/* Either form of "the TPM will not run this command". */
#define WOLFTPM_IS_COMMAND_UNAVAILABLE_OR_DISABLED(code) \
    (WOLFTPM_IS_COMMAND_UNAVAILABLE(code) || WOLFTPM_IS_COMMAND_DISABLED(code))

/* make sure advanced IO is enabled for I2C */
#ifdef WOLFTPM_I2C
    #undef  WOLFTPM_ADV_IO
    #define WOLFTPM_ADV_IO
#endif

#ifdef WOLFTPM_ADV_IO
typedef int (*TPM2HalIoCb)(struct TPM2_CTX*, INT32 isRead, UINT32 addr,
    BYTE* xferBuf, UINT16 xferSz, void* userCtx);
#else
typedef int (*TPM2HalIoCb)(struct TPM2_CTX*, const BYTE* txBuf, BYTE* rxBuf,
    UINT16 xferSz, void* userCtx);
#endif

/* Use local wolfCrypt RNG for nonce generation by default.
 * This is recommended because nonces generated via TPM2_GetRandom travel over
 * the SPI/I2C bus unprotected (no session exists yet), making them vulnerable
 * to physical bus tampering (e.g., an attacker driving MISO to force a known
 * nonce value). A known or attacker-controlled nonceCaller in an unsalted
 * session can undermine session security and enable active manipulation.
 * Even with a salted session, it violates the TPM 2.0
 * spec assumption that nonces are unpredictable (Part 1, Section 19.6).
 * Only define WOLFTPM2_USE_HW_RNG if the bus is physically secure. */
#if !defined(WOLFTPM2_NO_WOLFCRYPT) && !defined(WC_NO_RNG) && \
    !defined(WOLFTPM2_USE_HW_RNG)
    #define WOLFTPM2_USE_WOLF_RNG
#endif

#if MAX_RESPONSE_SIZE > MAX_COMMAND_SIZE
#define XFER_MAX_SIZE MAX_RESPONSE_SIZE
#else
#define XFER_MAX_SIZE MAX_COMMAND_SIZE
#endif
```

```
typedef struct TPM2_CTX {
    TPM2HalIoCb ioCb;
    void* userCtx;
#ifdef WOLFTPM_SWTPM
    struct wolfTPM_tcpContext tcpCtx;
#endif
#ifdef WOLFTPM_WINAPI
    struct wolfTPM_winContext winCtx;
#endif
#ifdef WOLFTPM_OPTEE
    void* opteeCtx; /* TEE session, see src/tpm2_optee.c */
#endif
#ifndef WOLFTPM2_NO_WOLFCRYPT
    #ifdef WOLFTPM2_USE_WOLF_RNG
    WC_RNG rng;
    #endif
#endif /* !WOLFTPM2_NO_WOLFCRYPT */

    /* TPM TIS Info */
    int locality;
    word32 caps;
    word32 did_vid;

    /* Pointer to current TPM auth sessions */
    TPM2_AUTH_SESSION* session;

    /* Command / Response Buffer */
    byte cmdBuf[XFER_MAX_SIZE];

    byte rid;
    /* Informational Bits - use unsigned int for best compiler compatibility */
#ifndef WOLFTPM2_NO_WOLFCRYPT
    #ifndef WC_NO_RNG
    unsigned int rngInit:1;
    #endif
#endif
    unsigned int initialized:1; /* context has been through TPM2_Init_ex */
#if defined(WOLFTPM_LINUX_DEV) || defined(WOLFTPM_LINUX_DEV_AUTODETECT)
    int fd;
#endif
#ifdef WOLFTPM_SPDM
    void* spdmCtx; /* Pointer to WOLFTPM2_SPDM_CTX when session active */
    unsigned int spdmOnlyDetected:1; /* TPM_RC_DISABLED from Startup */
#endif

#ifndef WOLFTPM_NO_RETRY
    /* Additional resubmit attempts on TPM_RC_RETRY (0 disables) */
    int retries;
#endif
} TPM2_CTX;


/* TPM Specification Functions */
typedef struct {
```

```
    TPM_SU startupType;
} Startup_In;
WOLFTPM_API TPM_RC TPM2_Startup(Startup_In* in);

typedef struct {
    TPM_SU shutdownType;
} Shutdown_In;
WOLFTPM_API TPM_RC TPM2_Shutdown(Shutdown_In* in);


typedef struct {
    TPM_CAP capability;
    UINT32 property;
    UINT32 propertyCount;
} GetCapability_In;
typedef struct {
    TPMI_YES_NO moreData;
    TPMS_CAPABILITY_DATA capabilityData;
} GetCapability_Out;
WOLFTPM_API TPM_RC TPM2_GetCapability(GetCapability_In* in,
    GetCapability_Out* out);

/* TPM2_SetCapability (TPM 2.0 Library v1.85). Sets one settable feature
 * configuration at a time. Requires Platform authorization. */
typedef struct {
    TPMI_RH_HIERARCHY authHandle; /* TPM_RH_PLATFORM */
    TPM_CAP capability;           /* TPM_CAP_CONFIGS */
    TPMS_TAGGED_CONFIG config;    /* the feature to enable/lock */
} SetCapability_In;
WOLFTPM_API TPM_RC TPM2_SetCapability(SetCapability_In* in);

typedef struct {
    TPMI_YES_NO fullTest;
} SelfTest_In;
WOLFTPM_API TPM_RC TPM2_SelfTest(SelfTest_In* in);

typedef struct {
    TPML_ALG toTest;
} IncrementalSelfTest_In;
typedef struct {
    TPML_ALG toDoList;
} IncrementalSelfTest_Out;
WOLFTPM_API TPM_RC TPM2_IncrementalSelfTest(IncrementalSelfTest_In* in,
    IncrementalSelfTest_Out* out);

typedef struct {
    TPM2B_MAX_BUFFER outData;
    UINT32 testResult; /* full 4-byte TPM_RC value on the wire */
} GetTestResult_Out;
WOLFTPM_API TPM_RC TPM2_GetTestResult(GetTestResult_Out* out);


typedef struct {
    UINT16 bytesRequested;
```

```
} GetRandom_In;
typedef struct {
    TPM2B_DIGEST randomBytes; /* hardware max is 32-bytes */
} GetRandom_Out;
WOLFTPM_API TPM_RC TPM2_GetRandom(GetRandom_In* in, GetRandom_Out* out);

typedef struct {
    TPM2B_SENSITIVE_DATA inData;
} StirRandom_In;
WOLFTPM_API TPM_RC TPM2_StirRandom(StirRandom_In* in);

typedef struct {
    TPML_PCR_SELECTION pcrSelectionIn;
} PCR_Read_In;
typedef struct {
    UINT32 pcrUpdateCounter;
    TPML_PCR_SELECTION pcrSelectionOut;
    TPML_DIGEST pcrValues;
} PCR_Read_Out;
WOLFTPM_API TPM_RC TPM2_PCR_Read(PCR_Read_In* in, PCR_Read_Out* out);


typedef struct {
    TPMI_DH_PCR pcrHandle;
    TPML_DIGEST_VALUES digests;
} PCR_Extend_In;
WOLFTPM_API TPM_RC TPM2_PCR_Extend(PCR_Extend_In* in);


typedef struct {
    TPMI_DH_OBJECT parentHandle;
    TPM2B_SENSITIVE_CREATE inSensitive;
    TPM2B_PUBLIC inPublic;
    TPM2B_DATA outsideInfo;
    TPML_PCR_SELECTION creationPCR;
} Create_In;
typedef struct {
    TPM2B_PRIVATE outPrivate;
    TPM2B_PUBLIC outPublic;
    TPM2B_CREATION_DATA creationData;
    TPM2B_DIGEST creationHash;
    TPMT_TK_CREATION creationTicket;
} Create_Out;
WOLFTPM_API TPM_RC TPM2_Create(Create_In* in, Create_Out* out);

typedef struct {
    TPMI_DH_OBJECT parentHandle;
    TPM2B_SENSITIVE_CREATE inSensitive;
    TPM2B_PUBLIC inPublic;
} CreateLoaded_In;
typedef struct {
    TPM_HANDLE objectHandle;
    TPM2B_PRIVATE outPrivate;
    TPM2B_PUBLIC outPublic;
```

```
    TPM2B_NAME name;
} CreateLoaded_Out;
WOLFTPM_API TPM_RC TPM2_CreateLoaded(CreateLoaded_In* in,
    CreateLoaded_Out* out);


typedef struct {
    TPMI_RH_HIERARCHY primaryHandle;
    TPM2B_SENSITIVE_CREATE inSensitive;
    TPM2B_PUBLIC inPublic;
    TPM2B_DATA outsideInfo;
    TPML_PCR_SELECTION creationPCR;
} CreatePrimary_In;
typedef struct {
    TPM_HANDLE objectHandle;
    TPM2B_PUBLIC outPublic;
    TPM2B_CREATION_DATA creationData;
    TPM2B_DIGEST creationHash;
    TPMT_TK_CREATION creationTicket;
    TPM2B_NAME name;
} CreatePrimary_Out;
WOLFTPM_API TPM_RC TPM2_CreatePrimary(CreatePrimary_In* in,
    CreatePrimary_Out* out);

typedef struct {
    TPMI_DH_OBJECT parentHandle;
    TPM2B_PRIVATE inPrivate;
    TPM2B_PUBLIC inPublic;
} Load_In;
typedef struct {
    TPM_HANDLE objectHandle;
    TPM2B_NAME name;
} Load_Out;
WOLFTPM_API TPM_RC TPM2_Load(Load_In* in, Load_Out* out);


typedef struct {
    TPMI_DH_CONTEXT flushHandle;
} FlushContext_In;
WOLFTPM_API TPM_RC TPM2_FlushContext(FlushContext_In* in);


typedef struct {
    TPMI_DH_OBJECT itemHandle;
} Unseal_In;
typedef struct {
    TPM2B_SENSITIVE_DATA outData;
} Unseal_Out;
WOLFTPM_API TPM_RC TPM2_Unseal(Unseal_In* in, Unseal_Out* out);


typedef struct {
    TPMI_DH_OBJECT tpmKey;
    TPMI_DH_ENTITY bind;
```

```
    TPM2B_NONCE nonceCaller;
    TPM2B_ENCRYPTED_SECRET encryptedSalt;
    TPM_SE sessionType;
    TPMT_SYM_DEF symmetric;
    TPMI_ALG_HASH authHash;
} StartAuthSession_In;
typedef struct {
    TPMI_SH_AUTH_SESSION sessionHandle;
    TPM2B_NONCE nonceTPM;
} StartAuthSession_Out;
WOLFTPM_API TPM_RC TPM2_StartAuthSession(StartAuthSession_In* in,
    StartAuthSession_Out* out);

typedef struct {
    TPMI_SH_POLICY sessionHandle;
} PolicyRestart_In;
WOLFTPM_API TPM_RC TPM2_PolicyRestart(PolicyRestart_In* in);


typedef struct {
    TPM2B_SENSITIVE inPrivate;
    TPM2B_PUBLIC inPublic;
    TPMI_RH_HIERARCHY hierarchy;
} LoadExternal_In;
typedef struct {
    TPM_HANDLE objectHandle;
    TPM2B_NAME name;
} LoadExternal_Out;
WOLFTPM_API TPM_RC TPM2_LoadExternal(LoadExternal_In* in,
    LoadExternal_Out* out);

typedef struct {
    TPMI_DH_OBJECT objectHandle;
} ReadPublic_In;
typedef struct {
    TPM2B_PUBLIC outPublic;
    TPM2B_NAME name;
    TPM2B_NAME qualifiedName;
} ReadPublic_Out;
WOLFTPM_API TPM_RC TPM2_ReadPublic(ReadPublic_In* in, ReadPublic_Out* out);

typedef struct {
    TPMI_DH_OBJECT activateHandle;
    TPMI_DH_OBJECT keyHandle;
    TPM2B_ID_OBJECT credentialBlob;
    TPM2B_ENCRYPTED_SECRET secret;
} ActivateCredential_In;
typedef struct {
    TPM2B_DIGEST certInfo;
} ActivateCredential_Out;
WOLFTPM_API TPM_RC TPM2_ActivateCredential(ActivateCredential_In* in,
    ActivateCredential_Out* out);

typedef struct {
```

```
    TPMI_DH_OBJECT handle;
    TPM2B_DIGEST credential;
    TPM2B_NAME objectName;
} MakeCredential_In;
typedef struct {
    TPM2B_ID_OBJECT credentialBlob;
    TPM2B_ENCRYPTED_SECRET secret;
} MakeCredential_Out;
WOLFTPM_API TPM_RC TPM2_MakeCredential(MakeCredential_In* in,
    MakeCredential_Out* out);

typedef struct {
    TPMI_DH_OBJECT objectHandle;
    TPMI_DH_OBJECT parentHandle;
    TPM2B_AUTH newAuth;
} ObjectChangeAuth_In;
typedef struct {
    TPM2B_PRIVATE outPrivate;
} ObjectChangeAuth_Out;
WOLFTPM_API TPM_RC TPM2_ObjectChangeAuth(ObjectChangeAuth_In* in,
    ObjectChangeAuth_Out* out);


typedef struct {
    TPMI_DH_OBJECT objectHandle;
    TPMI_DH_OBJECT newParentHandle;
    TPM2B_DATA encryptionKeyIn;
    TPMT_SYM_DEF_OBJECT symmetricAlg;
} Duplicate_In;
typedef struct {
    TPM2B_DATA encryptionKeyOut;
    TPM2B_PRIVATE duplicate;
    TPM2B_ENCRYPTED_SECRET outSymSeed;
} Duplicate_Out;
WOLFTPM_API TPM_RC TPM2_Duplicate(Duplicate_In* in, Duplicate_Out* out);

typedef struct {
    TPMI_DH_OBJECT oldParent;
    TPMI_DH_OBJECT newParent;
    TPM2B_PRIVATE inDuplicate;
    TPM2B_NAME name;
    TPM2B_ENCRYPTED_SECRET inSymSeed;
} Rewrap_In;
typedef struct {
    TPM2B_PRIVATE outDuplicate;
    TPM2B_ENCRYPTED_SECRET outSymSeed;
} Rewrap_Out;
WOLFTPM_API TPM_RC TPM2_Rewrap(Rewrap_In* in, Rewrap_Out* out);

typedef struct {
    TPMI_DH_OBJECT parentHandle;
    TPM2B_DATA encryptionKey;
    TPM2B_PUBLIC objectPublic;
    TPM2B_PRIVATE duplicate;
```

```
    TPM2B_ENCRYPTED_SECRET inSymSeed;
    TPMT_SYM_DEF_OBJECT symmetricAlg;
} Import_In;
typedef struct {
    TPM2B_PRIVATE outPrivate;
} Import_Out;
WOLFTPM_API TPM_RC TPM2_Import(Import_In* in, Import_Out* out);

typedef struct {
    TPMI_DH_OBJECT keyHandle;
    TPM2B_PUBLIC_KEY_RSA message;
    TPMT_RSA_DECRYPT inScheme;
    TPM2B_DATA label;
} RSA_Encrypt_In;
typedef struct {
    TPM2B_PUBLIC_KEY_RSA outData;
} RSA_Encrypt_Out;
WOLFTPM_API TPM_RC TPM2_RSA_Encrypt(RSA_Encrypt_In* in, RSA_Encrypt_Out* out);


typedef struct {
    TPMI_DH_OBJECT keyHandle;
    TPM2B_PUBLIC_KEY_RSA cipherText;
    TPMT_RSA_DECRYPT inScheme;
    TPM2B_DATA label;
} RSA_Decrypt_In;
typedef struct {
    TPM2B_PUBLIC_KEY_RSA message;
} RSA_Decrypt_Out;
WOLFTPM_API TPM_RC TPM2_RSA_Decrypt(RSA_Decrypt_In* in, RSA_Decrypt_Out* out);


typedef struct {
    TPMI_DH_OBJECT keyHandle;
} ECDH_KeyGen_In;
typedef struct {
    TPM2B_ECC_POINT zPoint;
    TPM2B_ECC_POINT pubPoint;
} ECDH_KeyGen_Out;
WOLFTPM_API TPM_RC TPM2_ECDH_KeyGen(ECDH_KeyGen_In* in, ECDH_KeyGen_Out* out);


typedef struct {
    TPMI_DH_OBJECT keyHandle;
    TPM2B_ECC_POINT inPoint;
} ECDH_ZGen_In;
typedef struct {
    TPM2B_ECC_POINT outPoint;
} ECDH_ZGen_Out;
WOLFTPM_API TPM_RC TPM2_ECDH_ZGen(ECDH_ZGen_In* in, ECDH_ZGen_Out* out);

typedef struct {
    TPMI_ECC_CURVE curveID;
} ECC_Parameters_In;
```

```
typedef struct {
    TPMS_ALGORITHM_DETAIL_ECC parameters;
} ECC_Parameters_Out;
WOLFTPM_API TPM_RC TPM2_ECC_Parameters(ECC_Parameters_In* in,
    ECC_Parameters_Out* out);

typedef struct {
    TPMI_DH_OBJECT keyA;
    TPM2B_ECC_POINT inQsB;
    TPM2B_ECC_POINT inQeB;
    TPMI_ECC_KEY_EXCHANGE inScheme;
    UINT16 counter;
} ZGen_2Phase_In;
typedef struct {
    TPM2B_ECC_POINT outZ1;
    TPM2B_ECC_POINT outZ2;
} ZGen_2Phase_Out;
WOLFTPM_API TPM_RC TPM2_ZGen_2Phase(ZGen_2Phase_In* in, ZGen_2Phase_Out* out);


typedef struct {
    TPMI_DH_OBJECT keyHandle;
    TPMI_YES_NO decrypt;
    TPMI_ALG_SYM_MODE mode;
    TPM2B_IV ivIn;
    TPM2B_MAX_BUFFER inData;
} EncryptDecrypt_In;
typedef struct {
    TPM2B_MAX_BUFFER outData;
    TPM2B_IV ivOut;
} EncryptDecrypt_Out;
WOLFTPM_API TPM_RC TPM2_EncryptDecrypt(EncryptDecrypt_In* in,
    EncryptDecrypt_Out* out);

typedef struct {
    TPMI_DH_OBJECT keyHandle;
    TPM2B_MAX_BUFFER inData;
    TPMI_YES_NO decrypt;
    TPMI_ALG_SYM_MODE mode;
    TPM2B_IV ivIn;
} EncryptDecrypt2_In;
typedef struct {
    TPM2B_MAX_BUFFER outData;
    TPM2B_IV ivOut;
} EncryptDecrypt2_Out;
WOLFTPM_API TPM_RC TPM2_EncryptDecrypt2(EncryptDecrypt2_In* in,
    EncryptDecrypt2_Out* out);


typedef struct {
    TPM2B_MAX_BUFFER data;
    TPMI_ALG_HASH hashAlg;
    TPMI_RH_HIERARCHY hierarchy;
} Hash_In;
```

```
typedef struct {
    TPM2B_DIGEST outHash;
    TPMT_TK_HASHCHECK validation;
} Hash_Out;
WOLFTPM_API TPM_RC TPM2_Hash(Hash_In* in, Hash_Out* out);

typedef struct {
    TPMI_DH_OBJECT handle;
    TPM2B_MAX_BUFFER buffer;
    TPMI_ALG_HASH hashAlg;
} HMAC_In;
typedef struct {
    TPM2B_DIGEST outHMAC;
} HMAC_Out;
WOLFTPM_API TPM_RC TPM2_HMAC(HMAC_In* in, HMAC_Out* out);


typedef struct {
    TPMI_DH_OBJECT handle;
    TPM2B_AUTH auth;
    TPMI_ALG_HASH hashAlg;
} HMAC_Start_In;
typedef struct {
    TPMI_DH_OBJECT sequenceHandle;
} HMAC_Start_Out;
WOLFTPM_API TPM_RC TPM2_HMAC_Start(HMAC_Start_In* in, HMAC_Start_Out* out);


typedef struct {
    TPM2B_AUTH auth;
    TPMI_ALG_HASH hashAlg;
} HashSequenceStart_In;
typedef struct {
    TPMI_DH_OBJECT sequenceHandle;
} HashSequenceStart_Out;
WOLFTPM_API TPM_RC TPM2_HashSequenceStart(HashSequenceStart_In* in,
    HashSequenceStart_Out* out);

typedef struct {
    TPMI_DH_OBJECT sequenceHandle;
    TPM2B_MAX_BUFFER buffer;
} SequenceUpdate_In;
WOLFTPM_API TPM_RC TPM2_SequenceUpdate(SequenceUpdate_In* in);

typedef struct {
    TPMI_DH_OBJECT sequenceHandle;
    TPM2B_MAX_BUFFER buffer;
    TPMI_RH_HIERARCHY hierarchy;
} SequenceComplete_In;
typedef struct {
    TPM2B_DIGEST result;
    TPMT_TK_HASHCHECK validation;
} SequenceComplete_Out;
WOLFTPM_API TPM_RC TPM2_SequenceComplete(SequenceComplete_In* in,
```

```
    SequenceComplete_Out* out);


typedef struct {
    TPMI_DH_PCR pcrHandle;
    TPMI_DH_OBJECT sequenceHandle;
    TPM2B_MAX_BUFFER buffer;
} EventSequenceComplete_In;
typedef struct {
    TPML_DIGEST_VALUES results;
} EventSequenceComplete_Out;
WOLFTPM_API TPM_RC TPM2_EventSequenceComplete(EventSequenceComplete_In* in,
    EventSequenceComplete_Out* out);


typedef struct {
    TPMI_DH_OBJECT objectHandle;
    TPMI_DH_OBJECT signHandle;
    TPM2B_DATA qualifyingData;
    TPMT_SIG_SCHEME inScheme;
} Certify_In;
typedef struct {
    TPM2B_ATTEST certifyInfo;
    TPMT_SIGNATURE signature;
} Certify_Out;
WOLFTPM_API TPM_RC TPM2_Certify(Certify_In* in, Certify_Out* out);


typedef struct {
    TPMI_DH_OBJECT signHandle;
    TPMI_DH_OBJECT objectHandle;
    TPM2B_DATA qualifyingData;
    TPM2B_DIGEST creationHash;
    TPMT_SIG_SCHEME inScheme;
    TPMT_TK_CREATION creationTicket;
} CertifyCreation_In;
typedef struct {
    TPM2B_ATTEST certifyInfo;
    TPMT_SIGNATURE signature;
} CertifyCreation_Out;
WOLFTPM_API TPM_RC TPM2_CertifyCreation(CertifyCreation_In* in,
 ↪  CertifyCreation_Out* out);


typedef struct {
    TPMI_DH_OBJECT signHandle;
    TPM2B_DATA qualifyingData;
    TPMT_SIG_SCHEME inScheme;
    TPML_PCR_SELECTION PCRselect;
} Quote_In;
typedef struct {
    TPM2B_ATTEST quoted;
    TPMT_SIGNATURE signature;
} Quote_Out;
```

```
WOLFTPM_API TPM_RC TPM2_Quote(Quote_In* in, Quote_Out* out);

typedef struct {
    TPMI_RH_ENDORSEMENT privacyAdminHandle;
    TPMI_DH_OBJECT signHandle;
    TPMI_SH_HMAC sessionHandle;
    TPM2B_DATA qualifyingData;
    TPMT_SIG_SCHEME inScheme;
} GetSessionAuditDigest_In;
typedef struct {
    TPM2B_ATTEST auditInfo;
    TPMT_SIGNATURE signature;
} GetSessionAuditDigest_Out;
WOLFTPM_API TPM_RC TPM2_GetSessionAuditDigest(GetSessionAuditDigest_In* in,
    GetSessionAuditDigest_Out* out);

typedef struct {
    TPMI_RH_ENDORSEMENT privacyHandle;
    TPMI_DH_OBJECT signHandle;
    TPM2B_DATA qualifyingData;
    TPMT_SIG_SCHEME inScheme;
} GetCommandAuditDigest_In;
typedef struct {
    TPM2B_ATTEST auditInfo;
    TPMT_SIGNATURE signature;
} GetCommandAuditDigest_Out;
WOLFTPM_API TPM_RC TPM2_GetCommandAuditDigest(GetCommandAuditDigest_In* in,
    GetCommandAuditDigest_Out* out);

typedef struct {
    TPMI_RH_ENDORSEMENT privacyAdminHandle;
    TPMI_DH_OBJECT signHandle;
    TPM2B_DATA qualifyingData;
    TPMT_SIG_SCHEME inScheme;
} GetTime_In;
typedef struct {
    TPM2B_ATTEST timeInfo;
    TPMT_SIGNATURE signature;
} GetTime_Out;
WOLFTPM_API TPM_RC TPM2_GetTime(GetTime_In* in, GetTime_Out* out);

typedef struct {
    TPMI_DH_OBJECT signHandle;
    TPM2B_ECC_POINT P1;
    TPM2B_SENSITIVE_DATA s2;
    TPM2B_ECC_PARAMETER y2;
} Commit_In;
typedef struct {
    TPM2B_ECC_POINT K;
    TPM2B_ECC_POINT L;
    TPM2B_ECC_POINT E;
    UINT16 counter;
} Commit_Out;
WOLFTPM_API TPM_RC TPM2_Commit(Commit_In* in, Commit_Out* out);
```

```
typedef struct {
    TPMI_ECC_CURVE curveID;
} EC_Ephemeral_In;
typedef struct {
    TPM2B_ECC_POINT Q;
    UINT16 counter;
} EC_Ephemeral_Out;
WOLFTPM_API TPM_RC TPM2_EC_Ephemeral(EC_Ephemeral_In* in,
    EC_Ephemeral_Out* out);

typedef struct {
    TPMI_DH_OBJECT keyHandle;
    TPM2B_DIGEST digest;
    TPMT_SIGNATURE signature;
} VerifySignature_In;
typedef struct {
    TPMT_TK_VERIFIED validation;
} VerifySignature_Out;
WOLFTPM_API TPM_RC TPM2_VerifySignature(VerifySignature_In* in,
    VerifySignature_Out* out);


typedef struct {
    TPMI_DH_OBJECT keyHandle;
    TPM2B_DIGEST digest;
    TPMT_SIG_SCHEME inScheme;
    TPMT_TK_HASHCHECK validation;
} Sign_In;
typedef struct {
    TPMT_SIGNATURE signature;
} Sign_Out;
WOLFTPM_API TPM_RC TPM2_Sign(Sign_In* in, Sign_Out* out);

#ifdef WOLFTPM_PQC
/* Post-Quantum Cryptography (PQC) Commands - TPM 2.0 v185 */

/* v185 rc4 Part 3 Sec.17.6.3 Table 89 – {keyHandle, auth, context} */
typedef struct {
    TPMI_DH_OBJECT keyHandle;
    TPM2B_AUTH auth;
    TPM2B_SIGNATURE_CTX context;
} SignSequenceStart_In;
typedef struct {
    TPMI_DH_OBJECT sequenceHandle;
} SignSequenceStart_Out;
WOLFTPM_API TPM_RC TPM2_SignSequenceStart(SignSequenceStart_In* in,
    SignSequenceStart_Out* out);

/* v185 rc4 Part 3 Sec.17.6.2 Table 87 – {keyHandle, auth, hint, context}
 * hint holds the encoded R value for EdDSA; zero-length for other schemes. */
typedef struct {
    TPMI_DH_OBJECT keyHandle;
```

```
    TPM2B_AUTH auth;
    TPM2B_SIGNATURE_HINT hint;
    TPM2B_SIGNATURE_CTX context;
} VerifySequenceStart_In;
typedef struct {
    TPMI_DH_OBJECT sequenceHandle;
} VerifySequenceStart_Out;
WOLFTPM_API TPM_RC TPM2_VerifySequenceStart(VerifySequenceStart_In* in,
    VerifySequenceStart_Out* out);

typedef struct {
    TPMI_DH_OBJECT sequenceHandle;
    TPMI_DH_OBJECT keyHandle;
    TPM2B_MAX_BUFFER buffer;
} SignSequenceComplete_In;
typedef struct {
    TPMT_SIGNATURE signature;
} SignSequenceComplete_Out;
WOLFTPM_API TPM_RC TPM2_SignSequenceComplete(SignSequenceComplete_In* in,
    SignSequenceComplete_Out* out);

/* v185 rc4 Part 3 Sec.20.3 Table 118 – {sequenceHandle, keyHandle, signature}.
 * The accumulated message lives in the sequence object on the TPM (built up
 * via TPM2_SequenceUpdate calls); there is no per-command buffer field. */
typedef struct {
    TPMI_DH_OBJECT sequenceHandle;
    TPMI_DH_OBJECT keyHandle;
    TPMT_SIGNATURE signature;
} VerifySequenceComplete_In;
typedef struct {
    TPMT_TK_VERIFIED validation;
} VerifySequenceComplete_Out;
WOLFTPM_API TPM_RC TPM2_VerifySequenceComplete(VerifySequenceComplete_In* in,
    VerifySequenceComplete_Out* out);

/* v185 rc4 Part 3 Sec.20.7.2 Table 126 – {keyHandle, context, digest,
 ↪  validation} */
typedef struct {
    TPMI_DH_OBJECT keyHandle;
    TPM2B_SIGNATURE_CTX context;
    TPM2B_DIGEST digest;
    TPMT_TK_HASHCHECK validation;
} SignDigest_In;
typedef struct {
    TPMT_SIGNATURE signature;
} SignDigest_Out;
WOLFTPM_API TPM_RC TPM2_SignDigest(SignDigest_In* in, SignDigest_Out* out);

/* v185 rc4 Part 3 Sec.20.4.2 Table 120 – {keyHandle, context, digest,
 ↪  signature} */
typedef struct {
    TPMI_DH_OBJECT keyHandle;
    TPM2B_SIGNATURE_CTX context;
    TPM2B_DIGEST digest;
```

```
    TPMT_SIGNATURE signature;
} VerifyDigestSignature_In;
typedef struct {
    TPMT_TK_VERIFIED validation;
} VerifyDigestSignature_Out;
WOLFTPM_API TPM_RC TPM2_VerifyDigestSignature(VerifyDigestSignature_In* in,
    VerifyDigestSignature_Out* out);

typedef struct {
    TPMI_DH_OBJECT keyHandle;
} Encapsulate_In;
typedef struct {
    TPM2B_SHARED_SECRET sharedSecret;
    TPM2B_KEM_CIPHERTEXT ciphertext;
} Encapsulate_Out;
WOLFTPM_API TPM_RC TPM2_Encapsulate(Encapsulate_In* in, Encapsulate_Out* out);

typedef struct {
    TPMI_DH_OBJECT keyHandle;
    TPM2B_KEM_CIPHERTEXT ciphertext;
} Decapsulate_In;
typedef struct {
    TPM2B_SHARED_SECRET sharedSecret;
} Decapsulate_Out;
WOLFTPM_API TPM_RC TPM2_Decapsulate(Decapsulate_In* in, Decapsulate_Out* out);
#endif /* WOLFTPM_PQC */

typedef struct {
    TPMI_RH_PROVISION auth;
    TPMI_ALG_HASH auditAlg;
    TPML_CC setList;
    TPML_CC clearList;
} SetCommandCodeAuditStatus_In;
WOLFTPM_API TPM_RC TPM2_SetCommandCodeAuditStatus(
    SetCommandCodeAuditStatus_In* in);


typedef struct {
    TPMI_DH_PCR pcrHandle;
    TPM2B_EVENT eventData;
} PCR_Event_In;
typedef struct {
    TPML_DIGEST_VALUES digests;
} PCR_Event_Out;
WOLFTPM_API TPM_RC TPM2_PCR_Event(PCR_Event_In* in, PCR_Event_Out* out);


typedef struct {
    TPMI_RH_PLATFORM authHandle;
    TPML_PCR_SELECTION pcrAllocation;
} PCR_Allocate_In;
typedef struct {
    TPMI_YES_NO allocationSuccess;
    UINT32 maxPCR;
```

```
    UINT32 sizeNeeded;
    UINT32 sizeAvailable;
} PCR_Allocate_Out;
WOLFTPM_API TPM_RC TPM2_PCR_Allocate(PCR_Allocate_In* in,
    PCR_Allocate_Out* out);

typedef struct {
    TPMI_RH_PLATFORM authHandle;
    TPM2B_DIGEST authPolicy;
    TPMI_ALG_HASH hashAlg;
    TPMI_DH_PCR pcrNum;
} PCR_SetAuthPolicy_In;
WOLFTPM_API TPM_RC TPM2_PCR_SetAuthPolicy(PCR_SetAuthPolicy_In* in);

typedef struct {
    TPMI_DH_PCR pcrHandle;
    TPM2B_DIGEST auth;
} PCR_SetAuthValue_In;
WOLFTPM_API TPM_RC TPM2_PCR_SetAuthValue(PCR_SetAuthValue_In* in);

typedef struct {
    TPMI_DH_PCR pcrHandle;
} PCR_Reset_In;
WOLFTPM_API TPM_RC TPM2_PCR_Reset(PCR_Reset_In* in);


typedef struct {
    TPMI_DH_OBJECT authObject;
    TPMI_SH_POLICY policySession;
    TPM2B_NONCE nonceTPM;
    TPM2B_DIGEST cpHashA;
    TPM2B_NONCE policyRef;
    INT32 expiration;
    TPMT_SIGNATURE auth;
} PolicySigned_In;
typedef struct {
    TPM2B_TIMEOUT timeout;
    TPMT_TK_AUTH policyTicket;
} PolicySigned_Out;
WOLFTPM_API TPM_RC TPM2_PolicySigned(PolicySigned_In* in,
    PolicySigned_Out* out);

typedef struct {
    TPMI_DH_ENTITY authHandle;
    TPMI_SH_POLICY policySession;
    TPM2B_NONCE nonceTPM;
    TPM2B_DIGEST cpHashA;
    TPM2B_NONCE policyRef;
    INT32 expiration;
} PolicySecret_In;
typedef struct {
    TPM2B_TIMEOUT timeout;
    TPMT_TK_AUTH policyTicket;
} PolicySecret_Out;
```

```
WOLFTPM_API TPM_RC TPM2_PolicySecret(PolicySecret_In* in,
    PolicySecret_Out* out);

typedef struct {
    TPMI_SH_POLICY policySession;
    TPM2B_TIMEOUT timeout;
    TPM2B_DIGEST cpHashA;
    TPM2B_NONCE policyRef;
    TPM2B_NAME authName;
    TPMT_TK_AUTH ticket;
} PolicyTicket_In;
WOLFTPM_API TPM_RC TPM2_PolicyTicket(PolicyTicket_In* in);

typedef struct {
    TPMI_SH_POLICY policySession;
    TPML_DIGEST pHashList;
} PolicyOR_In;
WOLFTPM_API TPM_RC TPM2_PolicyOR(PolicyOR_In* in);

typedef struct {
    TPMI_SH_POLICY policySession;
    TPM2B_DIGEST pcrDigest;
    TPML_PCR_SELECTION pcrs;
} PolicyPCR_In;
WOLFTPM_API TPM_RC TPM2_PolicyPCR(PolicyPCR_In* in);

#ifdef WOLFTPM_SPDM
typedef struct {
    TPMI_SH_POLICY policySession;
    TPM2B_NAME reqKeyName;
    TPM2B_NAME tpmKeyName;
} PolicyTransportSPDM_In;
WOLFTPM_API TPM_RC TPM2_PolicyTransportSPDM(PolicyTransportSPDM_In* in);
#endif

typedef struct {
    TPMI_SH_POLICY policySession;
    TPMA_LOCALITY locality;
} PolicyLocality_In;
WOLFTPM_API TPM_RC TPM2_PolicyLocality(PolicyLocality_In* in);

typedef struct {
    TPMI_RH_NV_AUTH authHandle;
    TPMI_RH_NV_INDEX nvIndex;
    TPMI_SH_POLICY policySession;
    TPM2B_OPERAND operandB;
    UINT16 offset;
    TPM_EO operation;
} PolicyNV_In;
WOLFTPM_API TPM_RC TPM2_PolicyNV(PolicyNV_In* in);

typedef struct {
    TPMI_SH_POLICY policySession;
    TPM2B_OPERAND operandB;
```

```
    UINT16 offset;
    TPM_EO operation;
} PolicyCounterTimer_In;
WOLFTPM_API TPM_RC TPM2_PolicyCounterTimer(PolicyCounterTimer_In* in);

typedef struct {
    TPMI_SH_POLICY policySession;
    TPM_CC code;
} PolicyCommandCode_In;
WOLFTPM_API TPM_RC TPM2_PolicyCommandCode(PolicyCommandCode_In* in);

typedef struct {
    TPMI_SH_POLICY policySession;
} PolicyPhysicalPresence_In;
WOLFTPM_API TPM_RC TPM2_PolicyPhysicalPresence(PolicyPhysicalPresence_In* in);

typedef struct {
    TPMI_SH_POLICY policySession;
    TPM2B_DIGEST cpHashA;
} PolicyCpHash_In;
WOLFTPM_API TPM_RC TPM2_PolicyCpHash(PolicyCpHash_In* in);

typedef struct {
    TPMI_SH_POLICY policySession;
    TPM2B_DIGEST nameHash;
} PolicyNameHash_In;
WOLFTPM_API TPM_RC TPM2_PolicyNameHash(PolicyNameHash_In* in);

typedef struct {
    TPMI_SH_POLICY policySession;
    TPM2B_NAME objectName;
    TPM2B_NAME newParentName;
    TPMI_YES_NO includeObject;
} PolicyDuplicationSelect_In;
WOLFTPM_API TPM_RC TPM2_PolicyDuplicationSelect(PolicyDuplicationSelect_In*
 ↪  in);

typedef struct {
    TPMI_SH_POLICY policySession;
    TPM2B_DIGEST approvedPolicy;
    TPM2B_NONCE policyRef;
    TPM2B_NAME keySign;
    TPMT_TK_VERIFIED checkTicket;
} PolicyAuthorize_In;
WOLFTPM_API TPM_RC TPM2_PolicyAuthorize(PolicyAuthorize_In* in);

typedef struct {
    TPMI_SH_POLICY policySession;
} PolicyAuthValue_In;
WOLFTPM_API TPM_RC TPM2_PolicyAuthValue(PolicyAuthValue_In* in);


typedef struct {
    TPMI_SH_POLICY policySession;
```

```
} PolicyPassword_In;
WOLFTPM_API TPM_RC TPM2_PolicyPassword(PolicyPassword_In* in);

typedef struct {
    TPMI_SH_POLICY policySession;
} PolicyGetDigest_In;
typedef struct {
    TPM2B_DIGEST policyDigest;
} PolicyGetDigest_Out;
WOLFTPM_API TPM_RC TPM2_PolicyGetDigest(PolicyGetDigest_In* in,
 ↪  PolicyGetDigest_Out* out);

typedef struct {
    TPMI_SH_POLICY policySession;
    TPMI_YES_NO writtenSet;
} PolicyNvWritten_In;
WOLFTPM_API TPM_RC TPM2_PolicyNvWritten(PolicyNvWritten_In* in);

typedef struct {
    TPMI_SH_POLICY policySession;
    TPM2B_DIGEST templateHash;
} PolicyTemplate_In;
WOLFTPM_API TPM_RC TPM2_PolicyTemplate(PolicyTemplate_In* in);

typedef struct {
    TPMI_RH_NV_AUTH authHandle;
    TPMI_RH_NV_INDEX nvIndex;
    TPMI_SH_POLICY policySession;
} PolicyAuthorizeNV_In;
WOLFTPM_API TPM_RC TPM2_PolicyAuthorizeNV(PolicyAuthorizeNV_In* in);



WOLFTPM_API void _TPM_Hash_Start(void);
WOLFTPM_API void _TPM_Hash_Data(UINT32 dataSize, BYTE *data);
WOLFTPM_API void _TPM_Hash_End(void);


typedef struct {
    TPMI_RH_HIERARCHY authHandle;
    TPMI_RH_ENABLES enable;
    TPMI_YES_NO state;
} HierarchyControl_In;
WOLFTPM_API TPM_RC TPM2_HierarchyControl(HierarchyControl_In* in);

typedef struct {
    TPMI_RH_HIERARCHY_AUTH authHandle;
    TPM2B_DIGEST authPolicy;
    TPMI_ALG_HASH hashAlg;
} SetPrimaryPolicy_In;
WOLFTPM_API TPM_RC TPM2_SetPrimaryPolicy(SetPrimaryPolicy_In* in);

typedef struct {
    TPMI_RH_PLATFORM authHandle;
```

```
} ChangeSeed_In;

typedef ChangeSeed_In ChangePPS_In;
WOLFTPM_API TPM_RC TPM2_ChangePPS(ChangePPS_In* in);

typedef ChangeSeed_In ChangeEPS_In;
WOLFTPM_API TPM_RC TPM2_ChangeEPS(ChangeEPS_In* in);


typedef struct {
    TPMI_RH_CLEAR authHandle;
} Clear_In;
WOLFTPM_API TPM_RC TPM2_Clear(Clear_In* in);

typedef struct {
    TPMI_RH_CLEAR auth;
    TPMI_YES_NO disable;
} ClearControl_In;
WOLFTPM_API TPM_RC TPM2_ClearControl(ClearControl_In* in);

typedef struct {
    TPMI_RH_HIERARCHY_AUTH authHandle;
    TPM2B_AUTH newAuth;
} HierarchyChangeAuth_In;
WOLFTPM_API TPM_RC TPM2_HierarchyChangeAuth(HierarchyChangeAuth_In* in);

typedef struct {
    TPMI_RH_LOCKOUT lockHandle;
} DictionaryAttackLockReset_In;
WOLFTPM_API TPM_RC
 ↪  TPM2_DictionaryAttackLockReset(DictionaryAttackLockReset_In* in);

typedef struct {
    TPMI_RH_LOCKOUT lockHandle;
    UINT32 newMaxTries;
    UINT32 newRecoveryTime;
    UINT32 lockoutRecovery;
} DictionaryAttackParameters_In;
WOLFTPM_API TPM_RC
 ↪  TPM2_DictionaryAttackParameters(DictionaryAttackParameters_In* in);


typedef struct {
    TPMI_RH_PLATFORM auth;
    TPML_CC setList;
    TPML_CC clearList;
} PP_Commands_In;
WOLFTPM_API TPM_RC TPM2_PP_Commands(PP_Commands_In* in);

typedef struct {
    TPMI_RH_PLATFORM authHandle;
    UINT32 algorithmSet;
} SetAlgorithmSet_In;
WOLFTPM_API TPM_RC TPM2_SetAlgorithmSet(SetAlgorithmSet_In* in);
```

```
typedef struct {
    TPMI_RH_PLATFORM authorization;
    TPMI_DH_OBJECT keyHandle;
    TPM2B_DIGEST fuDigest;
    TPMT_SIGNATURE manifestSignature;
} FieldUpgradeStart_In;
WOLFTPM_API TPM_RC TPM2_FieldUpgradeStart(FieldUpgradeStart_In* in);

typedef struct {
    TPM2B_MAX_BUFFER fuData;
} FieldUpgradeData_In;
typedef struct {
    TPMT_HA nextDigest;
    TPMT_HA firstDigest;
} FieldUpgradeData_Out;
WOLFTPM_API TPM_RC TPM2_FieldUpgradeData(FieldUpgradeData_In* in,
    FieldUpgradeData_Out* out);

typedef struct {
    UINT32 sequenceNumber;
} FirmwareRead_In;
typedef struct {
    TPM2B_MAX_BUFFER fuData;
} FirmwareRead_Out;
WOLFTPM_API TPM_RC TPM2_FirmwareRead(FirmwareRead_In* in, FirmwareRead_Out*
 ↪  out);


typedef struct {
    TPMI_DH_CONTEXT saveHandle;
} ContextSave_In;
typedef struct {
    TPMS_CONTEXT context;
} ContextSave_Out;
WOLFTPM_API TPM_RC TPM2_ContextSave(ContextSave_In* in, ContextSave_Out* out);

typedef struct {
    TPMS_CONTEXT context;
} ContextLoad_In;
typedef struct {
    TPMI_DH_CONTEXT loadedHandle;
} ContextLoad_Out;
WOLFTPM_API TPM_RC TPM2_ContextLoad(ContextLoad_In* in, ContextLoad_Out* out);


typedef struct {
    TPMI_RH_PROVISION auth;
    TPMI_DH_OBJECT objectHandle;
    TPMI_DH_PERSISTENT persistentHandle;
} EvictControl_In;
WOLFTPM_API TPM_RC TPM2_EvictControl(EvictControl_In* in);
```

```
typedef struct {
    TPMS_TIME_INFO currentTime;
} ReadClock_Out;
WOLFTPM_API TPM_RC TPM2_ReadClock(ReadClock_Out* out);

typedef struct {
    TPMI_RH_PROVISION auth;
    UINT64 newTime;
} ClockSet_In;
WOLFTPM_API TPM_RC TPM2_ClockSet(ClockSet_In* in);

typedef struct {
    TPMI_RH_PROVISION auth;
    TPM_CLOCK_ADJUST rateAdjust;
} ClockRateAdjust_In;
WOLFTPM_API TPM_RC TPM2_ClockRateAdjust(ClockRateAdjust_In* in);


typedef struct {
    TPMT_PUBLIC_PARMS parameters;
} TestParms_In;
WOLFTPM_API TPM_RC TPM2_TestParms(TestParms_In* in);


typedef struct {
    TPMI_RH_PROVISION authHandle;
    TPM2B_AUTH auth;
    TPM2B_NV_PUBLIC publicInfo;
} NV_DefineSpace_In;
WOLFTPM_API TPM_RC TPM2_NV_DefineSpace(NV_DefineSpace_In* in);

typedef struct {
    TPMI_RH_PROVISION authHandle;
    TPMI_RH_NV_INDEX nvIndex;
} NV_UndefineSpace_In;
WOLFTPM_API TPM_RC TPM2_NV_UndefineSpace(NV_UndefineSpace_In* in);

typedef struct {
    TPMI_RH_NV_INDEX nvIndex;
    TPMI_RH_PLATFORM platform;
} NV_UndefineSpaceSpecial_In;
WOLFTPM_API TPM_RC TPM2_NV_UndefineSpaceSpecial(NV_UndefineSpaceSpecial_In*
 ↪  in);

typedef struct {
    TPMI_RH_NV_INDEX nvIndex;
} NV_ReadPublic_In;
typedef struct {
    TPM2B_NV_PUBLIC nvPublic;
    TPM2B_NAME nvName;
} NV_ReadPublic_Out;
WOLFTPM_API TPM_RC TPM2_NV_ReadPublic(NV_ReadPublic_In* in, NV_ReadPublic_Out*
 ↪  out);
```

```
typedef struct {
    TPMI_RH_NV_AUTH authHandle;
    TPMI_RH_NV_INDEX nvIndex;
    TPM2B_MAX_NV_BUFFER data;
    UINT16 offset;
} NV_Write_In;
WOLFTPM_API TPM_RC TPM2_NV_Write(NV_Write_In* in);

typedef struct {
    TPMI_RH_NV_AUTH authHandle;
    TPMI_RH_NV_INDEX nvIndex;
} NV_Increment_In;
WOLFTPM_API TPM_RC TPM2_NV_Increment(NV_Increment_In* in);

typedef struct {
    TPMI_RH_NV_AUTH authHandle;
    TPMI_RH_NV_INDEX nvIndex;
    TPM2B_MAX_NV_BUFFER data;
} NV_Extend_In;
WOLFTPM_API TPM_RC TPM2_NV_Extend(NV_Extend_In* in);

typedef struct {
    TPMI_RH_NV_AUTH authHandle;
    TPMI_RH_NV_INDEX nvIndex;
    UINT64 bits;
} NV_SetBits_In;
WOLFTPM_API TPM_RC TPM2_NV_SetBits(NV_SetBits_In* in);

typedef struct {
    TPMI_RH_NV_AUTH authHandle;
    TPMI_RH_NV_INDEX nvIndex;
} NV_WriteLock_In;
WOLFTPM_API TPM_RC TPM2_NV_WriteLock(NV_WriteLock_In* in);

typedef struct {
    TPMI_RH_PROVISION authHandle;
} NV_GlobalWriteLock_In;
WOLFTPM_API TPM_RC TPM2_NV_GlobalWriteLock(NV_GlobalWriteLock_In* in);

typedef struct {
    TPMI_RH_NV_AUTH authHandle;
    TPMI_RH_NV_INDEX nvIndex;
    UINT16 size;
    UINT16 offset;
} NV_Read_In;
typedef struct {
    TPM2B_MAX_NV_BUFFER data;
} NV_Read_Out;
WOLFTPM_API TPM_RC TPM2_NV_Read(NV_Read_In* in, NV_Read_Out* out);

typedef struct {
    TPMI_RH_NV_AUTH authHandle;
    TPMI_RH_NV_INDEX nvIndex;
} NV_ReadLock_In;
```

```
WOLFTPM_API TPM_RC TPM2_NV_ReadLock(NV_ReadLock_In* in);

typedef struct {
    TPMI_RH_NV_INDEX nvIndex;
    TPM2B_AUTH newAuth;
} NV_ChangeAuth_In;
WOLFTPM_API TPM_RC TPM2_NV_ChangeAuth(NV_ChangeAuth_In* in);

typedef struct {
    TPMI_DH_OBJECT signHandle;
    TPMI_RH_NV_AUTH authHandle;
    TPMI_RH_NV_INDEX nvIndex;
    TPM2B_DATA qualifyingData;
    TPMT_SIG_SCHEME inScheme;
    UINT16 size;
    UINT16 offset;
} NV_Certify_In;
typedef struct {
    TPM2B_ATTEST certifyInfo;
    TPMT_SIGNATURE signature;
} NV_Certify_Out;
WOLFTPM_API TPM_RC TPM2_NV_Certify(NV_Certify_In* in, NV_Certify_Out* out);


/* Vendor Specific API's */
#if defined(WOLFTPM_ST33) || defined(WOLFTPM_AUTODETECT)
    /* Enable command code vendor API */
    typedef struct {
        TPMI_RH_HIERARCHY authHandle;
        TPM_CC commandCode;
        UINT32 enableFlag;
        UINT32 lockFlag;
    } SetCommandSet_In;
    WOLFTPM_API int TPM2_SetCommandSet(SetCommandSet_In* in);

    /* Mode bit-masks for STMicro ST33 */
    enum TPM_MODE_Vendor_Mask{
        TPMLib_2 = 0x01,
        TPMFips = 0x02,
        TPMLowPowerOff = 0x00,
        TPMLowPowerByRegister = 0x04,
        TPMLowPowerByGpio = 0x08,
        TPMLowPowerAuto = 0x0C,
    };
    typedef struct TPM_MODE_SET {
        BYTE CmdToLowPower;
        BYTE BootToLowPower;
        BYTE modeLock;
        BYTE mode;
    } TPM_MODE_SET;
    typedef struct {
        TPMI_RH_HIERARCHY authHandle;
        TPM_MODE_SET modeSet;
    } SetMode_In;
```

```
    WOLFTPM_API int TPM2_SetMode(SetMode_In* in);

    /* The TPM2_GetRandom2 command does not require any authorization */
    typedef GetRandom_In GetRandom2_In; /* same input */
    typedef struct {
        TPM2B_MAX_BUFFER randomBytes;
    } GetRandom2_Out;
    /* If bytesRequested is longer than TPM2B_MAX_BUFFER can accommodate, no
     * error is returned, but the TPM returns as much data as a TPM2B_DATA
     * buffer can contain. */
    WOLFTPM_API TPM_RC TPM2_GetRandom2(GetRandom2_In* in, GetRandom2_Out* out);

    WOLFTPM_API TPM_RC TPM2_GetProductInfo(uint8_t* info, uint16_t size);
#endif /* ST33 Vendor Specific */


#if defined(WOLFTPM_IFX_FW_UPGRADE) || defined(WOLFTPM_AUTODETECT)

#ifdef WOLFTPM_FIRMWARE_UPGRADE
WOLFTPM_API int TPM2_IFX_FieldUpgradeStart(TPM_HANDLE sessionHandle,
    uint8_t* data, uint32_t size);
WOLFTPM_API int TPM2_IFX_FieldUpgradeCommand(TPM_CC cc, uint8_t* data, uint32_t
 ↪  size);
#endif /* WOLFTPM_FIRMWARE_UPGRADE */

#endif /* Infineon SLB Vendor Specific */

#if defined(WOLFTPM_ST33) || defined(WOLFTPM_AUTODETECT)
#ifdef WOLFTPM_FIRMWARE_UPGRADE
WOLFTPM_API int TPM2_ST33_FieldUpgradeStart(TPM_HANDLE sessionHandle,
    uint8_t* data, uint32_t size);
/* Same as TPM2_ST33_FieldUpgradeStart, but with the field upgrade start
 * command code supplied by the caller. ST33 firmware differs on whether the
 * vendor code or the standard TPM_CC_FieldUpgradeStart is implemented. */
WOLFTPM_API int TPM2_ST33_FieldUpgradeStart_ex(TPM_HANDLE sessionHandle,
    TPM_CC cc, uint8_t* data, uint32_t size);
WOLFTPM_API int TPM2_ST33_FieldUpgradeCommand(TPM_CC cc, uint8_t* data,
 ↪  uint32_t size);
#endif /* WOLFTPM_FIRMWARE_UPGRADE */
#endif /* WOLFTPM_ST33 || WOLFTPM_AUTODETECT */


/* Vendor Specific GPIO */
#ifdef WOLFTPM_ST33

    #ifdef WOLFTPM_I2C
        #define MAX_GPIO_COUNT 4
    #else /* SPI variant */
        #define MAX_GPIO_COUNT 2
    #endif

    /* ST33 variants can have different count of GPIO available:
     * - SPI variant - 0, 1 or 2
     * - I2C variant - 0, 1, 2, 3 or 4
```

```
     * The user can configure this option at build or use default value. */
    #ifndef TPM_GPIO_COUNT
        #define TPM_GPIO_COUNT MAX_GPIO_COUNT
    #endif

    #define TPM_GPIO_NUM_MIN (TPM_GPIO_A)
    #define TPM_GPIO_NUM_MAX (TPM_GPIO_A + TPM_GPIO_COUNT - 1)

    /* GPIO configuration uses specific range of NV space */
    #define TPM_NV_GPIO_SPACE     0x01C40000

    typedef enum {
        TPM_GPIO_PP = 0x00000000, /* GPIO A by default is a Physical Presence
↪   pin */
        TPM_GPIO_LP = 0x00000001, /* GPIO B can only be used as an input */
    #ifdef WOLFTPM_I2C
        /* Only the I2C variant of ST33 has GPIO C and D */
        TPM_GPIO_C  = 0x00000002,
        TPM_GPIO_D  = 0x00000003,
    #endif
    } TPMI_GPIO_NAME_T;
    typedef UINT32 TPMI_GPIO_NAME;

    /* For portability and readability in code */
    #define TPM_GPIO_A TPM_GPIO_PP
    #define TPM_GPIO_B TPM_GPIO_LP

    typedef enum {
        TPM_GPIO_MODE_STANDARD   = 0x00000000,
        TPM_GPIO_MODE_FLOATING   = 0x00000001,
        TPM_GPIO_MODE_PULLUP     = 0x00000002,
        TPM_GPIO_MODE_PULLDOWN   = 0x00000003,
        TPM_GPIO_MODE_OPENDRAIN  = 0x00000004,
        TPM_GPIO_MODE_PUSHPULL   = 0x00000005,
        TPM_GPIO_MODE_UNCONFIG   = 0x00000006,
        TPM_GPIO_MODE_DEFAULT    = TPM_GPIO_MODE_PULLDOWN,
        TPM_GPIO_MODE_MAX        = TPM_GPIO_MODE_UNCONFIG,
        TPM_GPIO_MODE_INPUT_MIN  = TPM_GPIO_MODE_FLOATING,
        TPM_GPIO_MODE_INPUT_MAX  = TPM_GPIO_MODE_PULLDOWN
    } TPMI_GPIO_MODE_T;
    typedef UINT32 TPMI_GPIO_MODE;

    typedef struct TPMS_GPIO_CONFIG {
        TPMI_GPIO_NAME name;
        TPMI_RH_NV_INDEX index;
        TPMI_GPIO_MODE mode;
    } TPMS_GPIO_CONFIG;

    typedef struct TPML_GPIO_CONFIG {
        UINT32 count;
        TPMS_GPIO_CONFIG gpio[MAX_GPIO_COUNT];
    } TPML_GPIO_CONFIG;

    typedef struct {
```

```
        TPMI_RH_PLATFORM authHandle;
        TPML_GPIO_CONFIG config;
    } GpioConfig_In;
    WOLFTPM_API int TPM2_GPIO_Config(GpioConfig_In* in);

#elif defined(WOLFTPM_NUVOTON)

    #define MAX_GPIO_COUNT 2

    /* NPCT7XX supports a maximum of 2 GPIO for user control */
    /* Added in FW-US version 7.2.3.0 or later */
    #ifndef TPM_GPIO_COUNT
    #define TPM_GPIO_COUNT MAX_GPIO_COUNT
    #endif

    /* For portability */
    #undef  TPM_GPIO_A
    #define TPM_GPIO_A 3 /* NPCT75xx GPIO start at number 3 */

    #define TPM_GPIO_NUM_MIN (TPM_GPIO_A)
    #define TPM_GPIO_NUM_MAX (TPM_GPIO_A + TPM_GPIO_COUNT - 1)

    /* GPIO configuration uses specific range of NV space */
    #define TPM_NV_GPIO_SPACE     0x01C40003

    /* Nuvoton GPIO Modes */
    typedef enum {
        TPM_GPIO_MODE_PUSHPULL   = 1,
        TPM_GPIO_MODE_OPENDRAIN  = 2,
        TPM_GPIO_MODE_PULLUP     = 3,
        TPM_GPIO_MODE_UNCONFIG   = 4,
        TPM_GPIO_MODE_DEFAULT    = TPM_GPIO_MODE_PUSHPULL,
        TPM_GPIO_MODE_MAX        = TPM_GPIO_MODE_UNCONFIG,
        TPM_GPIO_MODE_INPUT_MIN  = TPM_GPIO_MODE_PULLUP,
        TPM_GPIO_MODE_INPUT_MAX  = TPM_GPIO_MODE_PULLUP
    } TPMI_GPIO_MODE_T;
    typedef UINT32 TPMI_GPIO_MODE;

    typedef struct {
        BYTE Base0;
        BYTE Base1;
        BYTE GpioAltCfg;
        BYTE GpioInitValue;
        BYTE GpioPullUp;
        BYTE GpioPushPull;
        BYTE Cfg_A;
        BYTE Cfg_B;
        BYTE Cfg_C;
        BYTE Cfg_D;
        BYTE Cfg_E;
        BYTE Cfg_F;
        BYTE Cfg_G;
        BYTE Cfg_H;
        BYTE Cfg_I;
```

```
        BYTE Cfg_J;
        BYTE isValid;
        BYTE isLocked;
    } CFG_STRUCT;

    typedef struct {
        TPMI_RH_PLATFORM authHandle;
        CFG_STRUCT preConfig;
    } NTC2_PreConfig_In;
    WOLFTPM_API int TPM2_NTC2_PreConfig(NTC2_PreConfig_In* in);

    typedef struct {
        CFG_STRUCT preConfig;
    } NTC2_GetConfig_Out;
    WOLFTPM_API int TPM2_NTC2_GetConfig(NTC2_GetConfig_Out* out);
#endif /* Vendor GPIO Commands */

/* NTC2 PreConfig/GetConfig for WOLFTPM_AUTODETECT (runtime vendor detection).
 * When a specific vendor is not selected at compile time, the NTC2 types
 * must still be available for SPDM enable via NTC2_PreConfig. */
#if defined(WOLFTPM_AUTODETECT) && !defined(WOLFTPM_NUVOTON) && \
    !defined(WOLFTPM_ST33)
    typedef struct {
        BYTE Base0;
        BYTE Base1;
        BYTE GpioAltCfg;
        BYTE GpioInitValue;
        BYTE GpioPullUp;
        BYTE GpioPushPull;
        BYTE Cfg_A;
        BYTE Cfg_B;
        BYTE Cfg_C;
        BYTE Cfg_D;
        BYTE Cfg_E;
        BYTE Cfg_F;
        BYTE Cfg_G;
        BYTE Cfg_H;
        BYTE Cfg_I;
        BYTE Cfg_J;
        BYTE isValid;
        BYTE isLocked;
    } CFG_STRUCT;

    typedef struct {
        TPMI_RH_PLATFORM authHandle;
        CFG_STRUCT preConfig;
    } NTC2_PreConfig_In;
    WOLFTPM_API int TPM2_NTC2_PreConfig(NTC2_PreConfig_In* in);

    typedef struct {
        CFG_STRUCT preConfig;
    } NTC2_GetConfig_Out;
    WOLFTPM_API int TPM2_NTC2_GetConfig(NTC2_GetConfig_Out* out);
#endif /* WOLFTPM_AUTODETECT && !WOLFTPM_NUVOTON && !WOLFTPM_ST33 */
```

```
#ifdef WOLFTPM_NATIONS
    /* Nations Technology NS350 Vendor Commands */
    typedef struct {
        TPMI_RH_PLATFORM authHandle;
        UINT32 configuration;  /* 1 = set (provision), 0 = unset */
    } Nations_IdentityKeySet_In;
    WOLFTPM_API int TPM2_Nations_IdentityKeySet(Nations_IdentityKeySet_In* in);
#endif /* WOLFTPM_NATIONS */


/* Non-standard API's */

#define _TPM_Init TPM2_Init
WOLFTPM_API TPM_RC TPM2_Init(TPM2_CTX* ctx, TPM2HalIoCb ioCb, void* userCtx);

WOLFTPM_API TPM_RC TPM2_Init_ex(TPM2_CTX* ctx, TPM2HalIoCb ioCb, void* userCtx,
    int timeoutTries);

WOLFTPM_API TPM_RC TPM2_Init_minimal(TPM2_CTX* ctx);

WOLFTPM_API TPM_RC TPM2_Cleanup(TPM2_CTX* ctx);

/* Other API's - Not in TPM Specification */

WOLFTPM_API TPM_RC TPM2_ChipStartup(TPM2_CTX* ctx, int timeoutTries);

WOLFTPM_API TPM_RC TPM2_SetHalIoCb(TPM2_CTX* ctx, TPM2HalIoCb ioCb, void*
 ↪  userCtx);

#ifndef WOLFTPM_NO_RETRY
WOLFTPM_API TPM_RC TPM2_SetCommandRetries(TPM2_CTX* ctx, int retries);

WOLFTPM_API int TPM2_GetCommandRetries(TPM2_CTX* ctx);
#endif /* !WOLFTPM_NO_RETRY */

WOLFTPM_API TPM_RC TPM2_SetSessionAuth(TPM2_AUTH_SESSION *session);

WOLFTPM_API int    TPM2_GetSessionAuthCount(TPM2_CTX* ctx);

WOLFTPM_API void      TPM2_SetActiveCtx(TPM2_CTX* ctx);

WOLFTPM_API TPM2_CTX* TPM2_GetActiveCtx(void);

WOLFTPM_API int TPM2_GetHashDigestSize(TPMI_ALG_HASH hashAlg);

WOLFTPM_API int TPM2_GetHashType(TPMI_ALG_HASH hashAlg);

WOLFTPM_API TPMI_ALG_HASH TPM2_GetTpmHashType(int hashType);

WOLFTPM_API int TPM2_GetNonce(byte* nonceBuf, int nonceSz);

/* Internal API for getting nonce without taking lock */
WOLFTPM_LOCAL int TPM2_GetNonceNoLock(byte* nonceBuf, int nonceSz);
```

```
WOLFTPM_API void TPM2_SetupPCRSel(TPML_PCR_SELECTION* pcr, TPM_ALG_ID alg,
    int pcrIndex);

WOLFTPM_API void TPM2_SetupPCRSelArray(TPML_PCR_SELECTION* pcr, TPM_ALG_ID alg,
    byte* pcrArray, word32 pcrArraySz);

WOLFTPM_API int TPM2_IsAlgSupported(TPM_ALG_ID alg, int* isSupported);

WOLFTPM_API int TPM2_IsPcrBankAllocated(TPM_ALG_ID hashAlg, int pcrIndex,
    int* isAllocated);

WOLFTPM_API const char* TPM2_GetRCString(int rc);

WOLFTPM_API const char* TPM2_GetAlgName(TPM_ALG_ID alg);

WOLFTPM_API TPM_ALG_ID TPM2_GetAlgId(const char* name);

#ifdef DEBUG_WOLFTPM
WOLFTPM_API const char* TPM2_GetHierarchyDesc(TPMI_RH_HIERARCHY_AUTH
 ↪  authHandle);
#endif

WOLFTPM_API int TPM2_GetCurveSize(TPM_ECC_CURVE curveID);

WOLFTPM_API TPM_ALG_ID TPM2_GetCurveHashAlg(TPM_ECC_CURVE curveID);

WOLFTPM_API int TPM2_GetTpmCurve(int curveID);

WOLFTPM_API int TPM2_GetWolfCurve(int curve_id);

WOLFTPM_API int TPM2_ParseAttest(const TPM2B_ATTEST* in, TPMS_ATTEST* out);

WOLFTPM_API int TPM2_HashNvPublic(TPMS_NV_PUBLIC* nvPublic, byte* buffer,
 ↪  UINT16* size);

WOLFTPM_API int TPM2_AppendPublic(byte* buf, word32 size, int* sizeUsed,
 ↪  TPM2B_PUBLIC* pub);

WOLFTPM_API int TPM2_ParsePublic(TPM2B_PUBLIC* pub, byte* buf, word32 size,
 ↪  int* sizeUsed);

WOLFTPM_LOCAL int TPM2_GetName(TPM2_CTX* ctx, UINT32 handleValue, int
 ↪  handleCnt, int idx, TPM2B_NAME* name);

#ifdef WOLFTPM2_USE_WOLF_RNG
WOLFTPM_API int TPM2_GetWolfRng(WC_RNG** rng);
#endif

typedef enum {
    TPM_VENDOR_UNKNOWN = 0,
    TPM_VENDOR_INFINEON = 0x15d1,
    TPM_VENDOR_STM = 0x104a,
    TPM_VENDOR_MCHP = 0x1114,
```

```
    TPM_VENDOR_NUVOTON = 0x1050,
    TPM_VENDOR_NATIONTECH = 0x1B4E,
    TPM_VENDOR_SEALSQ = 0x2406,
} TPM_Vendor_t;


WOLFTPM_API UINT16 TPM2_GetVendorID(void);


WOLFTPM_API const char* TPM2_GetCapsFipsStr(int fips140_3, int fips140_2);


WOLFTPM_LOCAL void TPM2_ForceZero(void* mem, word32 len);

WOLFTPM_TEST_API int TPM2_ConstantCompare(const byte* a, const byte* b, word32
 ↪  len);


#ifdef DEBUG_WOLFTPM
WOLFTPM_API void TPM2_PrintBin(const byte* buffer, word32 length);

WOLFTPM_API void TPM2_PrintAuth(const TPMS_AUTH_COMMAND* authCmd);

WOLFTPM_API void TPM2_PrintPublicArea(const TPM2B_PUBLIC* pub);
#else
#define TPM2_PrintBin(b, l)
#define TPM2_PrintAuth(b)
#define TPM2_PrintPublicArea(b)
#endif

#ifdef __cplusplus
    }  /* extern "C" */
#endif

#endif /* __TPM2_H__ */
```

29.9 wolftpm/tpm2_wrap.h

29.9.1 Classes

	Name
struct	**WOLFTPM2_SESSION**
struct	**WOLFTPM2_DEV**
struct	**WOLFTPM2_KEY**
struct	**WOLFTPM2_PKEY**
struct	**WOLFTPM2_KEYBLOB**
struct	**WOLFTPM2_HASH**
struct	**WOLFTPM2_NV**
struct	**WOLFTPM2_HMAC**
struct	**WOLFTPM2_CSR**
struct	**TpmSignCbCtx** Context structure for TPM-based certificate signing callback.
struct	**WOLFTPM2_BUFFER**

	Name
struct	**WOLFTPM2_CAPS**
struct	**TpmCryptoDevCtx**

29.9.2 Types

	Name
enum	**WOLFTPM2_MFG** { TPM_MFG_UNKNOWN = 0, TPM_MFG_INFINEON, TPM_MFG_STM, TPM_MFG_MCHP, TPM_MFG_NUVOTON, TPM_MFG_NATIONTECH, TPM_MFG_SEALSQ, TPM_MFG_MSFT}
typedef struct WOLFTPM2_SESSION**	
typedef struct WOLFTPM2_DEV**	
typedef struct WOLFTPM2_KEY**	
typedef struct WOLFTPM2_PKEY**	
typedef struct WOLFTPM2_KEYBLOB**	
typedef struct WOLFTPM2_HASH**	
typedef struct WOLFTPM2_NV**	
typedef struct WOLFTPM2_HMAC**	
typedef struct WOLFTPM2_CSR**	
typedef struct TpmSignCbCtx** Context structure for TPM-based certificate signing callback.	
typedef struct WOLFTPM2_BUFFER**	
typedef enum WOLFTPM2_MFG**	
typedef struct WOLFTPM2_CAPS**	
typedef struct TpmCryptoDevCtx**	
typedef int*()(uint8_t* data, uint32_t data_req_sz, uint32_t offset, void *cb_ctx)	**wolfTPM2FwDataCb**

29.9.3 Functions

	Name
WOLFTPM_API int	**wolfTPM2_Test * caps)Test initialization of a TPM and optionally the TPM capabilities can be received.
WOLFTPM_API int	**wolfTPM2_Init ioCb, void * userCtx)Complete initialization of a TPM On failure after argument validation, the device is fully torn down, except for TPM_RC_UPGRADE, which leaves an active context for the vendor firmware-recovery flow.

	Name
WOLFTPM_API int	**wolfTPM2_InitWithSpdmKey ioCb, void * userCtx, const byte * rspPubKey, word32 rspPubKeySz)Initialize a TPM and pin a trusted SPDM responder public key. A successful call establishes an authenticated SPDM session even when the TPM accepted the initial cleartext startup probe. When the TPM is already in SPDM-only identity mode, startup is retried over that secure session. The identity mode is chosen for you: the one vendor adapter the build carries, generic TCG when it carries none, and otherwise from the TPM's DID/VID. That last case needs a transport that exposes DID/VID; use wolfTPM2_InitWithSpdmKey_ex to name the mode when it does not. On failure after argument validation, the device is fully torn down, except for TPM_RC_UPGRADE, which leaves an active context for the vendor firmware-recovery flow.
WOLFTPM_API int	**wolfTPM2_InitWithSpdmKey_ex ioCb, void * userCtx, const byte * rspPubKey, word32 rspPubKeySz, WOLFSPDM_MODE mode)Initialize with a trusted responder key and explicit identity mode. Use this form when the build carries more than one identity adapter and the transport does not expose TPM DID/VID. WOLFSPDM_MODE_TCG is the generic TCG identity flow and needs no vendor adapter; a vendor mode is accepted only when its adapter is compiled in, and BAD_FUNC_ARG is returned before the device is touched when it is not. WOLFSPDM_MODE_AUTO resolves from DID/VID and fails closed rather than guessing a vendor handshake. TPM_RC_UPGRADE leaves an active context for the vendor firmware-recovery flow; call cleanup when that flow is complete.
WOLFTPM_TEST_API int	**wolfTPM2_SpdmModeFromDidVid**(UINT32 didVid, WOLFSPDM_MODE * mode)
WOLFTPM_API int	**wolfTPM2_InitWithSpdmPsk ioCb, void * userCtx, const byte * psk, word32 pskSz, const byte * hint, word32 hintSz)Initialize a TPM and authenticate an SPDM session with a PSK. This entry point also recovers a PSK-configured TPM that is already locked in SPDM-only mode by retrying TPM2_Startup over the secure session. On failure after argument validation, the device is fully torn down, except for TPM_RC_UPGRADE, which leaves an active context for the vendor firmware-recovery flow.

	Name
WOLFTPM_API int	**wolfTPM2_OpenExisting ioCb, void * userCtx)Use an already initialized TPM, in its current TPM locality.
WOLFTPM_API int	**wolfTPM2_Cleanup * dev)Easy to use TPM and wolfcrypt deinitialization.
WOLFTPM_API int	**wolfTPM2_Cleanup_ex * dev, int doShutdown)Deinitialization of a TPM (and wolfcrypt if it was used)
WOLFTPM_API int	**wolfTPM2_GetTpmDevId * dev)Provides the device ID of a TPM.
WOLFTPM_API int	**wolfTPM2_SelfTest * dev)Asks the TPM to perform its self test.
WOLFTPM_API int	**wolfTPM2_GetCapabilities * caps)Reports the available TPM capabilities.
WOLFTPM_TEST_API int	**wolfTPM2_ParseCapabilities.
WOLFTPM_API int	**wolfTPM2_IsAlgSupported alg, int * isSupported)Report whether the TPM implements a given algorithm.
WOLFTPM_API int	**wolfTPM2_GetHandles * handles)Gets a list of handles.
WOLFTPM_API int	**wolfTPM2_SpdmInit * dev)Initialize SPDM support on a wolfTPM2 device. Allocates and configures the SPDM context using wolfSPDM. After init, call wolfTPM2_SpdmConnect to establish a secure session.
WOLFTPM_API int	**wolfTPM2_SpdmConnect * dev)Establish an SPDM secure session (full handshake). Uses standard SPDM flow: GET_VERSION -> GET_CAPABILITIES -> NEGOTIATE_ALGORITHMS -> KEY_EXCHANGE -> FINISH. Identity-key modes require a responder key pinned with wolfTPM2_SpdmSetResponderPubKey before this call.
WOLFTPM_API int	**wolfTPM2_SpdmIsConnected * dev)Check if an SPDM secure session is currently active.
WOLFTPM_API word32	**wolfTPM2_SpdmGetSessionId * dev)Get the current SPDM session ID.
WOLFTPM_API int	**wolfTPM2_GetCapability_SPDMSessionInfo * spdmSessionInfo)Read the TPM's view of the active SPDM session (GetCapability TPM_CAP_SPDM_SESSION_INFO, TPM 2.0 Library v1.84).
WOLFTPM_API int	**wolfTPM2_SpdmDisconnect * dev)Disconnect the SPDM secure session. After this, TPM commands are sent in the clear.
WOLFTPM_API int	**wolfTPM2_SpdmGetVersion * dev)Send the SPDM GET_VERSION handshake step. Required before any vendor-defined SPDM command (e.g. PSK provisioning, GET_STATUS).
WOLFTPM_API int	**wolfTPM2_SpdmCleanup * dev)Free SPDM context and resources.

	Name
WOLFTPM_API int	**wolfTPM2_SpdmSetResponderPubKey * dev, const byte * pubKey, word32 pubKeySz)Pin the trusted responder key before an identity-key connection.
WOLFTPM_API int	**wolfTPM2_SpdmGetPubKey * dev, byte * pubKey, word32 * pubKeySz)Get the TPM's SPDM-Identity public key (shared TCG function).
WOLFTPM_API int	**wolfTPM2_SpdmSetNuvotonMode for Nuvoton TPMs.
WOLFTPM_API int	**wolfTPM2_SpdmEnable * dev)Enable SPDM on the TPM via NTC2_PreConfig vendor command. Requires platform hierarchy auth. TPM must be reset after this call.
WOLFTPM_API int	**wolfTPM2_SpdmDisable * dev)Disable SPDM on Nuvoton TPM via NTC2_PreConfig. Sets Cfg_H bit 1 to disable SPDM. Requires TPM reset to take effect.
WOLFTPM_API int	**wolfTPM2_SpdmConnectNuvoton * dev, const byte * reqPubKey, word32 reqPubKeySz, const byte * reqPrivKey, word32 reqPrivKeySz)Establish Nuvoton SPDM secure session with mutual authentication. Uses Nuvoton flow: GET_VERSION -> GET_PUB_KEY -> KEY_EXCHANGE -> GIVE_PUB_KEY -> FINISH. Requires a responder key pinned with wolfTPM2_SpdmSetResponderPubKey.
WOLFTPM_API int	**wolfTPM2_SpdmGetStatus * dev, WOLFSPDM_NUVOTON_STATUS * status)Get SPDM status from the TPM (GET_STS_ vendor command).
WOLFTPM_API int	**wolfTPM2_SpdmSetOnlyMode * dev, int lock)Lock or unlock SPDM-only mode. When locked, TPM only accepts commands over SPDM secure channel.
WOLFTPM_API int	**wolfTPM2_SpdmSetNationsMode.
WOLFTPM_API int	**wolfTPM2_SpdmConnectNations * dev, const byte * reqPubKey, word32 reqPubKeySz, const byte * reqPrivKey, word32 reqPrivKeySz)Establish Nations SPDM secure session (identity key mode). Uses TCG flow: GET_VERSION -> GET_PUB_KEY -> KEY_EXCHANGE -> GIVE_PUB_KEY -> FINISH. Requires a responder key pinned with wolfTPM2_SpdmSetResponderPubKey.
WOLFTPM_API int	**wolfTPM2_SpdmNationsIdentityKeySet * dev, int set)Set/unset SPDM identity key provisioning on Nations NS350. Uses vendor command TPM2_VendorSpdmIdentityKeySet (0x20000708).

	Name
WOLFTPM_API int	**wolfTPM2_SpdmNationsGetStatus * dev, WOLFSPDM_NATIONS_STATUS * status)Get SPDM status from Nations TPM (PSK mode).
WOLFTPM_API int	**wolfTPM2_SpdmNationsSetOnlyMode * dev, int lock)Lock or unlock SPDM-only mode on Nations TPM (PSK mode).
WOLFTPM_API int	**wolfTPM2_SpdmNationsPskSet * dev, const byte * psk, word32 pskSz)Provision PSK on Nations TPM.
WOLFTPM_API int	**wolfTPM2_SpdmNationsPskClear * dev, const byte * clearAuth, word32 clearAuthSz)Clear PSK from Nations TPM. Requires ClearAuth from PSK_SET.
WOLFTPM_API int	**wolfTPM2_SpdmSetInfineonMode * dev)Configure for Infineon SLB9678 SPDM (PSK mode, Algorithm Set B). Must be called before PSK provisioning vendor commands.
WOLFTPM_API int	**wolfTPM2_SpdmConnectInfineonTcg * dev, const byte * reqPubKey, word32 reqPubKeySz, const byte * reqPrivKey, word32 reqPrivKeySz)Establish an SLB9678 asymmetric session using the shared TCG flow. Requires firmware advertising asymmetric SPDM and a responder key pinned with wolfTPM2_SpdmSetResponderPubKey. No PSK fallback is attempted.
WOLFTPM_API int	**wolfTPM2_SpdmInfineonGetStatus * dev, WOLFSPDM_INFINEON_STATUS * status)Get SPDM status from Infineon SLB9678 (PSK mode).
WOLFTPM_API int	**wolfTPM2_SpdmInfineonSetOnlyMode * dev, int lock)Lock or unlock SPDM-only mode on Infineon SLB9678.
WOLFTPM_API int	**wolfTPM2_SpdmInfineonEnable * dev, int enable, int lock)Enable or disable the SLB9678 SPDM-PSK secure channel via TPM2_SetCapability(TPM_CAP_CONFIGS, TPM_PT_CONFIG_SPDM_PSK). Requires Platform authorization configured in session slot zero. An unused slot uses empty-password authorization. Pass lock=0 to keep it reversible (enable is undone by disabling); lock=1 lifetime-locks the configuration.
WOLFTPM_API int	**wolfTPM2_SpdmConnectPsk * dev, const byte * psk, word32 pskSz, const byte * hint, word32 hintSz)Establish SPDM secure session via the DSP0274 PSK handshake (PSK_EXCHANGE / PSK_FINISH). Vendor-neutral spec implementation.
WOLFTPM_API int	**wolfTPM2_UnsetAuth * dev, int index)Clears one of the TPM Authorization slots, pointed by its index number.

	Name
WOLFTPM_API int	**wolfTPM2_UnsetAuthSession * session)Clears one of the TPM Authorization session slots, pointed by its index number and saves the nonce from the TPM so the session can continue to be used again with wolfTPM2_SetAuthSession.
WOLFTPM_API int	**wolfTPM2_SetAuth * name)Sets a TPM Authorization slot using the provided index, session handle, attributes and auth.
WOLFTPM_API int	**wolfTPM2_SetAuthPassword * auth)Sets a TPM Authorization slot using the provided user auth, typically a password.
WOLFTPM_API int	**wolfTPM2_SetAuthHandle * dev, int index, const WOLFTPM2_HANDLE * handle)Sets a TPM Authorization slot using the user auth associated with a wolfTPM2 Handle.
WOLFTPM_API int	**wolfTPM2_SetAuthSession sessionAttributes)Sets a TPM Authorization slot using the provided TPM session handle, index and session attributes.
WOLFTPM_API int	**wolfTPM2_SetSessionHandle * tpmSession)Sets a TPM Authorization slot using the provided wolfTPM2 session object.
WOLFTPM_API int	**wolfTPM2_SetAuthHandleName * dev, int index, const WOLFTPM2_HANDLE * handle)Updates the Name used in a TPM Session with the Name associated with wolfTPM2 Handle.
WOLFTPM_API int	**wolfTPM2_StartSession sesType, int encDecAlg)Create a TPM session, Policy, HMAC or Trial.
WOLFTPM_API int	**wolfTPM2_StartSession_ex authHash)Create a TPM session selecting the session auth/hash algorithm.
WOLFTPM_API int	**wolfTPM2_CreateAuthSession_EkPolicy * tpmSession)Creates a TPM session with Policy Secret to satisfy the default EK policy.
WOLFTPM_API int	**wolfTPM2_CreateAuthSession_EkPolicy_ex authHash)Creates a TPM EK policy session selecting the session auth/hash algorithm.
WOLFTPM_API int	**wolfTPM2_CreatePrimaryKey * publicTemplate, const byte * auth, int authSz)Single function to prepare and create a TPM 2.0 Primary Key.
WOLFTPM_API int	**wolfTPM2_CreatePrimaryKey_ex * publicTemplate, const byte * auth, int authSz)Single function to prepare and create a TPM 2.0 Primary Key.
WOLFTPM_API int	**wolfTPM2_ChangeAuthKey * key, WOLFTPM2_HANDLE * parent, const byte * auth, int authSz)Change the authorization secret of a TPM 2.0 key.

	Name
WOLFTPM_API int	**wolfTPM2_CreateKey * publicTemplate, const byte * auth, int authSz)Single function to prepare and create a TPM 2.0 Key.
WOLFTPM_API int	**wolfTPM2_LoadKey * keyBlob, WOLFTPM2_HANDLE * parent)Single function to load a TPM 2.0 key.
WOLFTPM_API int	**wolfTPM2_CreateAndLoadKey * publicTemplate, const byte * auth, int authSz)Single function to create and load a TPM 2.0 Key in one step.
WOLFTPM_API int	**wolfTPM2_CreateLoadedKey * publicTemplate, const byte * auth, int authSz)Creates and loads a key using single TPM 2.0 operation, and stores encrypted private key material.
WOLFTPM_API int	**wolfTPM2_LoadPublicKey * pub)Wrapper to load the public part of an external key.
WOLFTPM_API int	**wolfTPM2_LoadPublicKey_ex hierarchy)
WOLFTPM_API int	**wolfTPM2_LoadPrivateKey * sens)Single function to import an external private key and load it into the TPM in one step.
WOLFTPM_API int	**wolfTPM2_ImportPrivateKey * sens)Single function to import an external private key and load it into the TPM in one step.
WOLFTPM_API int	**wolfTPM2_LoadRsaPublicKey * key, const byte * rsaPub, word32 rsaPubSz, word32 exponent)Helper function to import the public part of an external RSA key.
WOLFTPM_API int	**wolfTPM2_LoadRsaPublicKey_ex hashAlg)Advanced helper function to import the public part of an external RSA key.
WOLFTPM_API int	**wolfTPM2_ImportRsaPrivateKey hashAlg)Import an external RSA private key.
WOLFTPM_API int	**wolfTPM2_ImportRsaPrivateKeySeed attributes, byte * seed, word32 seedSz)Import an external RSA private key with custom seed.
WOLFTPM_API int	**wolfTPM2_LoadRsaPrivateKey * key, const byte * rsaPub, word32 rsaPubSz, word32 exponent, const byte * rsaPriv, word32 rsaPrivSz)Helper function to import and load an external RSA private key in one step.
WOLFTPM_API int	**wolfTPM2_LoadRsaPrivateKey_ex hashAlg)Advanced helper function to import and load an external RSA private key in one step.
WOLFTPM_API int	**wolfTPM2_LoadEccPublicKey * key, int curveId, const byte * eccPubX, word32 eccPubXSz, const byte * eccPubY, word32 eccPubYSz)Helper function to import the public part of an external ECC key.

	Name
WOLFTPM_API int	**wolfTPM2_LoadEccPublicKey_ex objectAttributes)Variant of wolfTPM2_LoadEccPublicKey that lets the caller select the signing scheme, hash algorithm, and object attributes - required for loading ECDH peer keys (decrypt attribute) or non-default scheme/hash combinations.
WOLFTPM_API int	**wolfTPM2_ImportEccPrivateKey * keyBlob, int curveId, const byte * eccPubX, word32 eccPubXSz, const byte * eccPubY, word32 eccPubYSz, const byte * eccPriv, word32 eccPrivSz)Helper function to import the private material of an external ECC key.
WOLFTPM_API int	**wolfTPM2_ImportEccPrivateKeySeed attributes, byte * seed, word32 seedSz)Helper function to import the private material of an external ECC key.
WOLFTPM_API int	**wolfTPM2_LoadEccPrivateKey * key, int curveId, const byte * eccPubX, word32 eccPubXSz, const byte * eccPubY, word32 eccPubYSz, const byte * eccPriv, word32 eccPrivSz)Helper function to import and load an external ECC private key in one step.
WOLFTPM_API int	**wolfTPM2_ReadPublicKey handle)Helper function to receive the public part of a loaded TPM object using its handle.
WOLFTPM_API int	**wolfTPM2_CreateKeySeal * publicTemplate, const byte * auth, int authSz, const byte * sealData, int sealSize)Using this wrapper a secret can be sealed inside a TPM 2.0 Key.
WOLFTPM_API int	**wolfTPM2_CreateKeySeal_ex pcrAlg, byte * pcrArray, word32 pcrArraySz, const byte * sealData, int sealSize)Using this wrapper a secret can be sealed inside a TPM 2.0 Key with pcr selection.
WOLFTPM_API int	**wolfTPM2_ComputeName * out)Helper function to generate a hash of the public area of an object in the format expected by the TPM.
WOLFTPM_API int	**wolfTPM2_SensitiveToPrivate.
WOLFTPM_API word32	**wolfTPM2_RsaKey_Exponent**(const byte * e, word32 eSz)Converts a big-endian byte array to a native word32 value. Used for RSA exponent conversion from ASN.1/DER format.
WOLFTPM_API int	**wolfTPM2_ImportPrivateKeyBuffer objectAttributes, byte * seed, word32 seedSz)Helper function to import PEM/DER or RSA/ECC private key.
WOLFTPM_API int	**wolfTPM2_ImportPublicKeyBuffer objectAttributes)Helper function to import PEM/DER formatted RSA/ECC public key.

	Name
WOLFTPM_API int	**wolfTPM2_ExportPublicKeyBuffer * tpmKey, int encodingType, byte * out, word32 * outSz)Helper function to export a TPM RSA/ECC public key with PEM/DER formatting.
WOLFTPM_API int	**wolfTPM2_RsaPrivateKeyImportDer hashAlg)Helper function to import Der rsa key directly.
WOLFTPM_API int	**wolfTPM2_RsaPrivateKeyImportPem hashAlg)Helper function to import Pem rsa key directly.
WOLFTPM_API int	**wolfTPM2_RsaKey_TpmToWolf * tpmKey, RsaKey * wolfKey)Extract a RSA TPM key and convert it to a wolfcrypt key.
WOLFTPM_API int	**wolfTPM2_RsaKey_TpmToPemPub * keyBlob, byte * pem, word32 * pemSz)Convert a public RSA TPM key to PEM format public key. Note: This API is a wrapper around wolfTPM2_ExportPublicKeyBuffer.
WOLFTPM_API int	**wolfTPM2_RsaKey_WolfToTpm * tpmKey)Import a RSA wolfcrypt key into the TPM.
WOLFTPM_API int	**wolfTPM2_RsaKey_WolfToTpm_ex * tpmKey)Import a RSA wolfcrypt key into the TPM under a specific Primary Key or Hierarchy.
WOLFTPM_API int	**wolfTPM2_CreateRsaKeyBlob * tpmKey)Create an encrypted RSA key blob from a wolfCrypt key under a specific parent key.
WOLFTPM_API int	**wolfTPM2_RsaKey_PubPemToTpm * tpmKey, const byte * pem, word32 pemSz)Import a PEM format public key from a file into the TPM.
WOLFTPM_API int	**wolfTPM2_DecodeRsaDer attributes)Import DER RSA private or public key into TPM public and sensitive structures. This does not make any calls to TPM hardware.
WOLFTPM_API int	**wolfTPM2_EccKey_TpmToWolf * tpmKey, ecc_key * wolfKey)Extract a ECC TPM key and convert to to a wolfcrypt key.
WOLFTPM_API int	**wolfTPM2_EccKey_WolfToTpm * tpmKey)Import a ECC wolfcrypt key into the TPM.
WOLFTPM_API int	**wolfTPM2_EccKey_WolfToTpm_ex * tpmKey)Import ECC wolfcrypt key into the TPM under a specific Primary Key or Hierarchy.
WOLFTPM_API int	**wolfTPM2_CreateEccKeyBlob * tpmKey)Create an encrypted ECC key blob from a wolfCrypt key under a specific parent key.
WOLFTPM_API int	**wolfTPM2_EccKey_WolfToPubPoint * pubPoint)Import a ECC public key generated from wolfcrypt key into the TPM.

	Name
WOLFTPM_API int	**wolfTPM2_DecodeEccDer attributes)Import DER ECC private or public key into TPM public and sensitive structures. This does not make any calls to TPM hardware.
WOLFTPM_API int	**wolfTPM2_SignHash * key, const byte * digest, int digestSz, byte * sig, int * sigSz)Helper function to sign arbitrary data using a TPM key.
WOLFTPM_API int	**wolfTPM2_SignHashScheme hashAlg)Advanced helper function to sign arbitrary data using a TPM key, and specify the signature scheme and hashing algorithm.
WOLFTPM_API int	**wolfTPM2_VerifyHash * key, const byte * sig, int sigSz, const byte * digest, int digestSz)Helper function to verify a TPM generated signature.
WOLFTPM_API int	**wolfTPM2_VerifyHash_ex * key, const byte * sig, int sigSz, const byte * digest, int digestSz, int hashAlg)Helper function to verify a TPM generated signature.
WOLFTPM_API int	**wolfTPM2_VerifyHashScheme hashAlg)Advanced helper function to verify a TPM generated signature.
WOLFTPM_API int	**wolfTPM2_VerifyHashTicket * checkTicket)Advanced helper function to verify a TPM generated signature and return ticket.
WOLFTPM_API int	**wolfTPM2_SignSequenceStart * sequenceHandle)Post-Quantum Cryptography: Start a signing sequence with context support.
WOLFTPM_API int	**wolfTPM2_SignSequenceUpdate sequenceHandle, const byte * data, int dataSz)Post-Quantum Cryptography: Update a signing sequence with data.
WOLFTPM_API int	**wolfTPM2_SignSequenceComplete * key, const byte * data, int dataSz, byte * sig, int * sigSz)Post-Quantum Cryptography: Complete a signing sequence.
WOLFTPM_API int	**wolfTPM2_VerifySequenceStart * sequenceHandle)Post-Quantum Cryptography: Start a verification sequence with context support.
WOLFTPM_API int	**wolfTPM2_VerifySequenceUpdate sequenceHandle, const byte * data, int dataSz)Post-Quantum Cryptography: Update a verification sequence with data.
WOLFTPM_API int	**wolfTPM2_VerifySequenceComplete * validation)Post-Quantum Cryptography: Complete a verification sequence.

	Name
WOLFTPM_API int	**wolfTPM2_SignDigest * key, const byte * digest, int digestSz, const byte * context, int contextSz, byte * sig, int * sigSz)Post-Quantum Cryptography: Sign a digest with context support.
WOLFTPM_API int	**wolfTPM2_VerifyDigestSignature * validation)Post-Quantum Cryptography: Verify a digest signature with context support.
WOLFTPM_API int	**wolfTPM2_Encapsulate * key, byte * ciphertext, int * ciphertextSz, byte * sharedSecret, int * sharedSecretSz)Post-Quantum Cryptography: KEM Encapsulate (public key operation)
WOLFTPM_API int	**wolfTPM2_Decapsulate * key, const byte * ciphertext, int ciphertextSz, byte * sharedSecret, int * sharedSecretSz)Post-Quantum Cryptography: KEM Decapsulate (private key operation)
WOLFTPM_API int	**wolfTPM2_GetKeyTemplate_MLDSA parameterSet, int allowExternalMu)Create a key template for ML-DSA signing keys (TCG v185)
WOLFTPM_API int	**wolfTPM2_GetKeyTemplate_MLDSA_ex nameAlg)Create an ML-DSA key template selecting the name algorithm (TCG v185)
WOLFTPM_API int	**wolfTPM2_GetKeyTemplate_HASH_MLDSA hashAlg)Create a key template for Pre-Hash ML-DSA signing keys (TCG v185)
WOLFTPM_API int	**wolfTPM2_GetKeyTemplate_HASH_MLDSA_ex nameAlg)Create a Pre-Hash ML-DSA key template selecting the name algorithm (TCG v185)
WOLFTPM_API int	**wolfTPM2_GetKeyTemplate_MLKEM parameterSet)Create a key template for ML-KEM decryption keys (TCG v185)
WOLFTPM_API int	**wolfTPM2_GetKeyTemplate_MLKEM_ex nameAlg)Create an ML-KEM key template selecting the name algorithm (TCG v185)
WOLFTPM_API int	**wolfTPM2_ECDHGenKey * ecdhKey, int curve_id, const byte * auth, int authSz)Generates and then loads a ECC key-pair with NULL hierarchy for Diffie-Hellman exchange.
WOLFTPM_API int	**wolfTPM2_ECDHGen * pubPoint, byte * out, int * outSz)Generates ephemeral key and computes Z (shared secret)
WOLFTPM_API int	**wolfTPM2_ECDHGenZ * pubPoint, byte * out, int * outSz)Computes Z (shared secret) using pubPoint and loaded private ECC key.
WOLFTPM_API int	**wolfTPM2_ECDHEGenKey * ecdhKey, int curve_id)Generates ephemeral ECC key and returns array index (2 phase method)

	Name
WOLFTPM_API int	**wolfTPM2_ECDHEGenZ * pubPoint, byte * out, int * outSz)Computes Z (shared secret) using pubPoint and counter (2 phase method)
WOLFTPM_API int	**wolfTPM2_RsaEncrypt padScheme, const byte * msg, int msgSz, byte * out, int * outSz)Perform RSA encryption using a TPM 2.0 key.
WOLFTPM_API int	**wolfTPM2_RsaEncrypt_ex hashAlg)Perform RSA encryption selecting the padding scheme hash algorithm.
WOLFTPM_API int	**wolfTPM2_RsaDecrypt padScheme, const byte * in, int inSz, byte * msg, int * msgSz)Perform RSA decryption using a TPM 2.0 key.
WOLFTPM_API int	**wolfTPM2_RsaDecrypt_ex hashAlg)Perform RSA decryption selecting the padding scheme hash algorithm.
WOLFTPM_API int	**wolfTPM2_ReadPCR * dev, int pcrIndex, int hashAlg, byte * digest, int * pDigestLen)Read the values of a specified TPM 2.0 Platform Configuration Registers(PCR)
WOLFTPM_API int	**wolfTPM2_ResetPCR * dev, int pcrIndex)Reset a PCR register to its default value.
WOLFTPM_API int	**wolfTPM2_SetLocality * dev, int locality)Change the active TPM locality at runtime.
WOLFTPM_API int	**wolfTPM2_ExtendPCR * dev, int pcrIndex, int hashAlg, const byte * digest, int digestLen)Extend a PCR register with a user provided digest.
WOLFTPM_API int	**wolfTPM2_AllocatePCRBanks * allocOut)Re-provision which PCR banks the TPM has allocated.
WOLFTPM_API int	**wolfTPM2_AllocatePCRBanks_ex * allocOut)Re-provision the TPM's PCR banks using a caller supplied session.
WOLFTPM_API int	**wolfTPM2_NVCreateAuth * nv, word32 nvIndex, word32 nvAttributes, word32 maxSize, const byte * auth, int authSz)Creates a new NV Index to be later used for storing data into the TPM's NVRAM.
WOLFTPM_API int	**wolfTPM2_NVCreateAuthPolicy * nv, word32 nvIndex, word32 nvAttributes, word32 maxSize, const byte * auth, int authSz, const byte * authPolicy, int authPolicySz)Creates a new NV Index to be later used for storing data into the TPM's NVRAM.

	Name
WOLFTPM_API int	**wolfTPM2_NVCreateAuthPolicy_ex nameAlg)Variant of wolfTPM2_NVCreateAuthPolicy that lets the caller pass an explicit nameAlg. Required when authPolicy was computed with SM3-256, SHA3-256, or any other non-SHA hash that the size-based legacy inference cannot distinguish from SHA-256.
WOLFTPM_API int	**wolfTPM2_NVWriteAuth * nv, word32 nvIndex, byte * dataBuf, word32 dataSz, word32 offset)Stores user data to a NV Index, at a given offset.
WOLFTPM_API int	**wolfTPM2_NVWriteAuthPolicy * nv, word32 nvIndex, byte * dataBuf, word32 dataSz, word32 offset)Stores user data to a NV Index, at a given offset. Allows using a policy session and PCR's for authentication.
WOLFTPM_API int	**wolfTPM2_NVExtend * nv, word32 nvIndex, byte * dataBuf, word32 dataSz)Extend data to an NV index.
WOLFTPM_API int	**wolfTPM2_NVReadAuth * nv, word32 nvIndex, byte * dataBuf, word32 * pDataSz, word32 offset)Reads user data from a NV Index, starting at the given offset.
WOLFTPM_API int	**wolfTPM2_NVReadAuthPolicy * nv, word32 nvIndex, byte * dataBuf, word32 * pDataSz, word32 offset)Reads user data from a NV Index, starting at the given offset. Allows using a policy session and PCR's for authentication.
WOLFTPM_API int	**wolfTPM2_NVReadCert handle, uint8_t * buffer, uint32_t * len)Helper to get size of NV and read buffer without authentication. Typically used for reading a certificate from an NV.
WOLFTPM_API int	**wolfTPM2_NVIncrement * nv)Increments an NV one-way counter.
WOLFTPM_API int	**wolfTPM2_NVOpen * nv, word32 nvIndex, const byte * auth, word32 authSz)Open an NV and populate the required authentication and name hash.
WOLFTPM_API int	**wolfTPM2_NVWriteLock * nv)Lock writes on the specified NV Index.
WOLFTPM_API int	**wolfTPM2_NVDeleteAuth * dev, WOLFTPM2_HANDLE * parent, word32 nvIndex)Destroys an existing NV Index.
WOLFTPM_API int	**wolfTPM2_NVCreate authHandle, word32 nvIndex, word32 nvAttributes, word32 maxSize, const byte * auth, int authSz)Deprecated, use newer API.
WOLFTPM_API int	**wolfTPM2_NVWrite authHandle, word32 nvIndex, byte * dataBuf, word32 dataSz, word32 offset)Deprecated, use newer API.

	Name
WOLFTPM_API int	**wolfTPM2_NVRead authHandle, word32 nvIndex, byte * dataBuf, word32 * dataSz, word32 offset)Deprecated, use newer API.
WOLFTPM_API int	**wolfTPM2_NVDelete authHandle, word32 nvIndex)Deprecated, use newer API.
WOLFTPM_API int	**wolfTPM2_NVReadPublic * nvPublic)Extracts the public information about an nvIndex, such as maximum size.
WOLFTPM_API int	**wolfTPM2_NVStoreKey persistentHandle)Helper function to store a TPM 2.0 Key into the TPM's NVRAM.
WOLFTPM_API int	**wolfTPM2_NVDeleteKey * key)Helper function to delete a TPM 2.0 Key from the TPM's NVRAM.
WOLFTPM_API struct WC_RNG *	**wolfTPM2_GetRng * dev)Get the wolfcrypt RNG instance used for wolfTPM.
WOLFTPM_API int	**wolfTPM2_GetRandom * dev, byte * buf, word32 len)Get a set of random number, generated with the TPM RNG or wolfcrypt RNG.
WOLFTPM_API int	**wolfTPM2_UnloadHandle * dev, WOLFTPM2_HANDLE * handle)Use to discard any TPM loaded object.
WOLFTPM_API int	**wolfTPM2_Clear * dev)Clears the TPM with TPM2_Clear, authorized by the lockout hierarchy. This removes the owner and endorsement hierarchy objects and the NV indices created by the owner, and resets their authorizations. NV indices created by the platform (TPMA_NV_PLATFORMCREATE) are preserved. It does not deinitialize wolfTPM; use wolfTPM2_Cleanup for that.
WOLFTPM_API int	**wolfTPM2_DictionaryAttackLockReset * dev)Reset the dictionary attack lockout failed-tries counter.
WOLFTPM_API int	**wolfTPM2_DictionaryAttackParameters * dev, word32 newMaxTries, word32 newRecoveryTime, word32 lockoutRecovery)Set the dictionary attack lockout parameters.
WOLFTPM_API int	**wolfTPM2_HashStart hashAlg, const byte * usageAuth, word32 usageAuthSz)Helper function to start a TPM generated hash.
WOLFTPM_API int	**wolfTPM2_HashUpdate * hash, const byte * data, word32 dataSz)Update a TPM generated hash with new user data.
WOLFTPM_API int	**wolfTPM2_HashFinish * hash, byte * digest, word32 * digestSz)Finalize a TPM generated hash and get the digest output in a user buffer.

	Name
WOLFTPM_API int	**wolfTPM2_LoadKeyedHashKey * key, WOLFTPM2_HANDLE * parent, int hashAlg, const byte * keyBuf, word32 keySz, const byte * usageAuth, word32 usageAuthSz)Creates and loads a new TPM key of KeyedHash type, typically used for HMAC operations.
WOLFTPM_API int	**wolfTPM2_HmacStart hashAlg, const byte * keyBuf, word32 keySz, const byte * usageAuth, word32 usageAuthSz)Helper function to start a TPM generated hmac.
WOLFTPM_API int	**wolfTPM2_HmacUpdate * hmac, const byte * data, word32 dataSz)Update a TPM generated hmac with new user data.
WOLFTPM_API int	**wolfTPM2_HmacFinish * hmac, byte * digest, word32 * digestSz)Finalize a TPM generated hmac and get the digest output in a user buffer.
WOLFTPM_API int	**wolfTPM2_LoadSymmetricKey * key, int alg, const byte * keyBuf, word32 keySz)Loads an external symmetric key into the TPM.
WOLFTPM_API int	**wolfTPM2_EncryptDecryptBlock * key, const byte * in, byte * out, word32 inOutSz, byte * iv, word32 ivSz, int isDecrypt)Encrypt or decrypt a single block of data with the symmetric key.
WOLFTPM_API int	**wolfTPM2_EncryptDecrypt**(WOLFTPM2_DEV * key, const byte * in, byte * out, word32 inOutSz, byte * iv, word32 ivSz, int isDecrypt)
WOLFTPM_API int	**wolfTPM2_SetCommand commandCode, int enableFlag)Vendor specific TPM command, used to enable other restricted TPM commands.
WOLFTPM_API int	**wolfTPM2_Reset * dev, int doShutdown, int doStartup)Helper function to shutdown, startup or reset the TPM.
WOLFTPM_API int	**wolfTPM2_Shutdown * dev, int doStartup)Helper function to shutdown or reset the TPM.
WOLFTPM_API int	**wolfTPM2_UnloadHandles * dev, word32 handleStart, word32 handleCount)One-shot API to unload subsequent TPM handles.
WOLFTPM_API int	**wolfTPM2_UnloadHandles_AllTransient * dev)One-shot API to unload all transient TPM handles.
WOLFTPM_API int	**wolfTPM2_GetKeyTemplate_RSA objectAttributes)Prepares a TPM public template for new RSA key based on user selected object attributes.
WOLFTPM_API int	**wolfTPM2_GetKeyTemplate_RSA_ex sigHash)Prepares a TPM public template for new RSA key based on user selected object attributes.

	Name
WOLFTPM_API int	**wolfTPM2_GetKeyTemplate_ECC sigScheme)Prepares a TPM public template for new ECC key based on user selected object attributes.
WOLFTPM_API int	**wolfTPM2_GetKeyTemplate_ECC_ex sigHash)Prepares a TPM public template for new ECC key based on user selected object attributes.
WOLFTPM_API int	**wolfTPM2_GetKeyTemplate_Symmetric algMode, int isSign, int isDecrypt)Prepares a TPM public template for new Symmetric key.
WOLFTPM_API int	**wolfTPM2_GetKeyTemplate_KeyedHash hashAlg, int isSign, int isDecrypt)Prepares a TPM public template for new KeyedHash key.
WOLFTPM_API int	**wolfTPM2_GetKeyTemplate_KeySeal nameAlg)Prepares a TPM public template for new key for sealing secrets.
WOLFTPM_API int	**wolfTPM2_GetKeyTemplate_EK nameAlg, int highRange)Prepares a TPM public template for generating the TPM Endorsement Key.
WOLFTPM_API int	**wolfTPM2_GetKeyTemplate_EKIndex * publicTemplate)Helper to get the Endorsement public key template by NV index.
WOLFTPM_API int	**wolfTPM2_GetKeyTemplate_RSA_EK * publicTemplate)Prepares a TPM public template for generating the TPM Endorsement Key of RSA type.
WOLFTPM_API int	**wolfTPM2_GetKeyTemplate_ECC_EK * publicTemplate)Prepares a TPM public template for generating the TPM Endorsement Key of ECC type.
WOLFTPM_API int	**wolfTPM2_GetKeyTemplate_RSA_SRK * publicTemplate)Prepares a TPM public template for generating a new TPM Storage Key of RSA type.
WOLFTPM_API int	**wolfTPM2_GetKeyTemplate_ECC_SRK * publicTemplate)Prepares a TPM public template for generating a new TPM Storage Key of ECC type.
WOLFTPM_API int	**wolfTPM2_GetKeyTemplate_RSA_AIK * publicTemplate)Prepares a TPM public template for generating a new TPM Attestation Key of RSA type.
WOLFTPM_API int	**wolfTPM2_GetKeyTemplate_ECC_AIK * publicTemplate)Prepares a TPM public template for generating a new TPM Attestation Key of ECC type.
WOLFTPM_API int	**wolfTPM2_GetKeyTemplate_RSA_IAK hashAlg)
WOLFTPM_API int	**wolfTPM2_GetKeyTemplate_ECC_IAK hashAlg)

	Name
WOLFTPM_API int	**wolfTPM2_GetKeyTemplate_ECC_IDevID hashAlg)
WOLFTPM_API int	**wolfTPM2_GetKeyTemplate_RSA_IDevID hashAlg)
WOLFTPM_API int	**wolfTPM2_SetKeyTemplate_Unique * publicTemplate, const byte * unique, int uniqueSz)Sets the unique area of a public template used by Create or CreatePrimary.
WOLFTPM_API int	**wolfTPM2_GetNvAttributesTemplate auth, word32 * nvAttributes)Prepares a TPM NV Index template.
WOLFTPM_API int	**wolfTPM2_CreateEK alg)Generates a new TPM Endorsement key, based on the user selected algorithm, RSA or ECC.
WOLFTPM_API int	**wolfTPM2_CreateSRK alg, const byte * auth, int authSz)Generates a new TPM Primary Key that will be used as a Storage Key for other TPM keys.
WOLFTPM_API int	**wolfTPM2_CreateAndLoadAIK * srkKey, const byte * auth, int authSz)Generates a new TPM Attestation Key under the provided Storage Key.
WOLFTPM_API int	**wolfTPM2_GetTime * getTimeOut)One-shot API to generate a TPM signed timestamp.
WOLFTPM_API int	**wolfTPM2_CSR_SetCustomExt structure.
WOLFTPM_API int	**wolfTPM2_CSR_SetKeyUsage structure. Pass either extended key usage or key usage values. Mixed string types are not supported, however you can call `wolfTPM2_CSR_SetKeyUsage` twice (once for extended key usage strings and once for standard key usage strings).
WOLFTPM_API int	**wolfTPM2_CSR_SetSubject structure.
WOLFTPM_API int	**wolfTPM2_CSR_MakeAndSign_ex structure with subject and key usage already set.
WOLFTPM_API int	**wolfTPM2_CSR_MakeAndSign structure with subject and key usage already set.
WOLFTPM_API int	**wolfTPM2_CSR_Generate_ex). Single shot API for outputting a CSR or self-signed cert based on TPM key.
WOLFTPM_API int	**wolfTPM2_CSR_Generate). Single shot API for outputting a CSR or self-signed cert based on TPM key.
WOLFTPM_API int	**wolfTPM2_ChangePlatformAuth * session)Helper to set the platform hierarchy authentication value to random. Setting the platform auth to random value is used to prevent application from being able to use platform hierarchy. This is defined in section 10 of the TCG PC Client Platform specification.

	Name
WOLFTPM_API int	**wolfTPM2_ChangeHierarchyAuth authHandle)Helper to set the hierarchy authentication value to random. Setting the platform auth to random value is used to prevent application from being able to use platform hierarchy. This is defined in section 10 of the TCG PC Client Platform specification.
WOLFTPM_API int	**wolfTPM2_EncryptSecret**(WOLFTPM2_DEV * encSecret, const char * label)
WOLFTPM_API int	**wolfTPM2_CryptoDevCb**(int devId, wc_CryptoInfo * info, void * ctx)A reference crypto callback API for using the TPM for crypto offload. This callback function is registered using wolfTPM2_SetCryptoDevCb or wc_CryptoDev_RegisterDevice.
WOLFTPM_API int	**wolfTPM2_SetCryptoDevCb * tpmCtx, int * pDevId)Register a crypto callback function and return assigned devId.
WOLFTPM_API int	**wolfTPM2_ClearCryptoDevCb * dev, int devId)Clears the registered crypto callback.
WOLFTPM_API int	**wolfTPM2_PK_RsaSign**(WOLFSSL * ssl, const unsigned char * in, unsigned int inSz, unsigned char * out, word32 * outSz, const unsigned char * keyDer, unsigned int keySz, void * ctx)
WOLFTPM_API int	**wolfTPM2_PK_RsaSignCheck**(WOLFSSL * ssl, unsigned char * sig, unsigned int sigSz, unsigned char ** out, const unsigned char * keyDer, unsigned int keySz, void * ctx)
WOLFTPM_API int	**wolfTPM2_PK_RsaPssSign**(WOLFSSL * ssl, const unsigned char * in, unsigned int inSz, unsigned char * out, unsigned int * outSz, int hash, int mgf, const unsigned char * keyDer, unsigned int keySz, void * ctx)
WOLFTPM_API int	**wolfTPM2_PK_RsaPssSignCheck**(WOLFSSL * ssl, unsigned char * sig, unsigned int sigSz, unsigned char ** out, int hash, int mgf, const unsigned char * keyDer, unsigned int keySz, void * ctx)
WOLFTPM_API int	**wolfTPM2_PK_EccSign**(WOLFSSL * ssl, const unsigned char * in, unsigned int inSz, unsigned char * out, word32 * outSz, const unsigned char * keyDer, unsigned int keySz, void * ctx)
WOLFTPM_API int	**wolfTPM_PK_SetCb**(WOLFSSL_CTX * ctx)
WOLFTPM_API int	**wolfTPM_PK_SetCbCtx**(WOLFSSL * ssl, void * userCtx)
WOLFTPM_API WOLFTPM2_DEV.	
WOLFTPM_API int	**wolfTPM2_Free that was allocated by wolfTPM2_New.
WOLFTPM_API WOLFTPM2_KEYBLOB.	
WOLFTPM_API int	**wolfTPM2_FreeKeyBlob that was allocated with wolfTPM2_NewKeyBlob.
WOLFTPM_API TPMT_PUBLIC.	

	Name
WOLFTPM_API int	**wolfTPM2_FreePublicTemplate that was allocated with wolfTPM2_NewPublicTemplate.
WOLFTPM_API WOLFTPM2_KEY.	
WOLFTPM_API int	**wolfTPM2_FreeKey that was allocated with wolfTPM2_NewKey.
WOLFTPM_API WOLFTPM2_SESSION.	
WOLFTPM_API int	**wolfTPM2_FreeSession that was allocated with wolfTPM2_NewSession.
WOLFTPM_API WOLFTPM2_CSR.	
WOLFTPM_API int	**wolfTPM2_FreeCSR that was allocated with wolfTPM2_NewCSR.
WOLFTPM_API WOLFTPM2_HANDLE *	**wolfTPM2_GetHandleRefFromKey.
WOLFTPM_API WOLFTPM2_HANDLE *	**wolfTPM2_GetHandleRefFromKeyBlob.
WOLFTPM_API WOLFTPM2_HANDLE *	**wolfTPM2_GetHandleRefFromSession.
WOLFTPM_API TPM_HANDLE	**wolfTPM2_GetHandleValue**(WOLFTPM2_HANDLE * handle)Get the 32-bit handle value from the WOLFTPM2_HANDLE.
WOLFTPM_API int	**wolfTPM2_SetKeyAuthPassword * key, const byte * auth, int authSz)Set the authentication data for a key.
WOLFTPM_API int	**wolfTPM2_GetKeyBlobAsBuffer * key)Marshal data from a keyblob to a binary buffer. This can be stored to disk for loading in a separate process or after power cycling. If buffer is not provided then size only will be returned.
WOLFTPM_API int	**wolfTPM2_GetKeyBlobAsSeparateBuffers * key)Marshal data from a keyblob to a binary buffer. This can be stored to disk for loading in a separate process or after power cycling. If either buffer is NULL then the size will be returned for each part.
WOLFTPM_API int	**wolfTPM2_SetKeyBlobFromBuffer struct. This can be used to load a keyblob that was previously marshaled by wolfTPM2_GetKeyBlobAsBuffer.
WOLFTPM_API int	**wolfTPM2_PolicyRestart sessionHandle)Restart the policy digest for a policy session.
WOLFTPM_API int	**wolfTPM2_GetPolicyDigest sessionHandle, byte * policyDigest, word32 * policyDigestSz)Get the policy digest of the session that was passed in wolfTPM2_GetPolicyDigest.
WOLFTPM_API int	**wolfTPM2_PolicyPCR pcrAlg, byte * pcrArray, word32 pcrArraySz)Apply the PCR's to the policy digest for the policy session.

	Name
WOLFTPM_API int	**wolfTPM2_PolicyTransportSPDM * tpmKeyName)Add TPM2_PolicyTransportSPDM (TPM 2.0 Library v1.84) to a policy session. An entity whose authPolicy contains this term can only be authorized by a command that arrives inside an SPDM session. Optional key names further restrict which SPDM session qualifies.
WOLFTPM_API int	**wolfTPM2_PolicyAuthorize * checkTicket, const byte * pcrDigest, word32 pcrDigestSz, const byte * policyRef, word32 policyRefSz)Apply the PCR's to the policy digest for the policy session.
WOLFTPM_API int	**wolfTPM2_PCRGetDigest pcrAlg, byte * pcrArray, word32 pcrArraySz, byte * pcrDigest, word32 * pcrDigestSz)Get a cumulative digest of the PCR's specified.
WOLFTPM_API int	**wolfTPM2_PolicyRefMake pcrAlg, byte * digest, word32 * digestSz, const byte * policyRef, word32 policyRefSz)Utility for generating a policy ref digest. If no policy reference (nonce) used then just rehash the provided digest again (update -> final)
WOLFTPM_API int	**wolfTPM2_PolicyPCRMake pcrAlg, byte * pcrArray, word32 pcrArraySz, const byte * pcrDigest, word32 pcrDigestSz, byte * digest, word32 * digestSz)Utility for generating a policy PCR digest.
WOLFTPM_API int	**wolfTPM2_PolicyTransportSPDMMake * tpmKeyName, byte * digest, word32 * digestSz)Compute the policy digest that TPM2_PolicyTransportSPDM produces, without a TPM. Use it to build the authPolicy of an NV index or key that must only be reachable over an SPDM session.
WOLFTPM_API int	**wolfTPM2_PolicyCommandCodeMake cc)Compute the policy digest for PolicyCommandCode on a fresh session.
WOLFTPM_API int	**wolfTPM2_PolicyHash cc, const byte * input, word32 inputSz)Utility for creating a policy hash. Generic helper that takes command code and input array. policyDigestnew = hash(policyDigestOld
WOLFTPM_API int	**wolfTPM2_PolicyAuthorizeMake * pub, byte * digest, word32 * digestSz, const byte * policyRef, word32 policyRefSz)Utility for generating a policy authorization digest based on a public key.

	Name
WOLFTPM_API int	**wolfTPM2_PolicyPassword * tpmSession, const byte * auth, int authSz)Wrapper for setting a policy password and calling TPM2_PolicyPassword. This will set a password (in clear) for the policy session instead of HMAC.
WOLFTPM_API int	**wolfTPM2_PolicyAuthValue * tpmSession, const byte * auth, int authSz)Wrapper for setting a policy auth value that is added to the HMAC key for a policy session.
WOLFTPM_API int	**wolfTPM2_PolicyCommandCode cc)Wrapper for setting a policy command code.
WOLFTPM_API int	**wolfTPM2_PolicyOR * pHashList)Wrapper for satisfying a policy session with a compound OR of digests.
WOLFTPM_API int	**wolfTPM2_SetPrimaryPolicy hashAlg, const byte * authPolicy, word32 authPolicySz)Set (or clear) the authPolicy of a hierarchy.
WOLFTPM_API int	**wolfTPM2_SetIdentityAuth * dev, WOLFTPM2_HANDLE * handle, uint8_t * masterPassword, uint16_t masterPasswordSz)Set authentication for pre-provisioned identity keys.
WOLFTPM_LOCAL int	**GetKeyTemplateRSA sigHash)Internal helper to create RSA key template.
WOLFTPM_LOCAL int	**GetKeyTemplateECC sigHash)Internal helper to create ECC key template.
WOLFTPM_TEST_API int	**wolfTPM2_EccZToBuffer**(byte * out, int * outSz, const TPM2B_ECC_PARAMETER * z)
WOLFTPM_API int	**wolfTPM2_FirmwareUpgradeHash cb, void * cb_ctx)Perform TPM firmware upgrade with pre-computed manifest hash.
WOLFTPM_API int	**wolfTPM2_FirmwareUpgradeHash_ex * startSession)Perform TPM firmware upgrade using a caller-supplied authorization session.
WOLFTPM_API int	**wolfTPM2_FirmwareUpgrade cb, void * cb_ctx)Perform TPM firmware upgrade.
WOLFTPM_API int	**wolfTPM2_FirmwareUpgrade_ex * startSession)Perform TPM firmware upgrade using a caller-supplied authorization session.
WOLFTPM_API int	**wolfTPM2_FirmwareUpgradeRecover cb, void * cb_ctx)Recover from failed TPM firmware upgrade.
WOLFTPM_API int	**wolfTPM2_FirmwareUpgradeRecover_ex * startSession)Recover from a failed firmware upgrade using a caller-supplied session.
WOLFTPM_API int	**wolfTPM2_FirmwareUpgradeCancel * dev)Cancel ongoing TPM firmware upgrade.

	Name
WOLFTPM_API int	**wolfTPM2_ST33_ManifestVersion**(const uint8_t * manifest, uint32_t manifest_sz, word16 * major, word16 * minor)Read the firmware version an ST33 firmware image upgrades to.
WOLFTPM_API int	**wolfTPM2_ST33_CmdImplemented cc, int * isImpl)Report whether the TPM implements a command code.
WOLFTPM_API int	**wolfTPM2_ST33_GetFwUpgradeCommands * ccData, int * fromTpm)Choose the ST33 field upgrade command codes for the attached TPM.
WOLFTPM_TEST_API int	**wolfTPM2_ST33_FwUpgradeCommands * ccData)Choose the ST33 field upgrade command codes from the version fields.

29.9.4 Attributes

Name
C

29.9.5 Types Documentation

29.9.5.1 enum WOLFTPM2_MFG

Enumerator	Value	Description
TPM_MFG_UNKNOWN	0	
TPM_MFG_INFINEON		
TPM_MFG_STM		
TPM_MFG_MCHP		
TPM_MFG_NUVOTON		
TPM_MFG_NATIONTECH		
TPM_MFG_SEALSQ		
TPM_MFG_MSFT		

29.9.5.2 typedef WOLFTPM2_SESSION

```
typedef struct WOLFTPM2_SESSION WOLFTPM2_SESSION;
```

29.9.5.3 typedef WOLFTPM2_DEV

```
typedef struct WOLFTPM2_DEV WOLFTPM2_DEV;
```

29.9.5.4 typedef WOLFTPM2_KEY

```
typedef struct WOLFTPM2_KEY WOLFTPM2_KEY;
```

29.9.5.5 typedef WOLFTPM2_PKEY

```
typedef struct WOLFTPM2_PKEY WOLFTPM2_PKEY;
```

29.9.5.6 typedef WOLFTPM2_KEYBLOB

```
typedef struct WOLFTPM2_KEYBLOB WOLFTPM2_KEYBLOB;
```

29.9.5.7 typedef WOLFTPM2_HASH

```
typedef struct WOLFTPM2_HASH WOLFTPM2_HASH;
```

29.9.5.8 typedef WOLFTPM2_NV

```
typedef struct WOLFTPM2_NV WOLFTPM2_NV;
```

29.9.5.9 typedef WOLFTPM2_HMAC

```
typedef struct WOLFTPM2_HMAC WOLFTPM2_HMAC;
```

29.9.5.10 typedef WOLFTPM2_CSR

```
typedef struct WOLFTPM2_CSR WOLFTPM2_CSR;
```

29.9.5.11 typedef TpmSignCbCtx

```
typedef struct TpmSignCbCtx TpmSignCbCtx;
```

Context structure for TPM-based certificate signing callback.

See:

- wolfTPM2_CSR_MakeAndSign_ex
- wc_SignCert_cb

This structure holds the TPM device and key references needed for the certificate signing callback (wc_SignCertCb). It is used internally by wolfTPM2_CSR_MakeAndSign_ex when using the callback-based signing approach.

29.9.5.12 typedef WOLFTPM2_BUFFER

```
typedef struct WOLFTPM2_BUFFER WOLFTPM2_BUFFER;
```

29.9.5.13 typedef WOLFTPM2_MFG

```
typedef enum WOLFTPM2_MFG WOLFTPM2_MFG;
```

29.9.5.14 typedef WOLFTPM2_CAPS

```
typedef struct WOLFTPM2_CAPS WOLFTPM2_CAPS;
```

29.9.5.15 typedef TpmCryptoDevCtx

```
typedef struct TpmCryptoDevCtx TpmCryptoDevCtx;
```

29.9.5.16 typedef wolfTPM2FwDataCb

```
typedef int(* wolfTPM2FwDataCb) (uint8_t *data, uint32_t data_req_sz, uint32_t
 ↪  offset, void *cb_ctx);
```

29.9.6 Functions Documentation

29.9.6.1 function wolfTPM2_Test

```
WOLFTPM_API int wolfTPM2_Test(
    TPM2HalIoCb ioCb,
    void * userCtx,
    WOLFTPM2_CAPS * caps
)
```

Test initialization of a TPM and optionally the TPM capabilities can be received.

Parameters:

- **ioCb** function pointer to a IO callback (see [hal#file-tpm-io.h))
- **userCtx** pointer to a user context (can be NULL)
- **caps** to a structure of WOLFTPM2_CAPS type for returning the TPM capabilities (can be NULL)

See:

- wolfTPM2_Init
- TPM2_Init

Return:

- TPM_RC_SUCCESS: successful
- TPM_RC_FAILURE: generic failure (check TPM IO and TPM return code)
- BAD_FUNC_ARG: check the provided arguments

29.9.6.2 function wolfTPM2_Init

```
WOLFTPM_API int wolfTPM2_Init(
    WOLFTPM2_DEV * dev,
    TPM2HalIoCb ioCb,
    void * userCtx
)
```

Complete initialization of a TPM On failure after argument validation, the device is fully torn down, except for TPM_RC_UPGRADE, which leaves an active context for the vendor firmware-recovery flow.

Parameters:

- **dev** pointer to an empty structure of WOLFTPM2_DEV type
- **ioCb** function pointer to a IO callback (see [hal#file-tpm-io.h))
- **userCtx** pointer to a user context (can be NULL)

See:

- wolfTPM2_OpenExisting
- wolfTPM2_InitWithSpdmKey
- wolfTPM2_InitWithSpdmPsk
- wolfTPM2_Test
- TPM2_Init

Return:

- TPM_RC_SUCCESS: successful

- TPM_RC_FAILURE: generic failure (check TPM IO communication)
- TPM_RC_UPGRADE: TPM firmware recovery is active; call cleanup when the recovery flow is complete
- BAD_FUNC_ARG: check the provided arguments
- WOLFSPDM_E_BAD_STATE: TPM is in SPDM-only mode and no trusted identity key or PSK was supplied; use the matching SPDM init API

Example

```
int rc;
WOLFTPM2_DEV dev;

rc = wolfTPM2_Init(&dev, TPM2_IoCb, userCtx);
if (rc != TPM_RC_SUCCESS) {
    //wolfTPM2_Init failed
    goto exit;
}
```

29.9.6.3 function wolfTPM2_InitWithSpdmKey

```
WOLFTPM_API int wolfTPM2_InitWithSpdmKey(
    WOLFTPM2_DEV * dev,
    TPM2HalIoCb ioCb,
    void * userCtx,
    const byte * rspPubKey,
    word32 rspPubKeySz
)
```

Initialize a TPM and pin a trusted SPDM responder public key. A successful call establishes an authenticated SPDM session even when the TPM accepted the initial cleartext startup probe. When the TPM is already in SPDM-only identity mode, startup is retried over that secure session. The identity mode is chosen for you: the one vendor adapter the build carries, generic TCG when it carries none, and otherwise from the TPM's DID/VID. That last case needs a transport that exposes DID/VID; use wolfTPM2_InitWithSpdmKey_ex to name the mode when it does not. On failure after argument validation, the device is fully torn down, except for TPM_RC_UPGRADE, which leaves an active context for the vendor firmware-recovery flow.

Parameters:

- **dev** pointer to an empty WOLFTPM2_DEV structure
- **ioCb** function pointer to an I/O callback
- **userCtx** pointer to a user context (can be NULL)
- **rspPubKey** trusted raw P-384 responder key in X||Y format
- **rspPubKeySz** size of rspPubKey; must be 96 bytes

See:

- wolfTPM2_Init
- wolfTPM2_SpdmSetResponderPubKey
- wolfTPM2_SpdmConnect
- wolfTPM2_SpdmIsConnected

Return:

- TPM_RC_SUCCESS: successful
- TPM_RC_UPGRADE: TPM firmware recovery is active; call cleanup when the recovery flow is complete
- BAD_FUNC_ARG: invalid parameters or responder key size

- WOLFSPDM_E_BAD_STATE: SPDM identity mode cannot be initialized, or the DID/VID matches no compiled adapter
- a WOLFSPDM_E_* error if the authenticated handshake fails

29.9.6.4 function wolfTPM2_InitWithSpdmKey_ex

```
WOLFTPM_API int wolfTPM2_InitWithSpdmKey_ex(
    WOLFTPM2_DEV * dev,
    TPM2HalIoCb ioCb,
    void * userCtx,
    const byte * rspPubKey,
    word32 rspPubKeySz,
    WOLFSPDM_MODE mode
)
```

Initialize with a trusted responder key and explicit identity mode. Use this form when the build carries more than one identity adapter and the transport does not expose TPM DID/VID. WOLFSPDM_MODE_TCG is the generic TCG identity flow and needs no vendor adapter; a vendor mode is accepted only when its adapter is compiled in, and BAD_FUNC_ARG is returned before the device is touched when it is not. WOLFSPDM_MODE_AUTO resolves from DID/VID and fails closed rather than guessing a vendor handshake. TPM_RC_UPGRADE leaves an active context for the vendor firmware-recovery flow; call cleanup when that flow is complete.

Parameters:

- **dev** pointer to an empty WOLFTPM2_DEV structure
- **ioCb** function pointer to an I/O callback
- **userCtx** pointer to a user context (can be NULL)
- **rspPubKey** trusted raw P-384 responder key in X||Y format
- **rspPubKeySz** size of rspPubKey; must be 96 bytes
- **mode** WOLFSPDM_MODE_AUTO, WOLFSPDM_MODE_TCG, WOLFSPDM_MODE_NUVOTON, or WOLFSPDM_MODE_NATIONS

See: wolfTPM2_InitWithSpdmKey

Return:

- TPM_RC_SUCCESS: successful
- TPM_RC_UPGRADE: TPM firmware recovery is active
- BAD_FUNC_ARG: invalid parameters, or an identity mode whose vendor adapter is not compiled in
- WOLFSPDM_E_BAD_STATE: automatic vendor selection is unavailable
- a WOLFSPDM_E_* error if the authenticated handshake fails

29.9.6.5 function wolfTPM2_SpdmModeFromDidVid

```
WOLFTPM_TEST_API int wolfTPM2_SpdmModeFromDidVid(
    UINT32 didVid,
    WOLFSPDM_MODE * mode
)
```

29.9.6.6 function wolfTPM2_InitWithSpdmPsk

```
WOLFTPM_API int wolfTPM2_InitWithSpdmPsk(
    WOLFTPM2_DEV * dev,
    TPM2HalIoCb ioCb,
    void * userCtx,
```

```
    const byte * psk,
    word32 pskSz,
    const byte * hint,
    word32 hintSz
)
```

Initialize a TPM and authenticate an SPDM session with a PSK. This entry point also recovers a PSK-configured TPM that is already locked in SPDM-only mode by retrying TPM2_Startup over the secure session. On failure after argument validation, the device is fully torn down, except for TPM_RC_UPGRADE, which leaves an active context for the vendor firmware-recovery flow.

Parameters:

- **dev** pointer to an empty WOLFTPM2_DEV structure
- **ioCb** function pointer to an I/O callback
- **userCtx** pointer to a user context (can be NULL)
- **psk** pre-shared key bytes
- **pskSz** size of psk; must be nonzero
- **hint** optional PSK hint
- **hintSz** size of hint

See:

- wolfTPM2_Init
- wolfTPM2_SpdmConnectPsk

Return:

- TPM_RC_SUCCESS: successful
- TPM_RC_UPGRADE: TPM firmware recovery is active; call cleanup when the recovery flow is complete
- BAD_FUNC_ARG: invalid parameters
- a WOLFSPDM_E_* error if the authenticated handshake fails

29.9.6.7 function wolfTPM2_OpenExisting

```
WOLFTPM_API int wolfTPM2_OpenExisting(
    WOLFTPM2_DEV * dev,
    TPM2HalIoCb ioCb,
    void * userCtx
)
```

Use an already initialized TPM, in its current TPM locality.

Parameters:

- **dev** pointer to an empty structure of WOLFTPM2_DEV type
- **ioCb** function pointer to a IO callback (see [hal#file-tpm-io.h))
- **userCtx** pointer to a user context (can be NULL)

See:

- wolfTPM2_Init
- wolfTPM2_Cleanup
- TPM2_Init

Return:

- TPM_RC_SUCCESS: successful
- TPM_RC_FAILURE: generic failure (check TPM IO communication)
- BAD_FUNC_ARG: check the provided arguments

29.9.6.8 function wolfTPM2_Cleanup

```
WOLFTPM_API int wolfTPM2_Cleanup(
    WOLFTPM2_DEV * dev
)
```

Easy to use TPM and wolfcrypt deinitialization.

Parameters:

- **dev** pointer to a populated structure of WOLFTPM2_DEV type

See:

- wolfTPM2_OpenExisting
- wolfTPM2_Test
- TPM2_Init

Return:

- TPM_RC_SUCCESS: successful
- TPM_RC_FAILURE: generic failure (check TPM IO communication)
- BAD_FUNC_ARG: check the provided arguments

Note: Calls wolfTPM2_Cleanup_ex with appropriate doShutdown parameter

Example

```
int rc;

rc = wolfTPM2_Cleanup(&dev);
if (rc != TPM_RC_SUCCESS) {
    //wolfTPM2_Cleanup failed
    goto exit;
}
```

29.9.6.9 function wolfTPM2_Cleanup_ex

```
WOLFTPM_API int wolfTPM2_Cleanup_ex(
    WOLFTPM2_DEV * dev,
    int doShutdown
)
```

Deinitialization of a TPM (and wolfcrypt if it was used)

Parameters:

- **dev** pointer to a populated structure of WOLFTPM2_DEV type
- **doShutdown** flag value, if true a TPM2_Shutdown command will be executed

See:

- wolfTPM2_OpenExisting
- wolfTPM2_Test
- TPM2_Init

Return:

- TPM_RC_SUCCESS: successful
- TPM_RC_FAILURE: generic failure (check TPM IO communication)
- BAD_FUNC_ARG: check the provided arguments

Example

```
int rc;

//perform TPM2_Shutdown after deinitialization
rc = wolfTPM2_Cleanup_ex(&dev, 1);
if (rc != TPM_RC_SUCCESS) {
    //wolfTPM2_Cleanup_ex failed
    goto exit;
}
```

29.9.6.10 function wolfTPM2_GetTpmDevId

```
WOLFTPM_API int wolfTPM2_GetTpmDevId(
    WOLFTPM2_DEV * dev
)
```

Provides the device ID of a TPM.

Parameters:

- **dev** pointer to an populated structure of WOLFTPM2_DEV type

See:

- wolfTPM2_GetCapabilities
- wolfTPM2_Init

Return:

- an integer value of a valid TPM device ID
- or INVALID_DEVID if the TPM initialization could not extract DevID

Example

```
int tpmDevId;

tpmDevId = wolfTPM2_GetTpmDevId(&dev);
if (tpmDevId != INVALID_DEVID) {
    //wolfTPM2_Cleanup_ex failed
    goto exit;
}
```

29.9.6.11 function wolfTPM2_SelfTest

```
WOLFTPM_API int wolfTPM2_SelfTest(
    WOLFTPM2_DEV * dev
)
```

Asks the TPM to perform its self test.

Parameters:

- **dev** pointer to a populated structure of WOLFTPM2_DEV type

See:

- wolfTPM2_OpenExisting
- wolfTPM2_Test
- TPM2_Init

Return:

- TPM_RC_SUCCESS: successful

- TPM_RC_FAILURE: generic failure (check TPM IO communication and TPM return code)
- BAD_FUNC_ARG: check the provided arguments

Example

```
int rc;

//perform TPM2_Shutdown after deinitialization
rc = wolfTPM2_SelfTest(&dev);
if (rc != TPM_RC_SUCCESS) {
    //wolfTPM2_SelfTest failed
    goto exit;
}
```

29.9.6.12 function wolfTPM2_GetCapabilities

```
WOLFTPM_API int wolfTPM2_GetCapabilities(
    WOLFTPM2_DEV * dev,
    WOLFTPM2_CAPS * caps
)
```

Reports the available TPM capabilities.

Parameters:

- **dev** pointer to a populated structure of WOLFTPM2_DEV type
- **caps** pointer to an empty structure of WOLFTPM2_CAPS type to store the capabilities

See:

- wolfTPM2_GetTpmDevId
- wolfTPM2_SelfTest
- wolfTPM2_Init

Return:

- TPM_RC_SUCCESS: successful
- TPM_RC_FAILURE: generic failure (check TPM IO communication and TPM return code)
- BAD_FUNC_ARG: check the provided arguments

Example

```
int rc;
WOLFTPM2_CAPS caps;

//perform TPM2_Shutdown after deinitialization
rc = wolfTPM2_GetCapabilities(&dev, &caps);
if (rc != TPM_RC_SUCCESS) {
    //wolfTPM2_GetCapabilities failed
    goto exit;
}
```

29.9.6.13 function wolfTPM2_ParseCapabilities

```
WOLFTPM_TEST_API int wolfTPM2_ParseCapabilities(
    WOLFTPM2_CAPS * caps,
    TPML_TAGGED_TPM_PROPERTY * props
)
```

Decode a TPM property list into WOLFTPM2_CAPS.

Parameters:

- **caps** pointer to a WOLFTPM2_CAPS struct to fill
- **props** pointer to the tagged property list to decode

See: wolfTPM2_GetCapabilities

Return: TPM_RC_SUCCESS: successful

Note: Exposed so the decode can be unit tested against synthetic property lists; callers want wolfTPM2_GetCapabilities

29.9.6.14 function wolfTPM2_IsAlgSupported

```
WOLFTPM_API int wolfTPM2_IsAlgSupported(
    WOLFTPM2_DEV * dev,
    TPM_ALG_ID alg,
    int * isSupported
)
```

Report whether the TPM implements a given algorithm.

Parameters:

- **dev** pointer to a TPM2_DEV struct
- **alg** the algorithm identifier to test (for example TPM_ALG_SHA512)
- **isSupported** output, set to 1 if implemented by the TPM, else 0

See: wolfTPM2_GetCapabilities

Return:

- TPM_RC_SUCCESS: query completed; *isSupported is 1 or 0
- BAD_FUNC_ARG: dev or isSupported is NULL
- a TPM_RC (or other non-zero error) if the capability query fails

Note:

- Queries TPM_CAP_ALGS. Useful to skip a hash the TPM does not support (for example SHA2-512 on parts limited to SHA2-256/384) before starting a session with it.
- The result is returned through isSupported, not the return value, so a capability-query failure cannot be mistaken for "supported". On any error *isSupported is set to 0, so the call fails closed.
- This reports what the TPM implements, not what the local wolfCrypt build supports. A caller that also hashes locally (for example to precompute a policy digest) must check its own build as well.

29.9.6.15 function wolfTPM2_GetHandles

```
WOLFTPM_API int wolfTPM2_GetHandles(
    TPM_HANDLE handle,
    TPML_HANDLE * handles
)
```

Gets a list of handles.

Parameters:

- **handle** handle to start from (example: PCR_FIRST, NV_INDEX_FIRST, HMAC_SESSION_FIRST, POLICY_SESSION_FIRST, PERMANENT_FIRST, TRANSIENT_FIRST or PERSISTENT_FIRST)
- **handles** pointer to TPML_HANDLE to return handle results (optional)

See: wolfTPM2_GetCapabilities

Return:

- 0 or greater: successful, count of handles
- TPM_RC_FAILURE: generic failure (check TPM IO communication and TPM return code)
- BAD_FUNC_ARG: check the provided arguments
- BUFFER_E: the response did not carry a handle list

Example

```
int persistent_handle_count;

// get count of persistent handles
persistent_handle_count = wolfTPM2_GetHandles(PERSISTENT_FIRST, NULL);
```

29.9.6.16 function wolfTPM2_SpdmInit

```
WOLFTPM_API int wolfTPM2_SpdmInit(
    WOLFTPM2_DEV * dev
)
```

Initialize SPDM support on a wolfTPM2 device. Allocates and configures the SPDM context using wolfSPDM. After init, call wolfTPM2_SpdmConnect to establish a secure session.

Parameters:

- **dev** pointer to a WOLFTPM2_DEV structure

Return:

- TPM_RC_SUCCESS: successful
- BAD_FUNC_ARG: invalid parameters
- MEMORY_E: memory allocation failed

29.9.6.17 function wolfTPM2_SpdmConnect

```
WOLFTPM_API int wolfTPM2_SpdmConnect(
    WOLFTPM2_DEV * dev
)
```

Establish an SPDM secure session (full handshake). Uses standard SPDM flow: GET_VERSION -> GET_CAPABILITIES -> NEGOTIATE_ALGORITHMS -> KEY_EXCHANGE -> FINISH. Identity-key modes require a responder key pinned with wolfTPM2_SpdmSetResponderPubKey before this call.

Parameters:

- **dev** pointer to a WOLFTPM2_DEV structure

See: wolfTPM2_SpdmSetResponderPubKey

Return:

- TPM_RC_SUCCESS: session established
- TPM_RC_FAILURE: handshake failed
- WOLFSPDM_E_BAD_STATE: trusted responder key is not pinned

29.9.6.18 function wolfTPM2_SpdmIsConnected

```
WOLFTPM_API int wolfTPM2_SpdmIsConnected(
    WOLFTPM2_DEV * dev
)
```

Check if an SPDM secure session is currently active.

Parameters:

- **dev** pointer to a WOLFTPM2_DEV structure

Return: 1 if connected, 0 if not

29.9.6.19 function wolfTPM2_SpdmGetSessionId

```
WOLFTPM_API word32 wolfTPM2_SpdmGetSessionId(
    WOLFTPM2_DEV * dev
)
```

Get the current SPDM session ID.

Parameters:

- **dev** pointer to a WOLFTPM2_DEV structure

Return: Session ID, or 0 if not connected

29.9.6.20 function wolfTPM2_GetCapability_SPDMSessionInfo

```
WOLFTPM_API int wolfTPM2_GetCapability_SPDMSessionInfo(
    WOLFTPM2_DEV * dev,
    TPML_SPDM_SESSION_INFO * spdmSessionInfo
)
```

Read the TPM's view of the active SPDM session (GetCapability TPM_CAP_SPDM_SESSION_INFO, TPM 2.0 Library v1.84).

Parameters:

- **dev** pointer to a WOLFTPM2_DEV structure
- **spdmSessionInfo** output list of requester / TPM key names

See:

- wolfTPM2_PolicyTransportSPDM
- wolfTPM2_PolicyTransportSPDMMake

Return:

- TPM_RC_SUCCESS: successful
- TPM_RC_VALUE: the TPM does not implement the capability
- BAD_FUNC_ARG: check the provided arguments

Note: The TPM returns an empty list unless the command itself arrived inside an SPDM session. The names returned are the ones a PolicyTransportSPDM policy can bind to.

29.9.6.21 function wolfTPM2_SpdmDisconnect

```
WOLFTPM_API int wolfTPM2_SpdmDisconnect(
    WOLFTPM2_DEV * dev
)
```

Disconnect the SPDM secure session. After this, TPM commands are sent in the clear.

Parameters:

- **dev** pointer to a WOLFTPM2_DEV structure

Return: TPM_RC_SUCCESS: successful

29.9.6.22 function wolfTPM2_SpdmGetVersion

```
WOLFTPM_API int wolfTPM2_SpdmGetVersion(
    WOLFTPM2_DEV * dev
)
```

Send the SPDM GET_VERSION handshake step. Required before any vendor-defined SPDM command (e.g. PSK provisioning, GET_STATUS).

Parameters:

- **dev** pointer to a WOLFTPM2_DEV structure

Return: TPM_RC_SUCCESS: successful

29.9.6.23 function wolfTPM2_SpdmCleanup

```
WOLFTPM_API int wolfTPM2_SpdmCleanup(
    WOLFTPM2_DEV * dev
)
```

Free SPDM context and resources.

Parameters:

- **dev** pointer to a WOLFTPM2_DEV structure

Return: TPM_RC_SUCCESS: successful

29.9.6.24 function wolfTPM2_SpdmSetResponderPubKey

```
WOLFTPM_API int wolfTPM2_SpdmSetResponderPubKey(
    WOLFTPM2_DEV * dev,
    const byte * pubKey,
    word32 pubKeySz
)
```

Pin the trusted responder key before an identity-key connection.

Parameters:

- **dev** pointer to an initialized WOLFTPM2_DEV structure
- **pubKey** trusted raw P-384 responder key in X||Y format
- **pubKeySz** size of pubKey; must be 96 bytes

See:

- wolfTPM2_InitWithSpdmKey
- wolfTPM2_SpdmConnect

Return:

- WOLFSPDM_SUCCESS: successful
- BAD_FUNC_ARG: invalid device context
- WOLFSPDM_E_INVALID_ARG: invalid key pointer or size

29.9.6.25 function wolfTPM2_SpdmGetPubKey

```
WOLFTPM_API int wolfTPM2_SpdmGetPubKey(
    WOLFTPM2_DEV * dev,
    byte * pubKey,
```

```
    word32 * pubKeySz
)
```

Get the TPM's SPDM-Identity public key (shared TCG function).

Parameters:

- **dev** pointer to a WOLFTPM2_DEV structure
- **pubKey** output buffer for the public key
- **pubKeySz** in/out: buffer size / actual key size

See: wolfTPM2_SpdmSetResponderPubKey

Return:

- TPM_RC_SUCCESS: successful
- BAD_FUNC_ARG: invalid parameters

Note: This command runs in cleartext and its result is not a trust source. Use it only for diagnostics or compare it with a key obtained from a trusted provisioning channel. Do not install the returned key as its own trust anchor.

29.9.6.26 function wolfTPM2_SpdmSetNuvotonMode

```
WOLFTPM_API int wolfTPM2_SpdmSetNuvotonMode(
    WOLFTPM2_DEV * dev
)
```

Configure for Nuvoton TPM SPDM mode. Must be called before wolfTPM2_SpdmConnect() for Nuvoton TPMs.

Parameters:

- **dev** pointer to a WOLFTPM2_DEV structure

Return: TPM_RC_SUCCESS: successful

29.9.6.27 function wolfTPM2_SpdmEnable

```
WOLFTPM_API int wolfTPM2_SpdmEnable(
    WOLFTPM2_DEV * dev
)
```

Enable SPDM on the TPM via NTC2_PreConfig vendor command. Requires platform hierarchy auth. TPM must be reset after this call.

Parameters:

- **dev** pointer to a WOLFTPM2_DEV structure

Return: TPM_RC_SUCCESS: successful

29.9.6.28 function wolfTPM2_SpdmDisable

```
WOLFTPM_API int wolfTPM2_SpdmDisable(
    WOLFTPM2_DEV * dev
)
```

Disable SPDM on Nuvoton TPM via NTC2_PreConfig. Sets Cfg_H bit 1 to disable SPDM. Requires TPM reset to take effect.

Parameters:

- **dev** pointer to a WOLFTPM2_DEV structure

29.9.6.29 function wolfTPM2_SpdmConnectNuvoton

```
WOLFTPM_API int wolfTPM2_SpdmConnectNuvoton(
    WOLFTPM2_DEV * dev,
    const byte * reqPubKey,
    word32 reqPubKeySz,
    const byte * reqPrivKey,
    word32 reqPrivKeySz
)
```

Establish Nuvoton SPDM secure session with mutual authentication. Uses Nuvoton flow: GET_VERSION -> GET_PUB_KEY -> KEY_EXCHANGE -> GIVE_PUB_KEY -> FINISH. Requires a responder key pinned with wolfTPM2_SpdmSetResponderPubKey.

Parameters:

- **dev** pointer to a WOLFTPM2_DEV structure
- **reqPubKey** host's ECDSA P-384 public key (TPMT_PUBLIC format)
- **reqPubKeySz** size of reqPubKey in bytes
- **reqPrivKey** host's ECDSA P-384 private key (raw 48 bytes)
- **reqPrivKeySz** size of reqPrivKey in bytes

See: wolfTPM2_SpdmSetResponderPubKey

Return:

- TPM_RC_SUCCESS: session established
- TPM_RC_FAILURE: handshake failed
- WOLFSPDM_E_BAD_STATE: trusted responder key is not pinned

29.9.6.30 function wolfTPM2_SpdmGetStatus

```
WOLFTPM_API int wolfTPM2_SpdmGetStatus(
    WOLFTPM2_DEV * dev,
    WOLFSPDM_NUVOTON_STATUS * status
)
```

Get SPDM status from the TPM (GET_STS_ vendor command).

Parameters:

- **dev** pointer to a WOLFTPM2_DEV structure
- **status** output: SPDM status information

Return:

- TPM_RC_SUCCESS: successful
- BAD_FUNC_ARG: invalid parameters

29.9.6.31 function wolfTPM2_SpdmSetOnlyMode

```
WOLFTPM_API int wolfTPM2_SpdmSetOnlyMode(
    WOLFTPM2_DEV * dev,
    int lock
)
```

Lock or unlock SPDM-only mode. When locked, TPM only accepts commands over SPDM secure channel.

Parameters:

- **dev** pointer to a WOLFTPM2_DEV structure
- **lock** 1 to lock SPDM-only mode, 0 to unlock

Return: TPM_RC_SUCCESS: successful

29.9.6.32 function wolfTPM2_SpdmSetNationsMode

```
WOLFTPM_API int wolfTPM2_SpdmSetNationsMode(
    WOLFTPM2_DEV * dev
)
```

Configure for Nations Technology NS350 SPDM mode. Must be called before wolfTPM2_SpdmConnectNations().

Parameters:

- **dev** pointer to a WOLFTPM2_DEV structure

Return: TPM_RC_SUCCESS: successful

29.9.6.33 function wolfTPM2_SpdmConnectNations

```
WOLFTPM_API int wolfTPM2_SpdmConnectNations(
    WOLFTPM2_DEV * dev,
    const byte * reqPubKey,
    word32 reqPubKeySz,
    const byte * reqPrivKey,
    word32 reqPrivKeySz
)
```

Establish Nations SPDM secure session (identity key mode). Uses TCG flow: GET_VERSION -> GET_PUB_KEY -> KEY_EXCHANGE -> GIVE_PUB_KEY -> FINISH. Requires a responder key pinned with wolfTPM2_SpdmSetResponderPubKey.

Parameters:

- **dev** pointer to a WOLFTPM2_DEV structure
- **reqPubKey** host's ECDSA P-384 public key (TPMT_PUBLIC format, or NULL for auto-gen)
- **reqPubKeySz** size of reqPubKey in bytes
- **reqPrivKey** host's ECDSA P-384 private key (raw 48 bytes, or NULL for auto-gen)
- **reqPrivKeySz** size of reqPrivKey in bytes

See: wolfTPM2_SpdmSetResponderPubKey

Return:

- TPM_RC_SUCCESS: session established
- TPM_RC_FAILURE: handshake failed
- WOLFSPDM_E_BAD_STATE: trusted responder key is not pinned

29.9.6.34 function wolfTPM2_SpdmNationsIdentityKeySet

```
WOLFTPM_API int wolfTPM2_SpdmNationsIdentityKeySet(
    WOLFTPM2_DEV * dev,
    int set
)
```

Set/unset SPDM identity key provisioning on Nations NS350. Uses vendor command TPM2_VendorSpdmIdentityKeySe (0x20000708).

Parameters:

- **dev** pointer to a WOLFTPM2_DEV structure
- **set** 1 to provision identity key, 0 to un-provision

Return:

- TPM_RC_SUCCESS: successful
- BAD_FUNC_ARG: invalid parameters

29.9.6.35 function wolfTPM2_SpdmNationsGetStatus

```
WOLFTPM_API int wolfTPM2_SpdmNationsGetStatus(
    WOLFTPM2_DEV * dev,
    WOLFSPDM_NATIONS_STATUS * status
)
```

Get SPDM status from Nations TPM (PSK mode).

29.9.6.36 function wolfTPM2_SpdmNationsSetOnlyMode

```
WOLFTPM_API int wolfTPM2_SpdmNationsSetOnlyMode(
    WOLFTPM2_DEV * dev,
    int lock
)
```

Lock or unlock SPDM-only mode on Nations TPM (PSK mode).

29.9.6.37 function wolfTPM2_SpdmNationsPskSet

```
WOLFTPM_API int wolfTPM2_SpdmNationsPskSet(
    WOLFTPM2_DEV * dev,
    const byte * psk,
    word32 pskSz
)
```

Provision PSK on Nations TPM.

29.9.6.38 function wolfTPM2_SpdmNationsPskClear

```
WOLFTPM_API int wolfTPM2_SpdmNationsPskClear(
    WOLFTPM2_DEV * dev,
    const byte * clearAuth,
    word32 clearAuthSz
)
```

Clear PSK from Nations TPM. Requires ClearAuth from PSK_SET.

29.9.6.39 function wolfTPM2_SpdmSetInfineonMode

```
WOLFTPM_API int wolfTPM2_SpdmSetInfineonMode(
    WOLFTPM2_DEV * dev
)
```

Configure for Infineon SLB9678 SPDM (PSK mode, Algorithm Set B). Must be called before PSK provisioning vendor commands.

Parameters:

- **dev** pointer to a WOLFTPM2_DEV structure

Return: TPM_RC_SUCCESS: successful

29.9.6.40 function wolfTPM2_SpdmConnectInfineonTcg

```
WOLFTPM_API int wolfTPM2_SpdmConnectInfineonTcg(
    WOLFTPM2_DEV * dev,
    const byte * reqPubKey,
    word32 reqPubKeySz,
    const byte * reqPrivKey,
    word32 reqPrivKeySz
)
```

Establish an SLB9678 asymmetric session using the shared TCG flow. Requires firmware advertising asymmetric SPDM and a responder key pinned with wolfTPM2_SpdmSetResponderPubKey. No PSK fallback is attempted.

Parameters:

- **dev** initialized device with an SPDM context
- **reqPubKey** requester TPMT_PUBLIC, or NULL to generate a key pair
- **reqPubKeySz** public key size, or zero when generating a key pair
- **reqPrivKey** raw P-384 private key, or NULL to generate a key pair
- **reqPrivKeySz** private key size, or zero when generating a key pair

See:

- wolfTPM2_InitWithSpdmKey_ex
- wolfTPM2_SpdmConnectPsk

Return:

- TPM_RC_SUCCESS: session established
- BAD_FUNC_ARG: invalid context or requester key arguments
- WOLFSPDM_E_BAD_STATE: trusted responder key is not configured
- WOLFSPDM_E_PEER_ERROR: unsupported capabilities or peer failure

29.9.6.41 function wolfTPM2_SpdmInfineonGetStatus

```
WOLFTPM_API int wolfTPM2_SpdmInfineonGetStatus(
    WOLFTPM2_DEV * dev,
    WOLFSPDM_INFINEON_STATUS * status
)
```

Get SPDM status from Infineon SLB9678 (PSK mode).

29.9.6.42 function wolfTPM2_SpdmInfineonSetOnlyMode

```
WOLFTPM_API int wolfTPM2_SpdmInfineonSetOnlyMode(
    WOLFTPM2_DEV * dev,
    int lock
)
```

Lock or unlock SPDM-only mode on Infineon SLB9678.

29.9.6.43 function wolfTPM2_SpdmInfineonEnable

```
WOLFTPM_API int wolfTPM2_SpdmInfineonEnable(
    WOLFTPM2_DEV * dev,
    int enable,
    int lock
)
```

Enable or disable the SLB9678 SPDM-PSK secure channel via TPM2_SetCapability(TPM_CAP_CONFIGS, TPM_PT_CONFIG_SPDM_PSK). Requires Platform authorization configured in session slot zero. An unused slot uses empty-password authorization. Pass lock=0 to keep it reversible (enable is undone by disabling); lock=1 lifetime-locks the configuration.

Parameters:

- **dev** pointer to a WOLFTPM2_DEV structure
- **enable** 1 to enable SPDM-PSK, 0 to disable
- **lock** 1 to lifetime-lock the configuration (irreversible), 0 normally

Return: TPM_RC_SUCCESS: successful

29.9.6.44 function wolfTPM2_SpdmConnectPsk

```
WOLFTPM_API int wolfTPM2_SpdmConnectPsk(
    WOLFTPM2_DEV * dev,
    const byte * psk,
    word32 pskSz,
    const byte * hint,
    word32 hintSz
)
```

Establish SPDM secure session via the DSP0274 PSK handshake (PSK_EXCHANGE / PSK_FINISH). Vendor-neutral spec implementation.

29.9.6.45 function wolfTPM2_UnsetAuth

```
WOLFTPM_API int wolfTPM2_UnsetAuth(
    WOLFTPM2_DEV * dev,
    int index
)
```

Clears one of the TPM Authorization slots, pointed by its index number.

Parameters:

- **dev** pointer to a TPM2_DEV struct
- **index** valid TPM authorization slot from 0 through MAX_SESSION_NUM - 1

See:

- wolfTPM2_SetAuth
- wolfTPM2_SetAuthPassword
- wolfTPM2_SetAuthHandle
- wolfTPM2_SetAuthSession

Return:

- TPM_RC_SUCCESS: successful
- TPM_RC_FAILURE: unable to get lock on the TPM2 Context
- BAD_FUNC_ARG: check the provided arguments

29.9.6.46 function wolfTPM2_UnsetAuthSession

```
WOLFTPM_API int wolfTPM2_UnsetAuthSession(
    WOLFTPM2_DEV * dev,
    int index,
    WOLFTPM2_SESSION * session
)
```

Clears one of the TPM Authorization session slots, pointed by its index number and saves the nonce from the TPM so the session can continue to be used again with wolfTPM2_SetAuthSession.

Parameters:

- **dev** pointer to a TPM2_DEV struct
- **index** valid TPM authorization slot from 0 through MAX_SESSION_NUM - 1
- **session** pointer to a WOLFTPM2_SESSION struct used with wolfTPM2_StartSession and wolfTPM2_SetAuthSession

See:

- wolfTPM2_StartSession
- wolfTPM2_SetAuthSession

Return:

- TPM_RC_SUCCESS: successful
- TPM_RC_FAILURE: unable to get lock on the TPM2 Context
- BAD_FUNC_ARG: check the provided arguments

29.9.6.47 function wolfTPM2_SetAuth

```
WOLFTPM_API int wolfTPM2_SetAuth(
    WOLFTPM2_DEV * dev,
    int index,
    TPM_HANDLE sessionHandle,
    const TPM2B_AUTH * auth,
    TPMA_SESSION sessionAttributes,
    const TPM2B_NAME * name
)
```

Sets a TPM Authorization slot using the provided index, session handle, attributes and auth.

Parameters:

- **dev** pointer to a TPM2_DEV struct
- **index** valid TPM authorization slot from 0 through MAX_SESSION_NUM - 1
- **sessionHandle** integer value of TPM_HANDLE type
- **auth** pointer to a structure of type TPM2B_AUTH containing one TPM Authorization
- **sessionAttributes** integer value of type TPMA_SESSION, selecting one or more attributes for the Session
- **name** pointer to a TPM2B_NAME structure

See:

- wolfTPM2_SetAuthPassword
- wolfTPM2_SetAuthHandle
- wolfTPM2_SetAuthSession

Return:

- TPM_RC_SUCCESS: successful
- BAD_FUNC_ARG: check the provided arguments

Note: It is recommended to use one of the other wolfTPM2 wrappers, like wolfTPM2_SetAuthPassword. Because the wolfTPM2_SetAuth wrapper provides complete control over the TPM Authorization slot for advanced use cases. In most scenarios, wolfTPM2_SetAuthHandle and SetAuthPassword are used.

29.9.6.48 function wolfTPM2_SetAuthPassword

```
WOLFTPM_API int wolfTPM2_SetAuthPassword(
    WOLFTPM2_DEV * dev,
    int index,
    const TPM2B_AUTH * auth
)
```

Sets a TPM Authorization slot using the provided user auth, typically a password.

Parameters:

- **dev** pointer to a TPM2_DEV struct
- **index** valid TPM authorization slot from 0 through MAX_SESSION_NUM - 1
- **auth** pointer to a structure of type TPM2B_AUTH, typically containing a TPM Key Auth

See:

- wolfTPM2_SetAuthHandle
- wolfTPM2_SetAuthSession
- wolfTPM2_SetAuth

Return:

- TPM_RC_SUCCESS: successful
- BAD_FUNC_ARG: check the provided arguments

Note: Often used for authorizing the loading and use of TPM keys, including Primary Keys

29.9.6.49 function wolfTPM2_SetAuthHandle

```
WOLFTPM_API int wolfTPM2_SetAuthHandle(
    WOLFTPM2_DEV * dev,
    int index,
    const WOLFTPM2_HANDLE * handle
)
```

Sets a TPM Authorization slot using the user auth associated with a wolfTPM2 Handle.

Parameters:

- **dev** pointer to a TPM2_DEV struct
- **index** valid TPM authorization slot from 0 through MAX_SESSION_NUM - 1
- **handle** pointer to a populated structure of WOLFTPM2_HANDLE type

See:

- wolfTPM2_SetAuth
- wolfTPM2_SetAuthPassword
- wolfTPM2_SetAuthHandle
- wolfTPM2_SetAuthSession

Return:

- TPM_RC_SUCCESS: successful
- BAD_FUNC_ARG: check the provided arguments

Note: This wrapper is especially useful when using a TPM key for multiple operations and TPM Authorization is required again.

29.9.6.50 function wolfTPM2_SetAuthSession

```
WOLFTPM_API int wolfTPM2_SetAuthSession(
    WOLFTPM2_DEV * dev,
    int index,
    WOLFTPM2_SESSION * tpmSession,
    TPMA_SESSION sessionAttributes
)
```

Sets a TPM Authorization slot using the provided TPM session handle, index and session attributes.

Parameters:

- **dev** pointer to a TPM2_DEV struct
- **index** valid TPM authorization slot from 0 through MAX_SESSION_NUM - 1
- **tpmSession** pointer to a WOLFTPM2_SESSION struct used with wolfTPM2_StartSession and wolfTPM2_SetAuthSession
- **sessionAttributes** integer value of type TPMA_SESSION, selecting one or more attributes for the Session

See:

- wolfTPM2_SetAuth
- wolfTPM2_SetAuthPassword
- wolfTPM2_SetAuthHandle
- wolfTPM2_SetSessionHandle

Return:

- TPM_RC_SUCCESS: successful
- BAD_FUNC_ARG: check the provided arguments

Note: This wrapper is useful for configuring TPM sessions, e.g. session for parameter encryption

29.9.6.51 function wolfTPM2_SetSessionHandle

```
WOLFTPM_API int wolfTPM2_SetSessionHandle(
    WOLFTPM2_DEV * dev,
    int index,
    WOLFTPM2_SESSION * tpmSession
)
```

Sets a TPM Authorization slot using the provided wolfTPM2 session object.

Parameters:

- **dev** pointer to a TPM2_DEV struct
- **index** valid TPM authorization slot from 0 through MAX_SESSION_NUM - 1
- **tpmSession** pointer to a WOLFTPM2_SESSION struct used with wolfTPM2_StartSession and wolfTPM2_SetAuthSession

See:

- wolfTPM2_SetAuth
- wolfTPM2_SetAuthPassword
- wolfTPM2_SetAuthHandle
- wolfTPM2_SetAuthSession

Return:

- TPM_RC_SUCCESS: successful
- BAD_FUNC_ARG: check the provided arguments

Note: This wrapper is useful for configuring TPM sessions, e.g. session for parameter encryption

29.9.6.52 function wolfTPM2_SetAuthHandleName

```
WOLFTPM_API int wolfTPM2_SetAuthHandleName(
    WOLFTPM2_DEV * dev,
    int index,
    const WOLFTPM2_HANDLE * handle
)
```

Updates the Name used in a TPM Session with the Name associated with wolfTPM2 Handle.

Parameters:

- **dev** pointer to a TPM2_DEV struct
- **index** valid TPM authorization slot from 0 through MAX_SESSION_NUM - 1
- **handle** pointer to a populated structure of WOLFTPM2_HANDLE type

See:

- wolfTPM2_SetAuth
- wolfTPM2_SetAuthPassword
- wolfTPM2_SetAuthHandle
- wolfTPM2_SetAuthSession

Return:

- TPM_RC_SUCCESS: successful
- BAD_FUNC_ARG: check the provided arguments

Note: Typically, this wrapper is used from another wrappers and in very specific use cases. For example, wolfTPM2_NVWriteAuth

29.9.6.53 function wolfTPM2_StartSession

```
WOLFTPM_API int wolfTPM2_StartSession(
    WOLFTPM2_DEV * dev,
    WOLFTPM2_SESSION * session,
    WOLFTPM2_KEY * tpmKey,
    WOLFTPM2_HANDLE * bind,
    TPM_SE sesType,
    int encDecAlg
)
```

Create a TPM session, Policy, HMAC or Trial.

Parameters:

- **dev** pointer to a TPM2_DEV struct
- **session** pointer to an empty WOLFTPM2_SESSION struct
- **tpmKey** pointer to a WOLFTPM2_KEY that will be used as a salt for the session
- **bind** pointer to a WOLFTPM2_HANDLE that will be used to make the session bounded
- **sesType** byte value, the session type (HMAC, Policy or Trial)
- **encDecAlg** integer value, specifying the algorithm in case of parameter encryption (TPM_ALG_CFB or TPM_ALG_XOR). Any value not CFB or XOR is considered NULL and parameter encryption is disabled.

See:

- wolfTPM2_SetAuthSession
- wolfTPM2_StartSession_ex

Return:

- TPM_RC_SUCCESS: successful
- BAD_FUNC_ARG: check the provided arguments

Note: This wrapper can also be used to start TPM session for parameter encryption, see wolfTPM nvram or keygen example

29.9.6.54 function wolfTPM2_StartSession_ex

```
WOLFTPM_API int wolfTPM2_StartSession_ex(
    WOLFTPM2_DEV * dev,
    WOLFTPM2_SESSION * session,
    WOLFTPM2_KEY * tpmKey,
    WOLFTPM2_HANDLE * bind,
    TPM_SE sesType,
    int encDecAlg,
    TPMI_ALG_HASH authHash
)
```

Create a TPM session selecting the session auth/hash algorithm.

Parameters:

- **dev** pointer to a TPM2_DEV struct
- **session** pointer to an empty WOLFTPM2_SESSION struct
- **tpmKey** pointer to a WOLFTPM2_KEY that will be used as a salt for the session
- **bind** pointer to a WOLFTPM2_HANDLE that will be used to make the session bounded
- **sesType** byte value, the session type (HMAC, Policy or Trial)
- **encDecAlg** integer value, specifying the algorithm in case of parameter encryption (TPM_ALG_CFB or TPM_ALG_XOR). Any value not CFB or XOR is considered NULL and parameter encryption is disabled.
- **authHash** hash algorithm for the session (for example TPM_ALG_SHA256 or TPM_ALG_SHA512). TPM_ALG_NULL selects the wrapper default.

See:

- wolfTPM2_StartSession
- wolfTPM2_SetAuthSession

Return:

- TPM_RC_SUCCESS: successful
- BAD_FUNC_ARG: check the provided arguments

Note: Use this instead of wolfTPM2_StartSession when the session digest must match a non-default hash, such as SHA-512 policy digests in post-quantum environments

29.9.6.55 function wolfTPM2_CreateAuthSession_EkPolicy

```
WOLFTPM_API int wolfTPM2_CreateAuthSession_EkPolicy(
    WOLFTPM2_DEV * dev,
    WOLFTPM2_SESSION * tpmSession
)
```

Creates a TPM session with Policy Secret to satisfy the default EK policy.

Parameters:

- **dev** pointer to a TPM2_DEV struct
- **tpmSession** pointer to an empty WOLFTPM2_SESSION struct

See:

- wolfTPM2_SetAuthSession
- wolfTPM2_StartSession

Return:

- TPM_RC_SUCCESS: successful
- BAD_FUNC_ARG: check the provided arguments
- TPM_RC_FAILURE: check TPM return code, check available handles, check TPM IO

Note: This wrapper can be used only if the EK authorization is not changed from default

29.9.6.56 function wolfTPM2_CreateAuthSession_EkPolicy_ex

```
WOLFTPM_API int wolfTPM2_CreateAuthSession_EkPolicy_ex(
    WOLFTPM2_DEV * dev,
    WOLFTPM2_SESSION * tpmSession,
    TPMI_ALG_HASH authHash
)
```

Creates a TPM EK policy session selecting the session auth/hash algorithm.

Parameters:

- **dev** pointer to a TPM2_DEV struct
- **tpmSession** pointer to an empty WOLFTPM2_SESSION struct
- **authHash** hash algorithm for the session (for example TPM_ALG_SHA256 or TPM_ALG_SHA512). TPM_ALG_NULL selects the wrapper default.

See:

- wolfTPM2_CreateAuthSession_EkPolicy
- wolfTPM2_StartSession_ex

Return:

- TPM_RC_SUCCESS: successful
- BAD_FUNC_ARG: check the provided arguments
- TPM_RC_FAILURE: check TPM return code, check available handles, check TPM IO

Note: This wrapper can be used only if the EK authorization is not changed from default

29.9.6.57 function wolfTPM2_CreatePrimaryKey

```
WOLFTPM_API int wolfTPM2_CreatePrimaryKey(
    WOLFTPM2_DEV * dev,
    WOLFTPM2_KEY * key,
    TPM_HANDLE primaryHandle,
    TPMT_PUBLIC * publicTemplate,
    const byte * auth,
    int authSz
)
```

Single function to prepare and create a TPM 2.0 Primary Key.

Parameters:

- **dev** pointer to a TPM2_DEV struct
- **key** pointer to an empty struct of WOLFTPM2_KEY type
- **primaryHandle** integer value, specifying one of four TPM 2.0 Primary Seeds: TPM_RH_OWNER, TPM_RH_ENDORSEMENT, TPM_RH_PLATFORM or TPM_RH_NULL
- **publicTemplate** pointer to a TPMT_PUBLIC structure populated manually or using one of the wolfTPM2_GetKeyTemplate_... wrappers
- **auth** pointer to a string constant, specifying the password authorization for the Primary Key
- **authSz** integer value, specifying the size of the password authorization, in bytes

See:

- wolfTPM2_CreateKey
- wolfTPM2_CreatePrimaryKey_ex
- wolfTPM2_GetKeyTemplate_RSA
- wolfTPM2_GetKeyTemplate_ECC

Return:

- TPM_RC_SUCCESS: successful
- TPM_RC_FAILURE: generic failure (check TPM IO and TPM return code)
- BAD_FUNC_ARG: check the provided arguments

Note: TPM 2.0 allows only asymmetric RSA or ECC primary keys. Afterwards, both symmetric and asymmetric keys can be created under a TPM 2.0 Primary Key Typically, Primary Keys are used to create Hierarchies of TPM 2.0 Keys. The TPM uses a Primary Key to wrap the other keys, signing or decrypting.

29.9.6.58 function wolfTPM2_CreatePrimaryKey_ex

```
WOLFTPM_API int wolfTPM2_CreatePrimaryKey_ex(
    WOLFTPM2_DEV * dev,
    WOLFTPM2_PKEY * pkey,
    TPM_HANDLE primaryHandle,
    TPMT_PUBLIC * publicTemplate,
    const byte * auth,
    int authSz
)
```

Single function to prepare and create a TPM 2.0 Primary Key.

Parameters:

- **dev** pointer to a TPM2_DEV struct
- **pkey** pointer to an empty struct of WOLFTPM2_PKEY type including the creation hash and ticket.
- **primaryHandle** integer value, specifying one of four TPM 2.0 Primary Seeds: TPM_RH_OWNER, TPM_RH_ENDORSEMENT, TPM_RH_PLATFORM or TPM_RH_NULL
- **publicTemplate** pointer to a TPMT_PUBLIC structure populated manually or using one of the wolfTPM2_GetKeyTemplate_... wrappers
- **auth** pointer to a string constant, specifying the password authorization for the Primary Key
- **authSz** integer value, specifying the size of the password authorization, in bytes

See:

- wolfTPM2_CreateKey
- wolfTPM2_CreatePrimaryKey
- wolfTPM2_GetKeyTemplate_RSA

- wolfTPM2_GetKeyTemplate_ECC

Return:

- TPM_RC_SUCCESS: successful
- TPM_RC_FAILURE: generic failure (check TPM IO and TPM return code)
- BAD_FUNC_ARG: check the provided arguments

Note: TPM 2.0 allows only asymmetric RSA or ECC primary keys. Afterwards, both symmetric and asymmetric keys can be created under a TPM 2.0 Primary Key Typically, Primary Keys are used to create Hierarchies of TPM 2.0 Keys. The TPM uses a Primary Key to wrap the other keys, signing or decrypting.

29.9.6.59 function wolfTPM2_ChangeAuthKey

```
WOLFTPM_API int wolfTPM2_ChangeAuthKey(
    WOLFTPM2_DEV * dev,
    WOLFTPM2_KEY * key,
    WOLFTPM2_HANDLE * parent,
    const byte * auth,
    int authSz
)
```

Change the authorization secret of a TPM 2.0 key.

Parameters:

- **dev** pointer to a TPM2_DEV struct
- **key** pointer to an empty struct of WOLFTPM2_KEY type
- **parent** pointer to a struct of WOLFTPM2_HANDLE type, specifying a TPM 2.0 Primary Key to be used as the parent(Storage Key)
- **auth** pointer to a string constant, specifying the password authorization of the TPM 2.0 key
- **authSz** integer value, specifying the size of the password authorization, in bytes

See:

- wolfTPM2_CreatePrimaryKey
- wolfTPM2_SetAuthHandle
- wolfTPM2_UnloadHandle

Return:

- TPM_RC_SUCCESS: successful
- TPM_RC_FAILURE: generic failure (check TPM IO and TPM return code)
- BAD_FUNC_ARG: check the provided arguments

Note: TPM does not allow the authorization secret of a Primary Key to be changed. Instead, use wolfTPM2_CreatePrimary to create the same PrimaryKey with a new auth.

29.9.6.60 function wolfTPM2_CreateKey

```
WOLFTPM_API int wolfTPM2_CreateKey(
    WOLFTPM2_DEV * dev,
    WOLFTPM2_KEYBLOB * keyBlob,
    WOLFTPM2_HANDLE * parent,
    TPMT_PUBLIC * publicTemplate,
    const byte * auth,
    int authSz
)
```

Single function to prepare and create a TPM 2.0 Key.

Parameters:

- **dev** pointer to a TPM2_DEV struct
- **keyBlob** pointer to an empty struct of WOLFTPM2_KEYBLOB type
- **parent** pointer to a struct of WOLFTPM2_HANDLE type, specifying the a 2.0 Primary Key to be used as the parent(Storage Key)
- **publicTemplate** pointer to a TPMT_PUBLIC structure populated manually or using one of the wolfTPM2_GetKeyTemplate_... wrappers
- **auth** pointer to a string constant, specifying the password authorization for the TPM 2.0 Key
- **authSz** integer value, specifying the size of the password authorization, in bytes

See:

- wolfTPM2_LoadKey
- wolfTPM2_GetKeyTemplate_RSA
- wolfTPM2_GetKeyTemplate_ECC
- wolfTPM2_CreatePrimaryKey

Return:

- TPM_RC_SUCCESS: successful
- TPM_RC_FAILURE: generic failure (check TPM IO and TPM return code)
- BAD_FUNC_ARG: check the provided arguments

Note: This function only creates the key material and stores it into the keyblob argument. To load the key use wolfTPM2_LoadKey

29.9.6.61 function wolfTPM2_LoadKey

```
WOLFTPM_API int wolfTPM2_LoadKey(
    WOLFTPM2_DEV * dev,
    WOLFTPM2_KEYBLOB * keyBlob,
    WOLFTPM2_HANDLE * parent
)
```

Single function to load a TPM 2.0 key.

Parameters:

- **dev** pointer to a TPM2_DEV struct
- **keyBlob** pointer to a struct of WOLFTPM2_KEYBLOB type
- **parent** pointer to a struct of WOLFTPM2_HANDLE type, specifying a TPM 2.0 Primary Key to be used as the parent(Storage Key)

See:

- wolfTPM2_CreateKey
- wolfTPM2_CreatePrimaryKey
- wolfTPM2_GetKeyTemplate_RSA
- wolfTPM2_GetKeyTemplate_ECC

Return:

- TPM_RC_SUCCESS: successful
- TPM_RC_FAILURE: generic failure (check TPM IO and TPM return code)
- BAD_FUNC_ARG: check the provided arguments

Note: To load a TPM 2.0 key its parent(Primary Key) should also be loaded prior to this operation. Primary Keys are loaded when they are created.

29.9.6.62 function wolfTPM2_CreateAndLoadKey

```
WOLFTPM_API int wolfTPM2_CreateAndLoadKey(
    WOLFTPM2_DEV * dev,
    WOLFTPM2_KEY * key,
    WOLFTPM2_HANDLE * parent,
    TPMT_PUBLIC * publicTemplate,
    const byte * auth,
    int authSz
)
```

Single function to create and load a TPM 2.0 Key in one step.

Parameters:

- **dev** pointer to a TPM2_DEV struct
- **key** pointer to an empty struct of WOLFTPM2_KEY type
- **parent** pointer to a struct of WOLFTPM2_HANDLE type, specifying a TPM 2.0 Primary Key to be used as the parent(Storage Key)
- **publicTemplate** pointer to a TPMT_PUBLIC structure populated manually or using one of the wolfTPM2_GetKeyTemplate_... wrappers
- **auth** pointer to a string constant, specifying the password authorization of the TPM 2.0 key
- **authSz** integer value, specifying the size of the password authorization, in bytes

See:

- wolfTPM2_CreateKey
- wolfTPM2_GetKeyTemplate_RSA
- wolfTPM2_GetKeyTemplate_ECC

Return:

- TPM_RC_SUCCESS: successful
- TPM_RC_FAILURE: generic failure (check TPM IO and TPM return code)
- BAD_FUNC_ARG: check the provided arguments

29.9.6.63 function wolfTPM2_CreateLoadedKey

```
WOLFTPM_API int wolfTPM2_CreateLoadedKey(
    WOLFTPM2_DEV * dev,
    WOLFTPM2_KEYBLOB * keyBlob,
    WOLFTPM2_HANDLE * parent,
    TPMT_PUBLIC * publicTemplate,
    const byte * auth,
    int authSz
)
```

Creates and loads a key using single TPM 2.0 operation, and stores encrypted private key material.

Parameters:

- **dev** pointer to a TPM2_DEV struct
- **keyBlob** pointer to an empty struct of WOLFTPM2_KEYBLOB type, contains private key material as encrypted data
- **parent** pointer to a struct of WOLFTPM2_HANDLE type, specifying a TPM 2.0 Primary Key to be used as the parent(Storage Key)
- **publicTemplate** pointer to a TPMT_PUBLIC structure populated manually or using one of the wolfTPM2_GetKeyTemplate_... wrappers
- **auth** pointer to a string constant, specifying the password authorization of the TPM 2.0 key
- **authSz** integer value, specifying the size of the password authorization, in bytes

See:

- wolfTPM2_CreateAndLoadKey
- wolfTPM2_CreateKey
- wolfTPM2_LoadKey

Return:

- TPM_RC_SUCCESS: successful
- TPM_RC_FAILURE: generic failure (check TPM IO and TPM return code)
- BAD_FUNC_ARG: check the provided arguments

29.9.6.64 function wolfTPM2_LoadPublicKey

```
WOLFTPM_API int wolfTPM2_LoadPublicKey(
    WOLFTPM2_DEV * dev,
    WOLFTPM2_KEY * key,
    const TPM2B_PUBLIC * pub
)
```

Wrapper to load the public part of an external key.

Parameters:

- **dev** pointer to a TPM2_DEV struct
- **key** pointer to an empty struct of WOLFTPM2_KEY type
- **pub** pointer to a populated structure of TPM2B_PUBLIC type

See:

- wolfTPM2_LoadRsaPublicKey
- wolfTPM2_LoadEccPublicKey
- wolfTPM2_wolfTPM2_LoadPrivateKey

Return:

- TPM_RC_SUCCESS: successful
- TPM_RC_FAILURE: generic failure (check TPM IO and TPM return code)
- BAD_FUNC_ARG: check the provided arguments

Note: The key must be formatted to the format expected by the TPM, see the 'pub' argument and the alternative wrappers.

29.9.6.65 function wolfTPM2_LoadPublicKey_ex

```
WOLFTPM_API int wolfTPM2_LoadPublicKey_ex(
    WOLFTPM2_DEV * dev,
    WOLFTPM2_KEY * key,
    const TPM2B_PUBLIC * pub,
    TPM_HANDLE hierarchy
)
```

29.9.6.66 function wolfTPM2_LoadPrivateKey

```
WOLFTPM_API int wolfTPM2_LoadPrivateKey(
    WOLFTPM2_DEV * dev,
    const WOLFTPM2_KEY * parentKey,
    WOLFTPM2_KEY * key,
    const TPM2B_PUBLIC * pub,
```

```
    TPM2B_SENSITIVE * sens
)
```

Single function to import an external private key and load it into the TPM in one step.

Parameters:

- **dev** pointer to a TPM2_DEV struct
- **parentKey** pointer to a struct of WOLFTPM2_HANDLE type (required, must be a loaded storage key)
- **key** pointer to an empty struct of WOLFTPM2_KEY type
- **pub** pointer to a populated structure of TPM2B_PUBLIC type
- **sens** pointer to a populated structure of TPM2B_SENSITIVE type

See:

- wolfTPM2_CreateKey
- wolfTPM2_GetKeyTemplate_RSA
- wolfTPM2_GetKeyTemplate_ECC

Return:

- TPM_RC_SUCCESS: successful
- TPM_RC_FAILURE: generic failure (check TPM IO and TPM return code)
- BAD_FUNC_ARG: check the provided arguments

Note: The private key material needs to be prepared in a format that the TPM expects, see the 'sens' argument

29.9.6.67 function wolfTPM2_ImportPrivateKey

```
WOLFTPM_API int wolfTPM2_ImportPrivateKey(
    WOLFTPM2_DEV * dev,
    const WOLFTPM2_KEY * parentKey,
    WOLFTPM2_KEYBLOB * keyBlob,
    const TPM2B_PUBLIC * pub,
    TPM2B_SENSITIVE * sens
)
```

Single function to import an external private key and load it into the TPM in one step.

Parameters:

- **dev** pointer to a TPM2_DEV struct
- **parentKey** pointer to a struct of WOLFTPM2_HANDLE type (required, must be a loaded storage key)
- **keyBlob** pointer to an empty struct of WOLFTPM2_KEYBLOB type
- **pub** pointer to a populated structure of TPM2B_PUBLIC type
- **sens** pointer to a populated structure of TPM2B_SENSITIVE type

See:

- wolfTPM2_ImportRsaPrivateKey
- wolfTPM2_ImportEccPrivateKey

Return:

- TPM_RC_SUCCESS: successful
- TPM_RC_FAILURE: generic failure (check TPM IO and TPM return code)
- BAD_FUNC_ARG: check the provided arguments

Note: The primary key material needs to be prepared in a format that the TPM expects, see the 'sens' argument

29.9.6.68 function wolfTPM2_LoadRsaPublicKey

```
WOLFTPM_API int wolfTPM2_LoadRsaPublicKey(
    WOLFTPM2_DEV * dev,
    WOLFTPM2_KEY * key,
    const byte * rsaPub,
    word32 rsaPubSz,
    word32 exponent
)
```

Helper function to import the public part of an external RSA key.

Parameters:

- **dev** pointer to a TPM2_DEV struct
- **key** pointer to an empty struct of WOLFTPM2_KEY type
- **rsaPub** pointer to a byte buffer containing the public key material
- **rsaPubSz** integer value of word32 type, specifying the buffer size
- **exponent** integer value of word32 type, specifying the RSA exponent

See:

- wolfTPM2_LoadRsaPublicKey_ex
- wolfTPM2_LoadPublicKey
- wolfTPM2_LoadEccPublicKey
- wolfTPM2_ReadPublicKey

Return:

- TPM_RC_SUCCESS: successful
- TPM_RC_FAILURE: generic failure (check TPM IO and TPM return code)
- BAD_FUNC_ARG: check the provided arguments

Note: Recommended for use, because it does not require TPM format of the public part

29.9.6.69 function wolfTPM2_LoadRsaPublicKey_ex

```
WOLFTPM_API int wolfTPM2_LoadRsaPublicKey_ex(
    WOLFTPM2_DEV * dev,
    WOLFTPM2_KEY * key,
    const byte * rsaPub,
    word32 rsaPubSz,
    word32 exponent,
    TPMI_ALG_RSA_SCHEME scheme,
    TPMI_ALG_HASH hashAlg
)
```

Advanced helper function to import the public part of an external RSA key.

Parameters:

- **dev** pointer to a TPM2_DEV struct
- **key** pointer to an empty struct of WOLFTPM2_KEY type
- **rsaPub** pointer to a byte buffer containing the public key material
- **rsaPubSz** integer value of word32 type, specifying the buffer size
- **exponent** integer value of word32 type, specifying the RSA exponent
- **scheme** value of TPMI_ALG_RSA_SCHEME type, specifying the RSA scheme

- **hashAlg** value of TPMI_ALG_HASH type, specifying the TPM hashing algorithm

See:

- wolfTPM2_LoadRsaPublicKey
- wolfTPM2_LoadPublicKey
- wolfTPM2_LoadEccPublicKey
- wolfTPM2_ReadPublicKey

Return:

- TPM_RC_SUCCESS: successful
- TPM_RC_FAILURE: generic failure (check TPM IO and TPM return code)
- BAD_FUNC_ARG: check the provided arguments

Note: Allows the developer to specify TPM hashing algorithm and RSA scheme

29.9.6.70 function wolfTPM2_ImportRsaPrivateKey

```
WOLFTPM_API int wolfTPM2_ImportRsaPrivateKey(
    WOLFTPM2_DEV * dev,
    const WOLFTPM2_KEY * parentKey,
    WOLFTPM2_KEYBLOB * keyBlob,
    const byte * rsaPub,
    word32 rsaPubSz,
    word32 exponent,
    const byte * rsaPriv,
    word32 rsaPrivSz,
    TPMI_ALG_RSA_SCHEME scheme,
    TPMI_ALG_HASH hashAlg
)
```

Import an external RSA private key.

Parameters:

- **dev** pointer to a TPM2_DEV struct
- **parentKey** pointer to a struct of WOLFTPM2_HANDLE type (required, must be a loaded storage key)
- **keyBlob** pointer to an empty struct of WOLFTPM2_KEYBLOB type
- **rsaPub** pointer to a byte buffer, containing the public part of the RSA key
- **rsaPubSz** integer value of word32 type, specifying the public part buffer size
- **exponent** integer value of word32 type, specifying the RSA exponent
- **rsaPriv** pointer to a byte buffer, containing the private material of the RSA key
- **rsaPrivSz** integer value of word32 type, specifying the private material buffer size
- **scheme** value of TPMI_ALG_RSA_SCHEME type, specifying the RSA scheme
- **hashAlg** integer value of TPMI_ALG_HASH type, specifying a supported TPM 2.0 hash algorithm

See:

- wolfTPM2_ImportRsaPrivateKeySeed
- wolfTPM2_LoadRsaPrivateKey
- wolfTPM2_LoadRsaPrivateKey_ex
- wolfTPM2_LoadPrivateKey

Return:

- TPM_RC_SUCCESS: successful
- TPM_RC_FAILURE: generic failure (check TPM IO and TPM return code)
- BAD_FUNC_ARG: check the provided arguments

- BUFFER_E: arguments size is larger than what the TPM buffers allow

Note: The imported key is created with both sign and decrypt usage. To import a single-purpose key, call wolfTPM2_ImportRsaPrivateKeySeed with a restricted objectAttributes set.

29.9.6.71 function wolfTPM2_ImportRsaPrivateKeySeed

```
WOLFTPM_API int wolfTPM2_ImportRsaPrivateKeySeed(
    WOLFTPM2_DEV * dev,
    const WOLFTPM2_KEY * parentKey,
    WOLFTPM2_KEYBLOB * keyBlob,
    const byte * rsaPub,
    word32 rsaPubSz,
    word32 exponent,
    const byte * rsaPriv,
    word32 rsaPrivSz,
    TPMI_ALG_RSA_SCHEME scheme,
    TPMI_ALG_HASH hashAlg,
    TPMA_OBJECT attributes,
    byte * seed,
    word32 seedSz
)
```

Import an external RSA private key with custom seed.

Parameters:

- **dev** pointer to a TPM2_DEV struct
- **parentKey** pointer to a struct of WOLFTPM2_HANDLE type (required, must be a loaded storage key)
- **keyBlob** pointer to an empty struct of WOLFTPM2_KEYBLOB type
- **rsaPub** pointer to a byte buffer, containing the public part of the RSA key
- **rsaPubSz** integer value of word32 type, specifying the public part buffer size
- **exponent** integer value of word32 type, specifying the RSA exponent
- **rsaPriv** pointer to a byte buffer, containing the private material of the RSA key
- **rsaPrivSz** integer value of word32 type, specifying the private material buffer size
- **scheme** value of TPMI_ALG_RSA_SCHEME type, specifying the RSA scheme
- **hashAlg** integer value of TPMI_ALG_HASH type, specifying a supported TPM 2.0 hash algorithm
- **attributes** integer value of TPMA_OBJECT type, can contain one or more attributes, e.g. TPMA_OBJECT_fixedTPM (or 0 to automatically populate)
- **seedSz** Optional (use NULL) or supply a custom seed for KDF
- **seed** Size of the seed (use 32 bytes for SHA2-256)

See:

- wolfTPM2_ImportRsaPrivateKey
- wolfTPM2_LoadRsaPrivateKey
- wolfTPM2_LoadRsaPrivateKey_ex
- wolfTPM2_LoadPrivateKey

Return:

- TPM_RC_SUCCESS: successful
- TPM_RC_FAILURE: generic failure (check TPM IO and TPM return code)
- BAD_FUNC_ARG: check the provided arguments
- BUFFER_E: arguments size is larger than what the TPM buffers allow

29.9.6.72 function wolfTPM2_LoadRsaPrivateKey

```
WOLFTPM_API int wolfTPM2_LoadRsaPrivateKey(
    WOLFTPM2_DEV * dev,
    const WOLFTPM2_KEY * parentKey,
    WOLFTPM2_KEY * key,
    const byte * rsaPub,
    word32 rsaPubSz,
    word32 exponent,
    const byte * rsaPriv,
    word32 rsaPrivSz
)
```

Helper function to import and load an external RSA private key in one step.

Parameters:

- **dev** pointer to a TPM2_DEV struct
- **parentKey** pointer to a struct of WOLFTPM2_HANDLE type (required, must be a loaded storage key)
- **key** pointer to an empty struct of WOLFTPM2_KEY type
- **rsaPub** pointer to a byte buffer, containing the public part of the RSA key
- **rsaPubSz** integer value of word32 type, specifying the public part buffer size
- **exponent** integer value of word32 type, specifying the RSA exponent
- **rsaPriv** pointer to a byte buffer, containing the private material of the RSA key
- **rsaPrivSz** integer value of word32 type, specifying the private material buffer size

See:

- wolfTPM2_ImportRsaPrivateKey
- wolfTPM2_LoadRsaPrivateKey_ex
- wolfTPM2_LoadPrivateKey

Return:

- TPM_RC_SUCCESS: successful
- TPM_RC_FAILURE: generic failure (check TPM IO and TPM return code)
- BAD_FUNC_ARG: check the provided arguments

29.9.6.73 function wolfTPM2_LoadRsaPrivateKey_ex

```
WOLFTPM_API int wolfTPM2_LoadRsaPrivateKey_ex(
    WOLFTPM2_DEV * dev,
    const WOLFTPM2_KEY * parentKey,
    WOLFTPM2_KEY * key,
    const byte * rsaPub,
    word32 rsaPubSz,
    word32 exponent,
    const byte * rsaPriv,
    word32 rsaPrivSz,
    TPMI_ALG_RSA_SCHEME scheme,
    TPMI_ALG_HASH hashAlg
)
```

Advanced helper function to import and load an external RSA private key in one step.

Parameters:

- **dev** pointer to a TPM2_DEV struct

- **parentKey** pointer to a struct of WOLFTPM2_HANDLE type (required, must be a loaded storage key)
- **key** pointer to an empty struct of WOLFTPM2_KEY type
- **rsaPub** pointer to a byte buffer, containing the public part of the RSA key
- **rsaPubSz** integer value of word32 type, specifying the public part buffer size
- **exponent** integer value of word32 type, specifying the RSA exponent
- **rsaPriv** pointer to a byte buffer, containing the private material of the RSA key
- **rsaPrivSz** integer value of word32 type, specifying the private material buffer size
- **scheme** value of TPMI_ALG_RSA_SCHEME type, specifying the RSA scheme
- **hashAlg** value of TPMI_ALG_HASH type, specifying the TPM hashing algorithm

See:

- wolfTPM2_LoadRsaPrivateKey
- wolfTPM2_LoadPrivateKey
- wolfTPM2_ImportRsaPrivateKey
- wolfTPM2_LoadEccPrivateKey
- wolfTPM2_SetKeyAuthPassword

Return:

- TPM_RC_SUCCESS: successful
- TPM_RC_FAILURE: generic failure (check TPM IO and TPM return code)
- BAD_FUNC_ARG: check the provided arguments

Warning: If key->handle.auth is empty the key is loaded with empty authorization and any caller holding the handle can use it. Set key->handle.auth (e.g. via wolfTPM2_SetKeyAuthPassword) before loading to require a password.

29.9.6.74 function wolfTPM2_LoadEccPublicKey

```
WOLFTPM_API int wolfTPM2_LoadEccPublicKey(
    WOLFTPM2_DEV * dev,
    WOLFTPM2_KEY * key,
    int curveId,
    const byte * eccPubX,
    word32 eccPubXSz,
    const byte * eccPubY,
    word32 eccPubYSz
)
```

Helper function to import the public part of an external ECC key.

Parameters:

- **dev** pointer to a TPM2_DEV struct
- **key** pointer to an empty struct of WOLFTPM2_KEY type
- **curveId** integer value, one of the accepted TPM_ECC_CURVE values
- **eccPubX** pointer to a byte buffer containing the public material of point X
- **eccPubXSz** integer value of word32 type, specifying the point X buffer size
- **eccPubY** pointer to a byte buffer containing the public material of point Y
- **eccPubYSz** integer value of word32 type, specifying the point Y buffer size

See:

- wolfTPM2_LoadPublicKey
- wolfTPM2_LoadRsaPublicKey
- wolfTPM2_ReadPublicKey
- wolfTPM2_LoadEccPrivateKey

Return:

- TPM_RC_SUCCESS: successful
- TPM_RC_FAILURE: generic failure (check TPM IO and TPM return code)
- BAD_FUNC_ARG: check the provided arguments

Note: Recommended for use, because it does not require TPM format of the public part

29.9.6.75 function wolfTPM2_LoadEccPublicKey_ex

```
WOLFTPM_API int wolfTPM2_LoadEccPublicKey_ex(
    WOLFTPM2_DEV * dev,
    WOLFTPM2_KEY * key,
    int curveId,
    const byte * eccPubX,
    word32 eccPubXSz,
    const byte * eccPubY,
    word32 eccPubYSz,
    TPMI_ALG_ECC_SCHEME scheme,
    TPMI_ALG_HASH hashAlg,
    TPMA_OBJECT objectAttributes
)
```

Variant of wolfTPM2_LoadEccPublicKey that lets the caller select the signing scheme, hash algorithm, and object attributes - required for loading ECDH peer keys (decrypt attribute) or non-default scheme/hash combinations.

Parameters:

- **dev** pointer to a TPM2_DEV struct
- **key** pointer to an empty struct of WOLFTPM2_KEY type
- **curveId** integer value, one of the accepted TPM_ECC_CURVE values
- **eccPubX** pointer to a byte buffer containing the public material of point X
- **eccPubXSz** integer value of word32 type, specifying the point X buffer size
- **eccPubY** pointer to a byte buffer containing the public material of point Y
- **eccPubYSz** integer value of word32 type, specifying the point Y buffer size
- **scheme** ECC signing scheme (e.g. TPM_ALG_ECDSA, TPM_ALG_ECDH, TPM_ALG_NULL). Use TPM_ALG_ECDH for ECDH peer keys
- **hashAlg** hash algorithm bound to scheme; ignored when scheme is TPM_ALG_NULL
- **objectAttributes** TPMA_OBJECT bitmask. Use TPMA_OBJECT_decrypt for ECDH peer keys, TPMA_OBJECT_sign for verification keys; typically OR'd with TPMA_OBJECT_userWithAuth and TPMA_OBJECT_noDA

See:

- wolfTPM2_LoadEccPublicKey
- wolfTPM2_LoadRsaPublicKey_ex

Return:

- TPM_RC_SUCCESS: successful
- BAD_FUNC_ARG: check the provided arguments, or scheme is TPM_ALG_ECDAA (use wolfTPM2_LoadPublicKey directly for ECDAA)
- BUFFER_E: caller buffer is too small for ECC point coordinates

29.9.6.76 function wolfTPM2_ImportEccPrivateKey

```
WOLFTPM_API int wolfTPM2_ImportEccPrivateKey(
    WOLFTPM2_DEV * dev,
```

```
    const WOLFTPM2_KEY * parentKey,
    WOLFTPM2_KEYBLOB * keyBlob,
    int curveId,
    const byte * eccPubX,
    word32 eccPubXSz,
    const byte * eccPubY,
    word32 eccPubYSz,
    const byte * eccPriv,
    word32 eccPrivSz
)
```

Helper function to import the private material of an external ECC key.

Parameters:

- **dev** pointer to a TPM2_DEV struct
- **parentKey** pointer to a struct of WOLFTPM2_HANDLE type (required, must be a loaded storage key)
- **keyBlob** pointer to an empty struct of WOLFTPM2_KEYBLOB type
- **curveId** integer value, one of the accepted TPM_ECC_CURVE values
- **eccPubX** pointer to a byte buffer containing the public material of point X
- **eccPubXSz** integer value of word32 type, specifying the point X buffer size
- **eccPubY** pointer to a byte buffer containing the public material of point Y
- **eccPubYSz** integer value of word32 type, specifying the point Y buffer size
- **eccPriv** pointer to a byte buffer containing the private material
- **eccPrivSz** integer value of word32 type, specifying the private material size

See:

- wolfTPM2_ImportEccPrivateKeySeed
- wolfTPM2_LoadEccPrivateKey
- wolfTPM2_LoadEccPrivateKey_ex
- wolfTPM2_LoadPrivateKey

Return:

- TPM_RC_SUCCESS: successful
- TPM_RC_FAILURE: generic failure (check TPM IO and TPM return code)
- BAD_FUNC_ARG: check the provided arguments

Note: The imported key is created with both sign and decrypt usage. To import a single-purpose key, call wolfTPM2_ImportEccPrivateKeySeed with a restricted objectAttributes set.

29.9.6.77 function wolfTPM2_ImportEccPrivateKeySeed

```
WOLFTPM_API int wolfTPM2_ImportEccPrivateKeySeed(
    WOLFTPM2_DEV * dev,
    const WOLFTPM2_KEY * parentKey,
    WOLFTPM2_KEYBLOB * keyBlob,
    int curveId,
    const byte * eccPubX,
    word32 eccPubXSz,
    const byte * eccPubY,
    word32 eccPubYSz,
    const byte * eccPriv,
    word32 eccPrivSz,
    TPMA_OBJECT attributes,
    byte * seed,
```

```
    word32 seedSz
)
```

Helper function to import the private material of an external ECC key.

Parameters:

- **dev** pointer to a TPM2_DEV struct
- **parentKey** pointer to a struct of WOLFTPM2_HANDLE type (required, must be a loaded storage key)
- **keyBlob** pointer to an empty struct of WOLFTPM2_KEYBLOB type
- **curveId** integer value, one of the accepted TPM_ECC_CURVE values
- **eccPubX** pointer to a byte buffer containing the public material of point X
- **eccPubXSz** integer value of word32 type, specifying the point X buffer size
- **eccPubY** pointer to a byte buffer containing the public material of point Y
- **eccPubYSz** integer value of word32 type, specifying the point Y buffer size
- **eccPriv** pointer to a byte buffer containing the private material
- **eccPrivSz** integer value of word32 type, specifying the private material size
- **attributes** integer value of TPMA_OBJECT type, can contain one or more attributes, e.g. TPMA_OBJECT_fixedTPM (or 0 to automatically populate)
- **seedSz** Optional (use NULL) or supply a custom seed for KDF
- **seed** Size of the seed (use 32 bytes for SHA2-256)

See:

- wolfTPM2_ImportEccPrivateKey
- wolfTPM2_LoadEccPrivateKey
- wolfTPM2_LoadEccPrivateKey_ex
- wolfTPM2_LoadPrivateKey

Return:

- TPM_RC_SUCCESS: successful
- TPM_RC_FAILURE: generic failure (check TPM IO and TPM return code)
- BAD_FUNC_ARG: check the provided arguments

29.9.6.78 function wolfTPM2_LoadEccPrivateKey

```
WOLFTPM_API int wolfTPM2_LoadEccPrivateKey(
    WOLFTPM2_DEV * dev,
    const WOLFTPM2_KEY * parentKey,
    WOLFTPM2_KEY * key,
    int curveId,
    const byte * eccPubX,
    word32 eccPubXSz,
    const byte * eccPubY,
    word32 eccPubYSz,
    const byte * eccPriv,
    word32 eccPrivSz
)
```

Helper function to import and load an external ECC private key in one step.

Parameters:

- **dev** pointer to a TPM2_DEV struct
- **parentKey** pointer to a struct of WOLFTPM2_HANDLE type (required, must be a loaded storage key)
- **key** pointer to an empty struct of WOLFTPM2_KEY type

- **curveId** integer value, one of the accepted TPM_ECC_CURVE values
- **eccPubX** pointer to a byte buffer containing the public material of point X
- **eccPubXSz** integer value of word32 type, specifying the point X buffer size
- **eccPubY** pointer to a byte buffer containing the public material of point Y
- **eccPubYSz** integer value of word32 type, specifying the point Y buffer size
- **eccPriv** pointer to a byte buffer containing the private material
- **eccPrivSz** integer value of word32 type, specifying the private material size

See:

- wolfTPM2_ImportEccPrivateKey
- wolfTPM2_LoadEccPublicKey
- wolfTPM2_LoadPrivateKey
- wolfTPM2_SetKeyAuthPassword

Return:

- TPM_RC_SUCCESS: successful
- TPM_RC_FAILURE: generic failure (check TPM IO and TPM return code)
- BAD_FUNC_ARG: check the provided arguments

Warning: If key->handle.auth is empty the key is loaded with empty authorization and any caller holding the handle can use it. Set key->handle.auth (e.g. via wolfTPM2_SetKeyAuthPassword) before loading to require a password.

29.9.6.79 function wolfTPM2_ReadPublicKey

```
WOLFTPM_API int wolfTPM2_ReadPublicKey(
    WOLFTPM2_DEV * dev,
    WOLFTPM2_KEY * key,
    const TPM_HANDLE handle
)
```

Helper function to receive the public part of a loaded TPM object using its handle.

Parameters:

- **dev** pointer to a TPM2_DEV struct
- **key** pointer to an empty struct of WOLFTPM2_KEY type
- **handle** integer value of TPM_HANDLE type, specifying handle of a loaded TPM object

See:

- wolfTPM2_LoadRsaPublicKey
- wolfTPM2_LoadEccPublicKey
- wolfTPM2_LoadPublicKey

Return:

- TPM_RC_SUCCESS: successful
- TPM_RC_FAILURE: generic failure (check TPM IO and TPM return code)
- BAD_FUNC_ARG: check the provided arguments

Note: The public part of a TPM symmetric keys contains just TPM meta data

29.9.6.80 function wolfTPM2_CreateKeySeal

```
WOLFTPM_API int wolfTPM2_CreateKeySeal(
    WOLFTPM2_DEV * dev,
    WOLFTPM2_KEYBLOB * keyBlob,
    WOLFTPM2_HANDLE * parent,
```

```
    TPMT_PUBLIC * publicTemplate,
    const byte * auth,
    int authSz,
    const byte * sealData,
    int sealSize
)
```

Using this wrapper a secret can be sealed inside a TPM 2.0 Key.

Parameters:

- **dev** pointer to a TPM2_DEV struct
- **keyBlob** pointer to an empty struct of WOLFTPM2_KEYBLOB type
- **parent** pointer to a struct of WOLFTPM2_HANDLE type, specifying the a 2.0 Primary Key to be used as the parent(Storage Key)
- **publicTemplate** pointer to a TPMT_PUBLIC structure populated using one of the wolfTPM2_GetKeyTemplate_Key
- **auth** pointer to a string constant, specifying the password authorization for the TPM 2.0 Key
- **authSz** integer value, specifying the size of the password authorization, in bytes
- **sealData** pointer to a byte buffer, containing the secret(user data) to be sealed
- **sealSize** integer value, specifying the size of the seal buffer, in bytes

See:

- wolfTPM2_GetKeyTemplate_KeySeal
- TPM2_Unseal
- wolfTPM2_CreatePrimary

Return:

- TPM_RC_SUCCESS: successful
- TPM_RC_FAILURE: generic failure (check TPM IO and TPM return code)
- BAD_FUNC_ARG: check the provided arguments

Note: The secret size can not be larger than 128 bytes

29.9.6.81 function wolfTPM2_CreateKeySeal_ex

```
WOLFTPM_API int wolfTPM2_CreateKeySeal_ex(
    WOLFTPM2_DEV * dev,
    WOLFTPM2_KEYBLOB * keyBlob,
    WOLFTPM2_HANDLE * parent,
    TPMT_PUBLIC * publicTemplate,
    const byte * auth,
    int authSz,
    TPM_ALG_ID pcrAlg,
    byte * pcrArray,
    word32 pcrArraySz,
    const byte * sealData,
    int sealSize
)
```

Using this wrapper a secret can be sealed inside a TPM 2.0 Key with pcr selection.

Parameters:

- **dev** pointer to a WOLFTPM2_DEV struct
- **keyBlob** pointer to an empty struct of WOLFTPM2_KEYBLOB type
- **parent** pointer to a struct of WOLFTPM2_HANDLE type, specifying the a 2.0 Primary Key to be used as the parent(Storage Key)
- **publicTemplate** pointer to a TPMT_PUBLIC structure populated using one of the wolfTPM2_GetKeyTemplate_Key

- **auth** pointer to a string constant, specifying the password authorization for the TPM 2.0 Key
- **authSz** integer value, specifying the size of the password authorization, in bytes
- **pcrAlg** hash algorithm to use when calculating pcr digest
- **pcrArray** optional array of pcrs to be used when creating the tpm object
- **pcrArraySz** length of the pcrArray
- **sealData** pointer to a byte buffer, containing the secret(user data) to be sealed
- **sealSize** integer value, specifying the size of the seal buffer, in bytes

See:

- wolfTPM2_GetKeyTemplate_KeySeal
- TPM2_Unseal
- wolfTPM2_CreatePrimary

Return:

- TPM_RC_SUCCESS: successful
- TPM_RC_FAILURE: generic failure (check TPM IO and TPM return code)
- BAD_FUNC_ARG: check the provided arguments

Note: The secret size can not be larger than 128 bytes

29.9.6.82 function wolfTPM2_ComputeName

```
WOLFTPM_API int wolfTPM2_ComputeName(
    const TPM2B_PUBLIC * pub,
    TPM2B_NAME * out
)
```

Helper function to generate a hash of the public area of an object in the format expected by the TPM.

Parameters:

- **pub** pointer to a populated structure of TPM2B_PUBLIC type, containing the public area of a TPM object
- **out** pointer to an empty struct of TPM2B_NAME type, to store the computed name

See: wolfTPM2_ImportPrivateKey

Return:

- TPM_RC_SUCCESS: successful
- TPM_RC_FAILURE: generic failure (check TPM IO and TPM return code)
- BAD_FUNC_ARG: check the provided arguments

Note: Computed TPM name includes hash of the TPM_ALG_ID and the public are of the object

29.9.6.83 function wolfTPM2_SensitiveToPrivate

```
WOLFTPM_API int wolfTPM2_SensitiveToPrivate(
    TPM2B_SENSITIVE * sens,
    TPM2B_PRIVATE * priv,
    TPMI_ALG_HASH nameAlg,
    TPM2B_NAME * name,
    const WOLFTPM2_KEY * parentKey,
    TPMT_SYM_DEF_OBJECT * sym,
    TPM2B_DATA * symSeed
)
```

Helper function to convert TPM2B_SENSITIVE.

Parameters:

- **sens** pointer to a correctly populated structure of TPM2B_SENSITIVE type
- **priv** pointer to an empty struct of TPM2B_PRIVATE type
- **nameAlg** integer value of TPMI_ALG_HASH type, specifying a valid TPM2 hashing algorithm
- **name** pointer to a TPM2B_NAME structure
- **parentKey** pointer to a WOLFTPM2_KEY structure, specifying a parentKey, if it exists
- **sym** pointer to a structure of TPMT_SYM_DEF_OBJECT type
- **symSeed** pointer to a structure of derived secret (RSA=random, ECC=ECDHE)

See: wolfTPM2_ImportPrivateKey

Return:

- TPM_RC_SUCCESS: successful
- TPM_RC_FAILURE: generic failure (check TPM IO and TPM return code)
- BAD_FUNC_ARG: check the provided arguments

29.9.6.84 function wolfTPM2_RsaKey_Exponent

```
WOLFTPM_API word32 wolfTPM2_RsaKey_Exponent(
    const byte * e,
    word32 eSz
)
```

Converts a big-endian byte array to a native word32 value. Used for RSA exponent conversion from ASN.1/DER format.

Parameters:

- **e** pointer to big-endian byte array
- **eSz** size of the byte array (max 4 bytes)

Return: word32 value in native byte order

29.9.6.85 function wolfTPM2_ImportPrivateKeyBuffer

```
WOLFTPM_API int wolfTPM2_ImportPrivateKeyBuffer(
    WOLFTPM2_DEV * dev,
    const WOLFTPM2_KEY * parentKey,
    int keyType,
    WOLFTPM2_KEYBLOB * keyBlob,
    int encodingType,
    const char * input,
    word32 inSz,
    const char * pass,
    TPMA_OBJECT objectAttributes,
    byte * seed,
    word32 seedSz
)
```

Helper function to import PEM/DER or RSA/ECC private key.

Parameters:

- **dev** pointer to a TPM2_DEV struct
- **keyType** The type of key (TPM_ALG_RSA or TPM_ALG_ECC)
- **parentKey** pointer to a WOLFTPM2_KEY struct, pointing to a Primary Key or TPM Hierarchy
- **keyBlob** pointer to a struct of WOLFTPM2_KEYBLOB type, to import the private key to
- **encodingType** ENCODING_TYPE_PEM or ENCODING_TYPE_ASN1 (DER)

- **input** buffer holding the rsa pem
- **inSz** length of the input pem buffer
- **pass** optional password of the key
- **objectAttributes** integer value of TPMA_OBJECT type, can contain one or more attributes, e.g. TPMA_OBJECT_fixedTPM
- **seedSz** Optional (use NULL) or supply a custom seed for KDF
- **seed** Size of the seed (use 32 bytes for SHA2-256)

Return:

- TPM_RC_SUCCESS: successful
- TPM_RC_FAILURE: generic failure (check TPM IO and TPM return code)
- BAD_FUNC_ARG: check the provided arguments

29.9.6.86 function wolfTPM2_ImportPublicKeyBuffer

```
WOLFTPM_API int wolfTPM2_ImportPublicKeyBuffer(
    WOLFTPM2_DEV * dev,
    int keyType,
    WOLFTPM2_KEY * key,
    int encodingType,
    const char * input,
    word32 inSz,
    TPMA_OBJECT objectAttributes
)
```

Helper function to import PEM/DER formatted RSA/ECC public key.

Parameters:

- **dev** pointer to a TPM2_DEV struct
- **keyType** The type of key (TPM_ALG_RSA or TPM_ALG_ECC)
- **key** pointer to a struct of WOLFTPM2_KEY type, to import the public key to
- **encodingType** ENCODING_TYPE_PEM or ENCODING_TYPE_ASN1 (DER)
- **input** buffer holding the rsa pem
- **inSz** length of the input pem buffer
- **objectAttributes** integer value of OR'd TPMA_OBJECT_* types

Return:

- TPM_RC_SUCCESS: successful - populates key->pub
- TPM_RC_FAILURE: generic failure (check TPM IO and TPM return code)
- BAD_FUNC_ARG: check the provided arguments

29.9.6.87 function wolfTPM2_ExportPublicKeyBuffer

```
WOLFTPM_API int wolfTPM2_ExportPublicKeyBuffer(
    WOLFTPM2_DEV * dev,
    WOLFTPM2_KEY * tpmKey,
    int encodingType,
    byte * out,
    word32 * outSz
)
```

Helper function to export a TPM RSA/ECC public key with PEM/DER formatting.

Parameters:

- **dev** pointer to a TPM2_DEV struct

- **tpmKey** pointer to a WOLFTPM2_KEY with populated key
- **encodingType** ENCODING_TYPE_PEM or ENCODING_TYPE_ASN1 (DER)
- **out** buffer to export public key
- **outSz** pointer to length of the out buffer

Return:

- TPM_RC_SUCCESS: successful - populates key->pub
- TPM_RC_FAILURE: generic failure (check TPM IO and TPM return code)
- BUFFER_E: insufficient space in provided buffer
- BAD_FUNC_ARG: check the provided arguments

29.9.6.88 function wolfTPM2_RsaPrivateKeyImportDer

```
WOLFTPM_API int wolfTPM2_RsaPrivateKeyImportDer(
    WOLFTPM2_DEV * dev,
    const WOLFTPM2_KEY * parentKey,
    WOLFTPM2_KEYBLOB * keyBlob,
    const byte * input,
    word32 inSz,
    TPMI_ALG_RSA_SCHEME scheme,
    TPMI_ALG_HASH hashAlg
)
```

Helper function to import Der rsa key directly.

Parameters:

- **dev** pointer to a TPM2_DEV struct
- **parentKey** pointer to a WOLFTPM2_KEY struct, pointing to a Primary Key or TPM Hierarchy
- **keyBlob** pointer to a struct of WOLFTPM2_KEYBLOB type, to import the rsa key to
- **input** buffer holding the rsa der
- **inSz** length of the input der buffer
- **scheme** value of TPMI_ALG_RSA_SCHEME type, specifying the RSA scheme
- **hashAlg** value of TPMI_ALG_HASH type, specifying the TPM hashing algorithm

Return:

- TPM_RC_SUCCESS: successful
- TPM_RC_FAILURE: generic failure (check TPM IO and TPM return code)
- BAD_FUNC_ARG: check the provided arguments

29.9.6.89 function wolfTPM2_RsaPrivateKeyImportPem

```
WOLFTPM_API int wolfTPM2_RsaPrivateKeyImportPem(
    WOLFTPM2_DEV * dev,
    const WOLFTPM2_KEY * parentKey,
    WOLFTPM2_KEYBLOB * keyBlob,
    const char * input,
    word32 inSz,
    char * pass,
    TPMI_ALG_RSA_SCHEME scheme,
    TPMI_ALG_HASH hashAlg
)
```

Helper function to import Pem rsa key directly.

Parameters:

- **dev** pointer to a TPM2_DEV struct
- **parentKey** pointer to a WOLFTPM2_KEY struct, pointing to a Primary Key or TPM Hierarchy
- **keyBlob** pointer to a struct of WOLFTPM2_KEYBLOB type, to import the rsa key to
- **input** buffer holding the rsa pem
- **inSz** length of the input pem buffer
- **pass** optional password of the key
- **scheme** value of TPMI_ALG_RSA_SCHEME type, specifying the RSA scheme
- **hashAlg** value of TPMI_ALG_HASH type, specifying the TPM hashing algorithm

Return:

- TPM_RC_SUCCESS: successful
- TPM_RC_FAILURE: generic failure (check TPM IO and TPM return code)
- BAD_FUNC_ARG: check the provided arguments

29.9.6.90 function wolfTPM2_RsaKey_TpmToWolf

```
WOLFTPM_API int wolfTPM2_RsaKey_TpmToWolf(
    WOLFTPM2_DEV * dev,
    WOLFTPM2_KEY * tpmKey,
    RsaKey * wolfKey
)
```

Extract a RSA TPM key and convert it to a wolfcrypt key.

Parameters:

- **dev** pointer to a TPM2_DEV struct
- **tpmKey** pointer to a struct of WOLFTPM2_KEY type, holding a TPM key
- **wolfKey** pointer to an empty struct of RsaKey type, to store the converted key

See:

- wolfTPM2_RsaKey_WolfToTpm
- wolfTPM2_RsaKey_WolfToTpm_ex

Return:

- TPM_RC_SUCCESS: successful
- TPM_RC_FAILURE: generic failure (check TPM IO and TPM return code)
- BAD_FUNC_ARG: check the provided arguments

29.9.6.91 function wolfTPM2_RsaKey_TpmToPemPub

```
WOLFTPM_API int wolfTPM2_RsaKey_TpmToPemPub(
    WOLFTPM2_DEV * dev,
    WOLFTPM2_KEY * keyBlob,
    byte * pem,
    word32 * pemSz
)
```

Convert a public RSA TPM key to PEM format public key. Note: This API is a wrapper around wolfTPM2_ExportPublicKeyBuffer.

Parameters:

- **dev** pointer to a TPM2_DEV struct
- **keyBlob** pointer to a struct of WOLFTPM2_KEY type, holding a TPM key
- **pem** pointer to an array of byte type, used as temporary storage for PEM conversation
- **pemSz** pointer to integer variable, to store the used buffer size

See:

- wolfTPM2_ExportPublicKeyBuffer
- wolfTPM2_RsaKey_TpmToWolf
- wolfTPM2_RsaKey_WolfToTpm

Return:

- TPM_RC_SUCCESS: successful
- TPM_RC_FAILURE: generic failure (check TPM IO and TPM return code)
- BAD_FUNC_ARG: check the provided arguments

29.9.6.92 function wolfTPM2_RsaKey_WolfToTpm

```
WOLFTPM_API int wolfTPM2_RsaKey_WolfToTpm(
    WOLFTPM2_DEV * dev,
    RsaKey * wolfKey,
    WOLFTPM2_KEY * tpmKey
)
```

Import a RSA wolfcrypt key into the TPM.

Parameters:

- **dev** pointer to a TPM2_DEV struct
- **wolfKey** pointer to a struct of RsaKey type, holding a wolfcrypt key
- **tpmKey** pointer to an empty struct of WOLFTPM2_KEY type, to hold the imported TPM key

See: wolfTPM2_RsaKey_TpmToWolf

Return:

- TPM_RC_SUCCESS: successful
- TPM_RC_FAILURE: generic failure (check TPM IO and TPM return code)
- BAD_FUNC_ARG: check the provided arguments

Note: Allows the use of externally generated keys by wolfcrypt to be used with TPM 2.0

29.9.6.93 function wolfTPM2_RsaKey_WolfToTpm_ex

```
WOLFTPM_API int wolfTPM2_RsaKey_WolfToTpm_ex(
    WOLFTPM2_DEV * dev,
    const WOLFTPM2_KEY * parentKey,
    RsaKey * wolfKey,
    WOLFTPM2_KEY * tpmKey
)
```

Import a RSA wolfcrypt key into the TPM under a specific Primary Key or Hierarchy.

Parameters:

- **dev** pointer to a TPM2_DEV struct
- **parentKey** pointer to a WOLFTPM2_KEY struct, pointing to a Primary Key or TPM Hierarchy
- **wolfKey** pointer to a struct of RsaKey type, holding a wolfcrypt key
- **tpmKey** pointer to an empty struct of WOLFTPM2_KEY type, to hold the imported TPM key

See:

- wolfTPM2_RsaKey_WolfToTpm
- wolfTPM2_RsaKey_TpmToWolf

Return:

- TPM_RC_SUCCESS: successful
- TPM_RC_FAILURE: generic failure (check TPM IO and TPM return code)
- BAD_FUNC_ARG: check the provided arguments

Note: Allows the use of wolfcrypt generated keys with wolfTPM

29.9.6.94 function wolfTPM2_CreateRsaKeyBlob

```
WOLFTPM_API int wolfTPM2_CreateRsaKeyBlob(
    WOLFTPM2_DEV * dev,
    const WOLFTPM2_KEY * parentKey,
    RsaKey * wolfKey,
    WOLFTPM2_KEYBLOB * tpmKey
)
```

Create an encrypted RSA key blob from a wolfCrypt key under a specific parent key.

Parameters:

- **dev** pointer to a TPM2_DEV struct
- **parentKey** pointer to a WOLFTPM2_KEY struct, pointing to a Primary Key or TPM Hierarchy
- **wolfKey** pointer to a struct of RsaKey type, holding a wolfcrypt key
- **tpmKey** pointer to an empty struct of WOLFTPM2_KEYBLOB type, to hold the encrypted key blob

See:

- wolfTPM2_LoadKey
- wolfTPM2_RsaKey_WolfToTpm_ex
- wolfTPM2_CreateEccKeyBlob

Return:

- TPM_RC_SUCCESS: successful
- TPM_RC_FAILURE: generic failure (check TPM IO and TPM return code)
- BAD_FUNC_ARG: check the provided arguments

Note: Creates an encrypted version of the key in WOLFTPM2_KEYBLOB to load the key.

29.9.6.95 function wolfTPM2_RsaKey_PubPemToTpm

```
WOLFTPM_API int wolfTPM2_RsaKey_PubPemToTpm(
    WOLFTPM2_DEV * dev,
    WOLFTPM2_KEY * tpmKey,
    const byte * pem,
    word32 pemSz
)
```

Import a PEM format public key from a file into the TPM.

Parameters:

- **dev** pointer to a TPM2_DEV struct
- **tpmKey** pointer to an empty struct of WOLFTPM2_KEY type, to hold the imported TPM key
- **pem** pointer to an array of byte type, containing a PEM formatted public key material
- **pemSz** pointer to integer variable, specifying the size of PEM key data

See:

- wolfTPM2_RsaKey_WolfToTpm
- wolfTPM2_RsaKey_TpmToPem
- wolfTPM2_RsaKey_TpmToWolf

Return:

- TPM_RC_SUCCESS: successful
- TPM_RC_FAILURE: generic failure (check TPM IO and TPM return code)

29.9.6.96 function wolfTPM2_DecodeRsaDer

```
WOLFTPM_API int wolfTPM2_DecodeRsaDer(
    const byte * der,
    word32 derSz,
    TPM2B_PUBLIC * pub,
    TPM2B_SENSITIVE * sens,
    TPMA_OBJECT attributes
)
```

Import DER RSA private or public key into TPM public and sensitive structures. This does not make any calls to TPM hardware.

Parameters:

- **der** The der encoding of the content of the extension.
- **derSz** The size in bytes of the der encoding.
- **pub** pointer to a populated structure of TPM2B_PUBLIC type
- **sens** pointer to a populated structure of TPM2B_SENSITIVE type
- **attributes** integer value of TPMA_OBJECT type, can contain one or more attributes, e.g. TPMA_OBJECT_fixedTPM (or 0 to automatically populate)

See:

- wolfTPM2_ImportPublicKeyBuffer
- wolfTPM2_ImportPrivateKeyBuffer
- wolfTPM2_DecodeEccDer

Return:

- TPM_RC_SUCCESS: successful
- TPM_RC_FAILURE: generic failure (check TPM IO and TPM return code)

29.9.6.97 function wolfTPM2_EccKey_TpmToWolf

```
WOLFTPM_API int wolfTPM2_EccKey_TpmToWolf(
    WOLFTPM2_DEV * dev,
    WOLFTPM2_KEY * tpmKey,
    ecc_key * wolfKey
)
```

Extract a ECC TPM key and convert to to a wolfcrypt key.

Parameters:

- **dev** pointer to a TPM2_DEV struct
- **tpmKey** pointer to a struct of WOLFTPM2_KEY type, holding a TPM key
- **wolfKey** pointer to an empty struct of ecc_key type, to store the converted key

See:

- wolfTPM2_EccKey_WolfToTpm
- wolfTPM2_EccKey_WolfToTpm_ex

Return:

- TPM_RC_SUCCESS: successful

- TPM_RC_FAILURE: generic failure (check TPM IO and TPM return code)
- BAD_FUNC_ARG: check the provided arguments

29.9.6.98 function wolfTPM2_EccKey_WolfToTpm

```
WOLFTPM_API int wolfTPM2_EccKey_WolfToTpm(
    WOLFTPM2_DEV * dev,
    ecc_key * wolfKey,
    WOLFTPM2_KEY * tpmKey
)
```

Import a ECC wolfcrypt key into the TPM.

Parameters:

- **dev** pointer to a TPM2_DEV struct
- **wolfKey** pointer to a struct of ecc_key type, holding a wolfcrypt key
- **tpmKey** pointer to an empty struct of WOLFTPM2_KEY type, to hold the imported TPM key

See: wolfTPM2_EccKey_TpmToWolf

Return:

- TPM_RC_SUCCESS: successful
- TPM_RC_FAILURE: generic failure (check TPM IO and TPM return code)
- BAD_FUNC_ARG: check the provided arguments

Note: Allows the use of externally generated keys by wolfcrypt to be used with TPM 2.0

29.9.6.99 function wolfTPM2_EccKey_WolfToTpm_ex

```
WOLFTPM_API int wolfTPM2_EccKey_WolfToTpm_ex(
    WOLFTPM2_DEV * dev,
    WOLFTPM2_KEY * parentKey,
    ecc_key * wolfKey,
    WOLFTPM2_KEY * tpmKey
)
```

Import ECC wolfcrypt key into the TPM under a specific Primary Key or Hierarchy.

Parameters:

- **dev** pointer to a TPM2_DEV struct
- **parentKey** pointer to a WOLFTPM2_KEY struct, pointing to a Primary Key or TPM Hierarchy
- **wolfKey** pointer to a struct of ecc_key type, holding a wolfcrypt key
- **tpmKey** pointer to an empty struct of WOLFTPM2_KEY type, to hold the imported TPM key

See:

- wolfTPM2_EccKey_WolfToTPM
- wolfTPM2_EccKey_TpmToWolf

Return:

- TPM_RC_SUCCESS: successful
- TPM_RC_FAILURE: generic failure (check TPM IO and TPM return code)
- BAD_FUNC_ARG: check the provided arguments

Note: Allows the use of wolfcrypt generated keys with wolfTPM

29.9.6.100 function wolfTPM2_CreateEccKeyBlob

```
WOLFTPM_API int wolfTPM2_CreateEccKeyBlob(
    WOLFTPM2_DEV * dev,
    WOLFTPM2_KEY * parentKey,
    ecc_key * wolfKey,
    WOLFTPM2_KEYBLOB * tpmKey
)
```

Create an encrypted ECC key blob from a wolfCrypt key under a specific parent key.

Parameters:

- **dev** pointer to a TPM2_DEV struct
- **parentKey** pointer to a WOLFTPM2_KEY struct, pointing to a Primary Key or TPM Hierarchy
- **wolfKey** pointer to a struct of ecc_key type, holding a wolfcrypt key
- **tpmKey** pointer to an empty struct of WOLFTPM2_KEYBLOB type, to hold the encrypted key blob

See:

- wolfTPM2_LoadKey
- wolfTPM2_EccKey_WolfToTpm_ex
- wolfTPM2_CreateRsaKeyBlob

Return:

- TPM_RC_SUCCESS: successful
- TPM_RC_FAILURE: generic failure (check TPM IO and TPM return code)
- BAD_FUNC_ARG: check the provided arguments

Note: Creates an encrypted version of the key in WOLFTPM2_KEYBLOB to load the key.

29.9.6.101 function wolfTPM2_EccKey_WolfToPubPoint

```
WOLFTPM_API int wolfTPM2_EccKey_WolfToPubPoint(
    WOLFTPM2_DEV * dev,
    ecc_key * wolfKey,
    TPM2B_ECC_POINT * pubPoint
)
```

Import a ECC public key generated from wolfcrypt key into the TPM.

Parameters:

- **dev** pointer to a TPM2_DEV struct
- **wolfKey** pointer to a struct of ecc_key type, holding a wolfcrypt public ECC key
- **pubPoint** pointer to an empty struct of TPM2B_ECC_POINT type

See: wolfTPM2_EccKey_TpmToWolf

Return:

- TPM_RC_SUCCESS: successful
- TPM_RC_FAILURE: generic failure (check TPM IO and TPM return code)
- BAD_FUNC_ARG: check the provided arguments

Note: Allows the use of externally generated public ECC key by wolfcrypt to be used with TPM 2.0

29.9.6.102 function wolfTPM2_DecodeEccDer

```
WOLFTPM_API int wolfTPM2_DecodeEccDer(
    const byte * der,
```

```
    word32 derSz,
    TPM2B_PUBLIC * pub,
    TPM2B_SENSITIVE * sens,
    TPMA_OBJECT attributes
)
```

Import DER ECC private or public key into TPM public and sensitive structures. This does not make any calls to TPM hardware.

Parameters:

- **der** The der encoding of the content of the extension.
- **derSz** The size in bytes of the der encoding.
- **pub** pointer to a populated structure of TPM2B_PUBLIC type
- **sens** pointer to a populated structure of TPM2B_SENSITIVE type
- **attributes** integer value of TPMA_OBJECT type, can contain one or more attributes, e.g. TPMA_OBJECT_fixedTPM (or 0 to automatically populate)

See:

- wolfTPM2_ImportPublicKeyBuffer
- wolfTPM2_ImportPrivateKeyBuffer
- wolfTPM2_DecodeRsaDer

Return:

- TPM_RC_SUCCESS: successful
- TPM_RC_FAILURE: generic failure (check TPM IO and TPM return code)

29.9.6.103 function wolfTPM2_SignHash

```
WOLFTPM_API int wolfTPM2_SignHash(
    WOLFTPM2_DEV * dev,
    WOLFTPM2_KEY * key,
    const byte * digest,
    int digestSz,
    byte * sig,
    int * sigSz
)
```

Helper function to sign arbitrary data using a TPM key.

Parameters:

- **dev** pointer to a TPM2_DEV struct
- **key** pointer to a struct of WOLFTPM2_KEY type, holding a TPM key material
- **digest** pointer to a byte buffer, containing the arbitrary data
- **digestSz** integer value, specifying the size of the digest buffer, in bytes
- **sig** pointer to a byte buffer, containing the generated signature
- **sigSz** integer value, specifying the size of the signature buffer, in bytes

See:

- wolfTPM2_VerifyHash
- wolfTPM2_SignHashScheme
- wolfTPM2_VerifyHashScheme

Return:

- TPM_RC_SUCCESS: successful
- TPM_RC_FAILURE: generic failure (check TPM IO and TPM return code)

- BAD_FUNC_ARG: check the provided arguments

29.9.6.104 function wolfTPM2_SignHashScheme

```
WOLFTPM_API int wolfTPM2_SignHashScheme(
    WOLFTPM2_DEV * dev,
    WOLFTPM2_KEY * key,
    const byte * digest,
    int digestSz,
    byte * sig,
    int * sigSz,
    TPMI_ALG_SIG_SCHEME sigAlg,
    TPMI_ALG_HASH hashAlg
)
```

Advanced helper function to sign arbitrary data using a TPM key, and specify the signature scheme and hashing algorithm.

Parameters:

- **dev** pointer to a TPM2_DEV struct
- **key** pointer to a struct of WOLFTPM2_KEY type, holding a TPM key material
- **digest** pointer to a byte buffer, containing the arbitrary data
- **digestSz** integer value, specifying the size of the digest buffer, in bytes
- **sig** pointer to a byte buffer, containing the generated signature
- **sigSz** integer value, specifying the size of the signature buffer, in bytes
- **sigAlg** integer value of TPMI_ALG_SIG_SCHEME type, specifying a supported TPM 2.0 signature scheme
- **hashAlg** integer value of TPMI_ALG_HASH type, specifying a supported TPM 2.0 hash algorithm

See:

- wolfTPM2_SignHash
- wolfTPM2_VerifyHash
- wolfTPM2_VerifyHashScheme

Return:

- TPM_RC_SUCCESS: successful
- TPM_RC_FAILURE: generic failure (check TPM IO and TPM return code)
- BAD_FUNC_ARG: check the provided arguments

Note: ECC signatures contain fixed-width R followed by S, each padded to the signing key curve size.

29.9.6.105 function wolfTPM2_VerifyHash

```
WOLFTPM_API int wolfTPM2_VerifyHash(
    WOLFTPM2_DEV * dev,
    WOLFTPM2_KEY * key,
    const byte * sig,
    int sigSz,
    const byte * digest,
    int digestSz
)
```

Helper function to verify a TPM generated signature.

Parameters:

- **dev** pointer to a TPM2_DEV struct

- **key** pointer to a struct of WOLFTPM2_KEY type, holding a TPM 2.0 key material
- **sig** pointer to a byte buffer, containing the generated signature
- **sigSz** integer value, specifying the size of the signature buffer, in bytes
- **digest** pointer to a byte buffer, containing the signed data
- **digestSz** integer value, specifying the size of the digest buffer, in bytes

See:

- wolfTPM2_SignHash
- wolfTPM2_SignHashScheme
- wolfTPM2_VerifyHashScheme
- wolfTPM2_VerifyHash_ex

Return:

- TPM_RC_SUCCESS: successful
- TPM_RC_FAILURE: generic failure (check TPM IO and TPM return code)
- BAD_FUNC_ARG: check the provided arguments

29.9.6.106 function wolfTPM2_VerifyHash_ex

```
WOLFTPM_API int wolfTPM2_VerifyHash_ex(
    WOLFTPM2_DEV * dev,
    WOLFTPM2_KEY * key,
    const byte * sig,
    int sigSz,
    const byte * digest,
    int digestSz,
    int hashAlg
)
```

Helper function to verify a TPM generated signature.

Parameters:

- **dev** pointer to a TPM2_DEV struct
- **key** pointer to a struct of WOLFTPM2_KEY type, holding a TPM 2.0 key material
- **sig** pointer to a byte buffer, containing the generated signature
- **sigSz** integer value, specifying the size of the signature buffer, in bytes
- **digest** pointer to a byte buffer, containing the signed data
- **digestSz** integer value, specifying the size of the digest buffer, in bytes
- **hashAlg** hash algorithm used to sign

See:

- wolfTPM2_SignHash
- wolfTPM2_SignHashScheme
- wolfTPM2_VerifyHashScheme

Return:

- TPM_RC_SUCCESS: successful
- TPM_RC_FAILURE: generic failure (check TPM IO and TPM return code)
- BAD_FUNC_ARG: check the provided arguments

29.9.6.107 function wolfTPM2_VerifyHashScheme

```
WOLFTPM_API int wolfTPM2_VerifyHashScheme(
    WOLFTPM2_DEV * dev,
    WOLFTPM2_KEY * key,
```

```
    const byte * sig,
    int sigSz,
    const byte * digest,
    int digestSz,
    TPMI_ALG_SIG_SCHEME sigAlg,
    TPMI_ALG_HASH hashAlg
)
```

Advanced helper function to verify a TPM generated signature.

Parameters:

- **dev** pointer to a TPM2_DEV struct
- **key** pointer to a struct of WOLFTPM2_KEY type, holding a TPM 2.0 key material
- **sig** pointer to a byte buffer, containing the generated signature
- **sigSz** integer value, specifying the size of the signature buffer, in bytes
- **digest** pointer to a byte buffer, containing the signed data
- **digestSz** integer value, specifying the size of the digest buffer, in bytes
- **sigAlg** integer value of TPMI_ALG_SIG_SCHEME type, specifying a supported TPM 2.0 signature scheme
- **hashAlg** integer value of TPMI_ALG_HASH type, specifying a supported TPM 2.0 hash algorithm

See:

- wolfTPM2_SignHash
- wolfTPM2_SignHashScheme
- wolfTPM2_VerifyHash

Return:

- TPM_RC_SUCCESS: successful
- TPM_RC_FAILURE: generic failure (check TPM IO and TPM return code)
- BAD_FUNC_ARG: check the provided arguments

29.9.6.108 function wolfTPM2_VerifyHashTicket

```
WOLFTPM_API int wolfTPM2_VerifyHashTicket(
    WOLFTPM2_DEV * dev,
    WOLFTPM2_KEY * key,
    const byte * sig,
    int sigSz,
    const byte * digest,
    int digestSz,
    TPMI_ALG_SIG_SCHEME sigAlg,
    TPMI_ALG_HASH hashAlg,
    TPMT_TK_VERIFIED * checkTicket
)
```

Advanced helper function to verify a TPM generated signature and return ticket.

Parameters:

- **dev** pointer to a TPM2_DEV struct
- **key** pointer to a struct of WOLFTPM2_KEY type, holding a TPM 2.0 key material
- **sig** pointer to a byte buffer, containing the generated signature
- **sigSz** integer value, specifying the size of the signature buffer, in bytes
- **digest** pointer to a byte buffer, containing the signed data
- **digestSz** integer value, specifying the size of the digest buffer, in bytes

- **sigAlg** integer value of TPMI_ALG_SIG_SCHEME type, specifying a supported TPM 2.0 signature scheme
- **hashAlg** integer value of TPMI_ALG_HASH type, specifying a supported TPM 2.0 hash algorithm
- **checkTicket** returns the validation ticket proving the signature for digest was checked

See:

- wolfTPM2_VerifyHashScheme
- wolfTPM2_VerifyHashTicket
- wolfTPM2_VerifyHash

Return:

- TPM_RC_SUCCESS: successful
- TPM_RC_FAILURE: generic failure (check TPM IO and TPM return code)
- BAD_FUNC_ARG: check the provided arguments

29.9.6.109 function wolfTPM2_SignSequenceStart

```
WOLFTPM_API int wolfTPM2_SignSequenceStart(
    WOLFTPM2_DEV * dev,
    WOLFTPM2_KEY * key,
    const byte * context,
    int contextSz,
    TPM_HANDLE * sequenceHandle
)
```

Post-Quantum Cryptography: Start a signing sequence with context support.

Parameters:

- **dev** Device structure
- **key** Signing key
- **context** Signature context (for PQ algorithms like ML-DSA)
- **contextSz** Size of context buffer
- **sequenceHandle** Output sequence handle

See:

- wolfTPM2_SignSequenceUpdate
- wolfTPM2_SignSequenceComplete
- wolfTPM2_VerifySequenceStart

Return: 0 on success, negative on error

Example

```
WOLFTPM2_DEV dev;
WOLFTPM2_KEY key;
TPM_HANDLE sequenceHandle;
byte context[1024];
int contextSz = sizeof(context);

// Initialize and set up key...
rc = wolfTPM2_SignSequenceStart(&dev, &key, context, contextSz,
 ↪  &sequenceHandle);
```

29.9.6.110 function wolfTPM2_SignSequenceUpdate

```
WOLFTPM_API int wolfTPM2_SignSequenceUpdate(
    WOLFTPM2_DEV * dev,
    TPM_HANDLE sequenceHandle,
    const byte * data,
    int dataSz
)
```

Post-Quantum Cryptography: Update a signing sequence with data.

Parameters:

- **dev** Device structure
- **sequenceHandle** Sequence handle from SignSequenceStart
- **data** Data to add to sequence
- **dataSz** Size of data buffer

See:

- wolfTPM2_SignSequenceStart
- wolfTPM2_SignSequenceComplete

Return: 0 on success, negative on error

Example

```
byte data[256];
rc = wolfTPM2_SignSequenceUpdate(&dev, sequenceHandle, data, sizeof(data));
```

29.9.6.111 function wolfTPM2_SignSequenceComplete

```
WOLFTPM_API int wolfTPM2_SignSequenceComplete(
    WOLFTPM2_DEV * dev,
    TPM_HANDLE sequenceHandle,
    WOLFTPM2_KEY * key,
    const byte * data,
    int dataSz,
    byte * sig,
    int * sigSz
)
```

Post-Quantum Cryptography: Complete a signing sequence.

Parameters:

- **dev** Device structure
- **sequenceHandle** Sequence handle from SignSequenceStart
- **key** Signing key
- **data** Final data to add to sequence
- **dataSz** Size of data buffer
- **sig** Output signature buffer
- **sigSz** Input/output signature size

See:

- wolfTPM2_SignSequenceStart
- wolfTPM2_SignSequenceUpdate

Return: 0 on success, negative on error

Note: ECC signatures contain fixed-width R followed by S, each padded to the signing key curve size.

Example

```
byte sig[2048];
int sigSz = sizeof(sig);
rc = wolfTPM2_SignSequenceComplete(&dev, sequenceHandle, &key, data, dataSz,
 ↪  sig, &sigSz);
```

29.9.6.112 function wolfTPM2_VerifySequenceStart

```
WOLFTPM_API int wolfTPM2_VerifySequenceStart(
    WOLFTPM2_DEV * dev,
    WOLFTPM2_KEY * key,
    const byte * context,
    int contextSz,
    TPM_HANDLE * sequenceHandle
)
```

Post-Quantum Cryptography: Start a verification sequence with context support.

Parameters:

- **dev** Device structure
- **key** Verification key
- **context** Signature context (for PQ algorithms like ML-DSA)
- **contextSz** Size of context buffer
- **sequenceHandle** Output sequence handle

See:

- wolfTPM2_VerifySequenceUpdate
- wolfTPM2_VerifySequenceComplete
- wolfTPM2_SignSequenceStart

Return: 0 on success, negative on error

Example

```
TPM_HANDLE sequenceHandle;
byte context[1024];
rc = wolfTPM2_VerifySequenceStart(&dev, &key, context, sizeof(context),
 ↪  &sequenceHandle);
```

29.9.6.113 function wolfTPM2_VerifySequenceUpdate

```
WOLFTPM_API int wolfTPM2_VerifySequenceUpdate(
    WOLFTPM2_DEV * dev,
    TPM_HANDLE sequenceHandle,
    const byte * data,
    int dataSz
)
```

Post-Quantum Cryptography: Update a verification sequence with data.

Parameters:

- **dev** Device structure
- **sequenceHandle** Sequence handle from VerifySequenceStart
- **data** Data to add to sequence
- **dataSz** Size of data buffer

See:

- wolfTPM2_VerifySequenceStart

- wolfTPM2_VerifySequenceComplete

Return: 0 on success, negative on error

Example

```
rc = wolfTPM2_VerifySequenceUpdate(&dev, sequenceHandle, data, sizeof(data));
```

29.9.6.114 function wolfTPM2_VerifySequenceComplete

```
WOLFTPM_API int wolfTPM2_VerifySequenceComplete(
    WOLFTPM2_DEV * dev,
    TPM_HANDLE sequenceHandle,
    WOLFTPM2_KEY * key,
    const byte * data,
    int dataSz,
    const byte * sig,
    int sigSz,
    TPMT_TK_VERIFIED * validation
)
```

Post-Quantum Cryptography: Complete a verification sequence.

Parameters:

- **dev** Device structure
- **sequenceHandle** Sequence handle from VerifySequenceStart
- **key** Verification key
- **data** Optional final chunk of message data; if non-NULL it is folded into the sequence via an internal TPM2_SequenceUpdate before the Complete is sent (Part 3 Sec.20.3 — the wire command itself only carries the signature; the message must already be accumulated in the sequence object on the TPM).
- **dataSz** Size of data buffer; pass 0 to skip the internal update.
- **sig** Signature to verify
- **sigSz** Size of signature buffer
- **validation** Optional output validation ticket

See:

- wolfTPM2_VerifySequenceStart
- wolfTPM2_VerifySequenceUpdate

Return: 0 on success, negative on error

Example

```
TPMT_TK_VERIFIED validation;
rc = wolfTPM2_VerifySequenceComplete(&dev, sequenceHandle, &key, data, dataSz,
 ↪  sig, sigSz, &validation);
```

29.9.6.115 function wolfTPM2_SignDigest

```
WOLFTPM_API int wolfTPM2_SignDigest(
    WOLFTPM2_DEV * dev,
    WOLFTPM2_KEY * key,
    const byte * digest,
    int digestSz,
    const byte * context,
    int contextSz,
    byte * sig,
```

```
    int * sigSz
)
```

Post-Quantum Cryptography: Sign a digest with context support.

Parameters:

- **dev** Device structure
- **key** Signing key
- **digest** Digest to sign
- **digestSz** Size of digest buffer
- **context** Signature context (for PQ algorithms like ML-DSA)
- **contextSz** Size of context buffer
- **sig** Output signature buffer
- **sigSz** Input/output signature size

See:

- wolfTPM2_VerifyDigestSignature
- wolfTPM2_SignHash

Return: 0 on success, negative on error

Note: ECC signatures contain fixed-width R followed by S, each padded to the signing key curve size.

Example

```
byte digest[64], sig[2048];
int sigSz = sizeof(sig);
byte context[1024];
rc = wolfTPM2_SignDigest(&dev, &key, digest, sizeof(digest), context,
 ↪  sizeof(context), sig, &sigSz);
```

29.9.6.116 function wolfTPM2_VerifyDigestSignature

```
WOLFTPM_API int wolfTPM2_VerifyDigestSignature(
    WOLFTPM2_DEV * dev,
    WOLFTPM2_KEY * key,
    const byte * digest,
    int digestSz,
    const byte * sig,
    int sigSz,
    const byte * context,
    int contextSz,
    TPMT_TK_VERIFIED * validation
)
```

Post-Quantum Cryptography: Verify a digest signature with context support.

Parameters:

- **dev** Device structure
- **key** Verification key
- **digest** Digest that was signed
- **digestSz** Size of digest buffer
- **sig** Signature to verify
- **sigSz** Size of signature buffer
- **context** Signature context (for PQ algorithms like ML-DSA)
- **contextSz** Size of context buffer
- **validation** Optional output validation ticket

See:

- wolfTPM2_SignDigest
- wolfTPM2_VerifyHash

Return: 0 on success, negative on error

Example

```
TPMT_TK_VERIFIED validation;
rc = wolfTPM2_VerifyDigestSignature(&dev, &key, digest, sizeof(digest), sig,
 ↪  sigSz, context, sizeof(context), &validation);
```

29.9.6.117 function wolfTPM2_Encapsulate

```
WOLFTPM_API int wolfTPM2_Encapsulate(
    WOLFTPM2_DEV * dev,
    WOLFTPM2_KEY * key,
    byte * ciphertext,
    int * ciphertextSz,
    byte * sharedSecret,
    int * sharedSecretSz
)
```

Post-Quantum Cryptography: KEM Encapsulate (public key operation)

Parameters:

- **dev** Device structure
- **key** Public key for encapsulation
- **ciphertext** Output ciphertext buffer
- **ciphertextSz** Input/output ciphertext size
- **sharedSecret** Output shared secret buffer
- **sharedSecretSz** Input/output shared secret size

See: wolfTPM2_Decapsulate

Return: 0 on success, negative on error

Example

```
byte ciphertext[2048], sharedSecret[64];
int ciphertextSz = sizeof(ciphertext);
int sharedSecretSz = sizeof(sharedSecret);
rc = wolfTPM2_Encapsulate(&dev, &key, ciphertext, &ciphertextSz, sharedSecret,
 ↪  &sharedSecretSz);
```

29.9.6.118 function wolfTPM2_Decapsulate

```
WOLFTPM_API int wolfTPM2_Decapsulate(
    WOLFTPM2_DEV * dev,
    WOLFTPM2_KEY * key,
    const byte * ciphertext,
    int ciphertextSz,
    byte * sharedSecret,
    int * sharedSecretSz
)
```

Post-Quantum Cryptography: KEM Decapsulate (private key operation)

Parameters:

- **dev** Device structure
- **key** Private key for decapsulation
- **ciphertext** Ciphertext to decapsulate
- **ciphertextSz** Size of ciphertext buffer
- **sharedSecret** Output shared secret buffer
- **sharedSecretSz** Input/output shared secret size

See: wolfTPM2_Encapsulate

Return: 0 on success, negative on error

Example

```
byte sharedSecret[64];
int sharedSecretSz = sizeof(sharedSecret);
rc = wolfTPM2_Decapsulate(&dev, &key, ciphertext, ciphertextSz, sharedSecret,
 ↪  &sharedSecretSz);
```

29.9.6.119 function wolfTPM2_GetKeyTemplate_MLDSA

```
WOLFTPM_API int wolfTPM2_GetKeyTemplate_MLDSA(
    TPMT_PUBLIC * publicTemplate,
    TPMA_OBJECT objectAttributes,
    TPMI_MLDSA_PARAMETER_SET parameterSet,
    int allowExternalMu
)
```

Create a key template for ML-DSA signing keys (TCG v185)

Parameters:

- **publicTemplate** Pointer to TPMT_PUBLIC structure to populate
- **objectAttributes** Key attributes (sign flag will be enforced, decrypt cleared)
- **parameterSet** ML-DSA parameter set (TPM_MLDSA_44, TPM_MLDSA_65, or TPM_MLDSA_87)
- **allowExternalMu** Non-zero to allow TPM2_SignDigest/VerifyDigestSignature

See:

- wolfTPM2_GetKeyTemplate_HASH_MLDSA
- wolfTPM2_GetKeyTemplate_MLKEM

Return: TPM_RC_SUCCESS on success, BAD_FUNC_ARG if publicTemplate is NULL

29.9.6.120 function wolfTPM2_GetKeyTemplate_MLDSA_ex

```
WOLFTPM_API int wolfTPM2_GetKeyTemplate_MLDSA_ex(
    TPMT_PUBLIC * publicTemplate,
    TPMA_OBJECT objectAttributes,
    TPMI_MLDSA_PARAMETER_SET parameterSet,
    int allowExternalMu,
    TPMI_ALG_HASH nameAlg
)
```

Create an ML-DSA key template selecting the name algorithm (TCG v185)

Parameters:

- **publicTemplate** Pointer to TPMT_PUBLIC structure to populate
- **objectAttributes** Key attributes (sign flag will be enforced, decrypt cleared)
- **parameterSet** ML-DSA parameter set (TPM_MLDSA_44, TPM_MLDSA_65, or TPM_MLDSA_87)
- **allowExternalMu** Non-zero to allow TPM2_SignDigest/VerifyDigestSignature

- **nameAlg** Object name algorithm (e.g. TPM_ALG_SHA512). TPM_ALG_NULL selects the wrapper default.

See: wolfTPM2_GetKeyTemplate_MLDSA

Return: TPM_RC_SUCCESS on success, BAD_FUNC_ARG if publicTemplate is NULL

29.9.6.121 function wolfTPM2_GetKeyTemplate_HASH_MLDSA

```
WOLFTPM_API int wolfTPM2_GetKeyTemplate_HASH_MLDSA(
    TPMT_PUBLIC * publicTemplate,
    TPMA_OBJECT objectAttributes,
    TPMI_MLDSA_PARAMETER_SET parameterSet,
    TPMI_ALG_HASH hashAlg
)
```

Create a key template for Pre-Hash ML-DSA signing keys (TCG v185)

Parameters:

- **publicTemplate** Pointer to TPMT_PUBLIC structure to populate
- **objectAttributes** Key attributes (sign flag will be enforced, decrypt cleared)
- **parameterSet** ML-DSA parameter set (TPM_MLDSA_44, TPM_MLDSA_65, or TPM_MLDSA_87)
- **hashAlg** Pre-hash algorithm (e.g., TPM_ALG_SHA256)

See:

- wolfTPM2_GetKeyTemplate_MLDSA
- wolfTPM2_GetKeyTemplate_MLKEM

Return: TPM_RC_SUCCESS on success, BAD_FUNC_ARG if publicTemplate is NULL

29.9.6.122 function wolfTPM2_GetKeyTemplate_HASH_MLDSA_ex

```
WOLFTPM_API int wolfTPM2_GetKeyTemplate_HASH_MLDSA_ex(
    TPMT_PUBLIC * publicTemplate,
    TPMA_OBJECT objectAttributes,
    TPMI_MLDSA_PARAMETER_SET parameterSet,
    TPMI_ALG_HASH hashAlg,
    TPMI_ALG_HASH nameAlg
)
```

Create a Pre-Hash ML-DSA key template selecting the name algorithm (TCG v185)

Parameters:

- **publicTemplate** Pointer to TPMT_PUBLIC structure to populate
- **objectAttributes** Key attributes (sign flag will be enforced, decrypt cleared)
- **parameterSet** ML-DSA parameter set (TPM_MLDSA_44, TPM_MLDSA_65, or TPM_MLDSA_87)
- **hashAlg** Pre-hash algorithm (e.g., TPM_ALG_SHA256)
- **nameAlg** Object name algorithm (e.g. TPM_ALG_SHA512). TPM_ALG_NULL selects the wrapper default.

See: wolfTPM2_GetKeyTemplate_HASH_MLDSA

Return: TPM_RC_SUCCESS on success, BAD_FUNC_ARG if publicTemplate is NULL

29.9.6.123 function wolfTPM2_GetKeyTemplate_MLKEM

```
WOLFTPM_API int wolfTPM2_GetKeyTemplate_MLKEM(
    TPMT_PUBLIC * publicTemplate,
    TPMA_OBJECT objectAttributes,
    TPMI_MLKEM_PARAMETER_SET parameterSet
)
```

Create a key template for ML-KEM decryption keys (TCG v185)

Parameters:

- **publicTemplate** Pointer to TPMT_PUBLIC structure to populate
- **objectAttributes** Key attributes (decrypt flag will be enforced, sign cleared)
- **parameterSet** ML-KEM parameter set (TPM_MLKEM_512, TPM_MLKEM_768, or TPM_MLKEM_1024)

See:

- wolfTPM2_GetKeyTemplate_MLDSA
- wolfTPM2_GetKeyTemplate_HASH_MLDSA

Return: TPM_RC_SUCCESS on success, BAD_FUNC_ARG if publicTemplate is NULL

29.9.6.124 function wolfTPM2_GetKeyTemplate_MLKEM_ex

```
WOLFTPM_API int wolfTPM2_GetKeyTemplate_MLKEM_ex(
    TPMT_PUBLIC * publicTemplate,
    TPMA_OBJECT objectAttributes,
    TPMI_MLKEM_PARAMETER_SET parameterSet,
    TPMI_ALG_HASH nameAlg
)
```

Create an ML-KEM key template selecting the name algorithm (TCG v185)

Parameters:

- **publicTemplate** Pointer to TPMT_PUBLIC structure to populate
- **objectAttributes** Key attributes (decrypt flag will be enforced, sign cleared)
- **parameterSet** ML-KEM parameter set (TPM_MLKEM_512, TPM_MLKEM_768, or TPM_MLKEM_1024)
- **nameAlg** Object name algorithm (e.g. TPM_ALG_SHA512). TPM_ALG_NULL selects the wrapper default.

See: wolfTPM2_GetKeyTemplate_MLKEM

Return: TPM_RC_SUCCESS on success, BAD_FUNC_ARG if publicTemplate is NULL

29.9.6.125 function wolfTPM2_ECDHGenKey

```
WOLFTPM_API int wolfTPM2_ECDHGenKey(
    WOLFTPM2_DEV * dev,
    WOLFTPM2_KEY * ecdhKey,
    int curve_id,
    const byte * auth,
    int authSz
)
```

Generates and then loads a ECC key-pair with NULL hierarchy for Diffie-Hellman exchange.

Parameters:

- **dev** pointer to a TPM2_DEV struct
- **ecdhKey** pointer to an empty structure of WOLFTPM2_KEY type
- **curve_id** integer value, specifying a valid TPM_ECC_CURVE value

- **auth** pointer to a string constant, specifying the password authorization for the TPM 2.0 Key
- **authSz** integer value, specifying the size of the password authorization, in bytes

See:

- wolfTPM2_ECDHGen
- wolfTPM2_ECDHGenZ
- wolfTPM2_ECDHEGenKey
- wolfTPM2_ECDHEGenZ

Return:

- TPM_RC_SUCCESS: successful
- TPM_RC_FAILURE: generic failure (check TPM IO and TPM return code)
- BAD_FUNC_ARG: check the provided arguments

29.9.6.126 function wolfTPM2_ECDHGen

```
WOLFTPM_API int wolfTPM2_ECDHGen(
    WOLFTPM2_DEV * dev,
    WOLFTPM2_KEY * privKey,
    TPM2B_ECC_POINT * pubPoint,
    byte * out,
    int * outSz
)
```

Generates ephemeral key and computes Z (shared secret)

Parameters:

- **dev** pointer to a TPM2_DEV struct
- **privKey** pointer to a structure of WOLFTPM2_KEY type
- **pubPoint** pointer to an empty structure of TPM2B_ECC_POINT type
- **out** pointer to a byte buffer, to store the generated shared secret
- **outSz** integer value, specifying the size of the shared secret, in bytes

See:

- wolfTPM2_ECDHGenZ
- wolfTPM2_ECDHGenKey
- wolfTPM2_ECDHEGenKey
- wolfTPM2_ECDHEGenZ

Return:

- TPM_RC_SUCCESS: successful
- TPM_RC_FAILURE: generic failure (check TPM IO and TPM return code)
- BAD_FUNC_ARG: check the provided arguments

Note: One shot API using private key handle to generate key-pair and return public point and shared secret

29.9.6.127 function wolfTPM2_ECDHGenZ

```
WOLFTPM_API int wolfTPM2_ECDHGenZ(
    WOLFTPM2_DEV * dev,
    WOLFTPM2_KEY * privKey,
    const TPM2B_ECC_POINT * pubPoint,
    byte * out,
```

```
    int * outSz
)
```

Computes Z (shared secret) using pubPoint and loaded private ECC key.

Parameters:

- **dev** pointer to a TPM2_DEV struct
- **privKey** pointer to a structure of WOLFTPM2_KEY type, containing a valid TPM handle
- **pubPoint** pointer to a populated structure of TPM2B_ECC_POINT type
- **out** pointer to a byte buffer, to store the computed shared secret
- **outSz** integer value, specifying the size of the shared secret, in bytes

See:

- wolfTPM2_ECDHGen
- wolfTPM2_ECDHGenKey
- wolfTPM2_ECDHEGenKey
- wolfTPM2_ECDHEGenZ

Return:

- TPM_RC_SUCCESS: successful
- TPM_RC_FAILURE: generic failure (check TPM IO and TPM return code)
- BAD_FUNC_ARG: check the provided arguments

29.9.6.128 function wolfTPM2_ECDHEGenKey

```
WOLFTPM_API int wolfTPM2_ECDHEGenKey(
    WOLFTPM2_DEV * dev,
    WOLFTPM2_KEY * ecdhKey,
    int curve_id
)
```

Generates ephemeral ECC key and returns array index (2 phase method)

Parameters:

- **dev** pointer to a TPM2_DEV struct
- **ecdhKey** pointer to an empty structure of WOLFTPM2_KEY type
- **curve_id** integer value, specifying a valid TPM_ECC_CURVE value

See:

- wolfTPM2_ECDHEGenZ
- wolfTPM2_ECDHGen
- wolfTPM2_ECDHGenKey
- wolfTPM2_ECDHGenZ

Return:

- TPM_RC_SUCCESS: successful
- TPM_RC_FAILURE: generic failure (check TPM IO and TPM return code)
- BAD_FUNC_ARG: check the provided arguments

Note: One time use key

29.9.6.129 function wolfTPM2_ECDHEGenZ

```
WOLFTPM_API int wolfTPM2_ECDHEGenZ(
    WOLFTPM2_DEV * dev,
    WOLFTPM2_KEY * parentKey,
```

```
    WOLFTPM2_KEY * ecdhKey,
    const TPM2B_ECC_POINT * pubPoint,
    byte * out,
    int * outSz
)
```

Computes Z (shared secret) using pubPoint and counter (2 phase method)

Parameters:

- **dev** pointer to a TPM2_DEV struct
- **parentKey** pointer to a structure of WOLFTPM2_KEY type, containing a valid TPM handle of a primary key
- **ecdhKey** pointer to a structure of WOLFTPM2_KEY type, containing a valid TPM handle
- **pubPoint** pointer to an empty struct of TPM2B_ECC_POINT type
- **out** pointer to a byte buffer, to store the computed shared secret
- **outSz** integer value, specifying the size of the shared secret, in bytes

See:

- wolfTPM2_ECDHEGenKey
- wolfTPM2_ECDHGen
- wolfTPM2_ECDHGenKey
- wolfTPM2_ECDHGenZ

Return:

- TPM_RC_SUCCESS: successful
- TPM_RC_FAILURE: generic failure (check TPM IO and TPM return code)
- BAD_FUNC_ARG: check the provided arguments

Note: The counter, array ID, can only be used one time

29.9.6.130 function wolfTPM2_RsaEncrypt

```
WOLFTPM_API int wolfTPM2_RsaEncrypt(
    WOLFTPM2_DEV * dev,
    WOLFTPM2_KEY * key,
    TPM_ALG_ID padScheme,
    const byte * msg,
    int msgSz,
    byte * out,
    int * outSz
)
```

Perform RSA encryption using a TPM 2.0 key.

Parameters:

- **dev** pointer to a TPM2_DEV struct
- **key** pointer to a struct of WOLFTPM2_KEY type, holding a TPM key material
- **padScheme** integer value of TPM_ALG_ID type, specifying the padding scheme
- **msg** pointer to a byte buffer, containing the arbitrary data for encryption
- **msgSz** integer value, specifying the size of the arbitrary data buffer
- **out** pointer to a byte buffer, where the encrypted data will be stored
- **outSz** integer value, specifying the size of the encrypted data buffer

See: wolfTPM2_RsaDecrypt

Return:

- TPM_RC_SUCCESS: successful
- TPM_RC_FAILURE: generic failure (check TPM IO and TPM return code)
- BAD_FUNC_ARG: check the provided arguments

29.9.6.131 function wolfTPM2_RsaEncrypt_ex

```
WOLFTPM_API int wolfTPM2_RsaEncrypt_ex(
    WOLFTPM2_DEV * dev,
    WOLFTPM2_KEY * key,
    TPM_ALG_ID padScheme,
    const byte * msg,
    int msgSz,
    byte * out,
    int * outSz,
    TPMI_ALG_HASH hashAlg
)
```

Perform RSA encryption selecting the padding scheme hash algorithm.

Parameters:

- **dev** pointer to a TPM2_DEV struct
- **key** pointer to a struct of WOLFTPM2_KEY type, holding a TPM key material
- **padScheme** integer value of TPM_ALG_ID type, specifying the padding scheme
- **msg** pointer to a byte buffer, containing the arbitrary data for encryption
- **msgSz** integer value, specifying the size of the arbitrary data buffer
- **out** pointer to a byte buffer, where the encrypted data will be stored
- **outSz** integer value, specifying the size of the encrypted data buffer
- **hashAlg** hash for the padding scheme (e.g. TPM_ALG_SHA512 for OAEP). TPM_ALG_NULL selects the wrapper default.

See:

- wolfTPM2_RsaEncrypt
- wolfTPM2_RsaDecrypt_ex

Return:

- TPM_RC_SUCCESS: successful
- TPM_RC_FAILURE: generic failure (check TPM IO and TPM return code)
- BAD_FUNC_ARG: check the provided arguments

29.9.6.132 function wolfTPM2_RsaDecrypt

```
WOLFTPM_API int wolfTPM2_RsaDecrypt(
    WOLFTPM2_DEV * dev,
    WOLFTPM2_KEY * key,
    TPM_ALG_ID padScheme,
    const byte * in,
    int inSz,
    byte * msg,
    int * msgSz
)
```

Perform RSA decryption using a TPM 2.0 key.

Parameters:

- **dev** pointer to a TPM2_DEV struct

- **key** pointer to a struct of WOLFTPM2_KEY type, holding a TPM key material
- **padScheme** integer value of TPM_ALG_ID type, specifying the padding scheme
- **in** pointer to a byte buffer, containing the encrypted data
- **inSz** integer value, specifying the size of the encrypted data buffer
- **msg** pointer to a byte buffer, containing the decrypted data
- **msgSz** pointer to size of the encrypted data buffer, on return set actual size

See: wolfTPM2_RsaEncrypt

Return:

- TPM_RC_SUCCESS: successful
- TPM_RC_FAILURE: generic failure (check TPM IO and TPM return code)
- BAD_FUNC_ARG: check the provided arguments

29.9.6.133 function wolfTPM2_RsaDecrypt_ex

```
WOLFTPM_API int wolfTPM2_RsaDecrypt_ex(
    WOLFTPM2_DEV * dev,
    WOLFTPM2_KEY * key,
    TPM_ALG_ID padScheme,
    const byte * in,
    int inSz,
    byte * msg,
    int * msgSz,
    TPMI_ALG_HASH hashAlg
)
```

Perform RSA decryption selecting the padding scheme hash algorithm.

Parameters:

- **dev** pointer to a TPM2_DEV struct
- **key** pointer to a struct of WOLFTPM2_KEY type, holding a TPM key material
- **padScheme** integer value of TPM_ALG_ID type, specifying the padding scheme
- **in** pointer to a byte buffer, containing the encrypted data
- **inSz** integer value, specifying the size of the encrypted data buffer
- **msg** pointer to a byte buffer, containing the decrypted data
- **msgSz** pointer to size of the encrypted data buffer, on return set actual size
- **hashAlg** hash for the padding scheme (e.g. TPM_ALG_SHA512 for OAEP). TPM_ALG_NULL selects the wrapper default.

See:

- wolfTPM2_RsaDecrypt
- wolfTPM2_RsaEncrypt_ex

Return:

- TPM_RC_SUCCESS: successful
- TPM_RC_FAILURE: generic failure (check TPM IO and TPM return code)
- BAD_FUNC_ARG: check the provided arguments

29.9.6.134 function wolfTPM2_ReadPCR

```
WOLFTPM_API int wolfTPM2_ReadPCR(
    WOLFTPM2_DEV * dev,
    int pcrIndex,
    int hashAlg,
```

```
    byte * digest,
    int * pDigestLen
)
```

Read the values of a specified TPM 2.0 Platform Configuration Registers(PCR)

Parameters:

- **dev** pointer to a TPM2_DEV struct
- **pcrIndex** integer value, specifying a valid PCR index, between 0 and 23 (TPM locality could have an impact on successful access)
- **hashAlg** integer value, specifying a TPM_ALG_SHA256 or TPM_ALG_SHA1 registers to be accessed
- **digest** pointer to a byte buffer, where the PCR values will be stored
- **pDigestLen** pointer to an integer variable, where the size of the digest buffer will be stored

See: wolfTPM2_ExtendPCR

Return:

- TPM_RC_SUCCESS: successful
- TPM_RC_FAILURE: generic failure (check TPM IO and TPM return code)
- BAD_FUNC_ARG: check the provided arguments

Note: Make sure to specify the correct hashing algorithm, because there are two sets of PCR registers, one for SHA256 and the other for SHA1(deprecated, but still possible to be read)

29.9.6.135 function wolfTPM2_ResetPCR

```
WOLFTPM_API int wolfTPM2_ResetPCR(
    WOLFTPM2_DEV * dev,
    int pcrIndex
)
```

Reset a PCR register to its default value.

Parameters:

- **dev** pointer to a TPM2_DEV struct
- **pcrIndex** integer value from PCR_FIRST through PCR_LAST

See:

- wolfTPM2_ReadPCR
- wolfTPM2_ExtendPCR

Return:

- TPM_RC_SUCCESS: successful
- TPM_RC_FAILURE: generic failure (check TPM IO and TPM return code)
- BAD_FUNC_ARG: check the provided arguments

Note: Reset permission depends on the selected PCR's TPM attributes and locality.

29.9.6.136 function wolfTPM2_SetLocality

```
WOLFTPM_API int wolfTPM2_SetLocality(
    WOLFTPM2_DEV * dev,
    int locality
)
```

Change the active TPM locality at runtime.

Parameters:

- **dev** pointer to a TPM2_DEV struct
- **locality** integer value, specifying the TPM locality (0 to 4)

See: wolfTPM2_ResetPCR

Return:

- TPM_RC_SUCCESS: locality granted and now active
- other: on the built-in TIS driver any non-success TPM_RC_* code (for example TPM_RC_TIMEOUT) may be returned when the locality could not be granted, so test rc != TPM_RC_SUCCESS rather than a specific code
- NOT_COMPILED_IN: locality switching is not supported for this backend
- BAD_FUNC_ARG: check the provided arguments

Note: Startup uses WOLFTPM_LOCALITY_DEFAULT (0); this switches so subsequent commands run at the given locality (e.g. loc 2/3 to reset PCRs not resettable at loc 0). For the built-in TIS driver it performs the ACCESS handshake (request, then release-and-request if the TPM does not preempt, restoring the old locality on failure); for the SWTPM/mssim transport it records the locality (sent with each command). The Linux kernel driver and Windows TBS own the locality and return NOT_COMPILED_IN. Return to locality 0 when done.

29.9.6.137 function wolfTPM2_ExtendPCR

```
WOLFTPM_API int wolfTPM2_ExtendPCR(
    WOLFTPM2_DEV * dev,
    int pcrIndex,
    int hashAlg,
    const byte * digest,
    int digestLen
)
```

Extend a PCR register with a user provided digest.

Parameters:

- **dev** pointer to a TPM2_DEV struct
- **pcrIndex** integer value, specifying a valid PCR index, between 0 and 23 (TPM locality could have an impact on successful access)
- **hashAlg** integer value, specifying a TPM_ALG_SHA256 or TPM_ALG_SHA1 registers to be accessed
- **digest** pointer to a byte buffer, containing the digest value to be extended into the PCR
- **digestLen** the size of the digest buffer

See: wolfTPM2_ReadPCR

Return:

- TPM_RC_SUCCESS: successful
- TPM_RC_FAILURE: generic failure (check TPM IO and TPM return code)
- BAD_FUNC_ARG: check the provided arguments

Note: Make sure to specify the correct hashing algorithm

29.9.6.138 function wolfTPM2_AllocatePCRBanks

```
WOLFTPM_API int wolfTPM2_AllocatePCRBanks(
    WOLFTPM2_DEV * dev,
```

```
    const TPM_ALG_ID * hashAlgs,
    int hashAlgCount,
    PCR_Allocate_Out * allocOut
)
```

Re-provision which PCR banks the TPM has allocated.

Parameters:

- **dev** pointer to a TPM2_DEV struct
- **hashAlgs** array of TPM_ALG_ID hash algorithms naming the complete new bank set
- **hashAlgCount** number of entries in hashAlgs, 1 to HASH_COUNT; zero is rejected because it would leave the TPM with no PCR banks
- **allocOut** optional, receives the TPM's allocationSuccess, maxPCR, sizeNeeded and sizeAvailable; zeroed on entry

See:

- wolfTPM2_AllocatePCRBanks_ex
- TPM2_IsPcrBankAllocated
- wolfTPM2_ExtendPCR

Return:

- TPM_RC_SUCCESS: the new allocation is staged for the next reset
- BUFFER_E: the TPM has no room for the requested bank set; see allocOut->sizeNeeded and allocOut->sizeAvailable
- TPM_RC_HASH: the TPM does not implement one of the requested algs; nothing was sent and the allocation is unchanged
- TPM_RC_PCR: the TPM refused the selection, typically because it keeps a single bank active and more than one was requested
- TPM_RC_HIERARCHY: the platform hierarchy is disabled
- TPM_RC_BAD_AUTH: platform auth is not the empty password; use wolfTPM2_AllocatePCRBanks_ex with a session
- BAD_FUNC_ARG: check the provided arguments

Note:

- This REPLACES the allocation: hashAlgs is the complete new set and every bank not named in it is deallocated. Allocating SHA-384 alone on a SHA-256 TPM removes the SHA-256 bank. Many parts (Infineon SLB9672 and later) support only one active bank at a time and reject a multi-bank request outright with TPM_RC_PCR. A TPM that answers but lacks the space reports it in allocOut instead, which maps to BUFFER_E.
- Changing banks invalidates every PolicyPCR digest and makes every blob sealed to PCR values unsealable. The PCR contents are zeroed at the Startup(CLEAR) and re-allocating the old bank does not bring them back.
- Success means the change is STAGED. The TPM applies it at the next Startup(CLEAR) following a _TPM_Init, so the caller must power cycle the TPM (or restart the simulator process) and re-read the banks with TPM2_IsPcrBankAllocated to confirm. No TPM command performs that reset.
- Requires the platform hierarchy. Under an OS the platform firmware has usually cleared phEnable, in which case the TPM answers TPM_RC_HIERARCHY no matter what auth is supplied.

29.9.6.139 function wolfTPM2_AllocatePCRBanks_ex

```
WOLFTPM_API int wolfTPM2_AllocatePCRBanks_ex(
    WOLFTPM2_DEV * dev,
    WOLFTPM2_SESSION * session,
    const TPM_ALG_ID * hashAlgs,
    int hashAlgCount,
```

```
    PCR_Allocate_Out * allocOut
)
```

Re-provision the TPM's PCR banks using a caller supplied session.

Parameters:

- **dev** pointer to a TPM2_DEV struct
- **session** pointer to a WOLFTPM2_SESSION for platform auth, or NULL
- **hashAlgs** array of TPM_ALG_ID hash algorithms naming the complete new bank set
- **hashAlgCount** number of entries in hashAlgs, 1 to HASH_COUNT
- **allocOut** optional, receives the TPM's allocation result

See: wolfTPM2_AllocatePCRBanks

Return:

- TPM_RC_SUCCESS: the new allocation is staged for the next reset
- BAD_FUNC_ARG: check the provided arguments

Note:

- Same as wolfTPM2_AllocatePCRBanks, but takes a session for a platform hierarchy that carries an authPolicy. This must be a POLICY session: slot 0 authorization is a password or a policy session, so an HMAC session there is not an authorization session and the TPM answers TPM_RC_AUTH_MISSING. Passing NULL is identical to wolfTPM2_AllocatePCRBanks.
- Session slot 0 is saved and restored around the command, and the response nonce is copied back into the caller's session so it stays usable for a following command.

29.9.6.140 function wolfTPM2_NVCreateAuth

```
WOLFTPM_API int wolfTPM2_NVCreateAuth(
    WOLFTPM2_DEV * dev,
    WOLFTPM2_HANDLE * parent,
    WOLFTPM2_NV * nv,
    word32 nvIndex,
    word32 nvAttributes,
    word32 maxSize,
    const byte * auth,
    int authSz
)
```

Creates a new NV Index to be later used for storing data into the TPM's NVRAM.

Parameters:

- **dev** pointer to a TPM2_DEV struct
- **parent** pointer to a WOLFTPM2_HANDLE, specifying the TPM hierarchy for the new NV Index
- **nv** pointer to an empty structure of WOLFTPM2_NV type, to hold the new NV Index
- **nvIndex** integer value, holding the NV Index Handle given by the TPM upon success
- **nvAttributes** integer value, use wolfTPM2_GetNvAttributesTemplate to create correct value
- **maxSize** integer value, specifying the maximum number of bytes written at this NV Index
- **auth** pointer to a string constant, specifying the password authorization for this NV Index
- **authSz** integer value, specifying the size of the password authorization, in bytes

See:

- wolfTPM2_NVCreateAuthPolicy
- wolfTPM2_NVWriteAuth
- wolfTPM2_NVReadAuth
- wolfTPM2_NVDeleteAuth

- wolfTPM2_NVOpen

Return:

- TPM_RC_SUCCESS: successful
- TPM_RC_FAILURE: generic failure (check TPM IO and TPM return code)
- BAD_FUNC_ARG: check the provided arguments

Note: This is a wolfTPM2 wrapper around TPM2_NV_DefineSpace

29.9.6.141 function wolfTPM2_NVCreateAuthPolicy

```
WOLFTPM_API int wolfTPM2_NVCreateAuthPolicy(
    WOLFTPM2_DEV * dev,
    WOLFTPM2_HANDLE * parent,
    WOLFTPM2_NV * nv,
    word32 nvIndex,
    word32 nvAttributes,
    word32 maxSize,
    const byte * auth,
    int authSz,
    const byte * authPolicy,
    int authPolicySz
)
```

Creates a new NV Index to be later used for storing data into the TPM's NVRAM.

Parameters:

- **dev** pointer to a TPM2_DEV struct
- **parent** pointer to a WOLFTPM2_HANDLE, specifying the TPM hierarchy for the new NV Index
- **nv** pointer to an empty structure of WOLFTPM2_NV type, to hold the new NV Index
- **nvIndex** integer value, holding the NV Index Handle given by the TPM upon success
- **nvAttributes** integer value, use wolfTPM2_GetNvAttributesTemplate to create correct value
- **maxSize** integer value, specifying the maximum number of bytes written at this NV Index
- **auth** pointer to a string constant, specifying the password authorization for this NV Index
- **authSz** integer value, specifying the size of the password authorization, in bytes
- **authPolicy** optional policy for using this key (The policy is computed using the nameAlg of the object)
- **authPolicySz** size of the authPolicy

See:

- wolfTPM2_NVCreateAuth
- wolfTPM2_NVWriteAuth
- wolfTPM2_NVReadAuth
- wolfTPM2_NVDeleteAuth
- wolfTPM2_NVOpen

Return:

- TPM_RC_SUCCESS: successful
- TPM_RC_FAILURE: generic failure (check TPM IO and TPM return code)
- BAD_FUNC_ARG: check the provided arguments

Note: This is a wolfTPM2 wrapper around TPM2_NV_DefineSpace

29.9.6.142 function wolfTPM2_NVCreateAuthPolicy_ex

```
WOLFTPM_API int wolfTPM2_NVCreateAuthPolicy_ex(
    WOLFTPM2_DEV * dev,
    WOLFTPM2_HANDLE * parent,
    WOLFTPM2_NV * nv,
    word32 nvIndex,
    word32 nvAttributes,
    word32 maxSize,
    const byte * auth,
    int authSz,
    const byte * authPolicy,
    int authPolicySz,
    TPMI_ALG_HASH nameAlg
)
```

Variant of wolfTPM2_NVCreateAuthPolicy that lets the caller pass an explicit nameAlg. Required when authPolicy was computed with SM3-256, SHA3-256, or any other non-SHA hash that the size-based legacy inference cannot distinguish from SHA-256.

Parameters:

- **dev** pointer to a TPM2_DEV struct
- **parent** pointer to a WOLFTPM2_HANDLE for the hierarchy
- **nv** pointer to an empty WOLFTPM2_NV that receives the created index
- **nvIndex** NV index handle (e.g. 0x01400001)
- **nvAttributes** TPMA_NV bitmask
- **maxSize** NV index data size in bytes
- **auth** optional NV authValue (NULL/0 for none)
- **authSz** optional NV authValue size
- **authPolicy** optional policy digest (NULL/0 for none)
- **authPolicySz** optional policy digest size
- **nameAlg** index name hash algorithm (e.g. TPM_ALG_SHA384). Pass TPM_ALG_NULL to fall back to size-based inference for SHA-only policies, or to use WOLFTPM2_WRAP_DIGEST for the no-policy case

Return:

- TPM_RC_SUCCESS: successful
- BAD_FUNC_ARG: nameAlg digest size disagrees with authPolicySz, or other invalid arguments
- BUFFER_E: auth or authPolicy buffer too large

29.9.6.143 function wolfTPM2_NVWriteAuth

```
WOLFTPM_API int wolfTPM2_NVWriteAuth(
    WOLFTPM2_DEV * dev,
    WOLFTPM2_NV * nv,
    word32 nvIndex,
    byte * dataBuf,
    word32 dataSz,
    word32 offset
)
```

Stores user data to a NV Index, at a given offset.

Parameters:

- **dev** pointer to a TPM2_DEV struct
- **nv** pointer to a populated structure of WOLFTPM2_NV type
- **nvIndex** integer value, holding an existing NV Index Handle value

- **dataBuf** pointer to a byte buffer, containing the user data to be written to the TPM's NVRAM
- **dataSz** integer value, specifying the size of the user data buffer, in bytes
- **offset** integer value of word32 type, specifying the offset from the NV Index memory start, can be zero

See:

- wolfTPM2_NVReadAuth
- wolfTPM2_NVCreateAuth
- wolfTPM2_NVDeleteAuth
- wolfTPM2_NVWriteAuthPolicy

Return:

- TPM_RC_SUCCESS: successful
- TPM_RC_FAILURE: generic failure (check TPM IO and TPM return code)
- BAD_FUNC_ARG: check the provided arguments

Note:

- User data size should be less or equal to the NV Index maxSize specified using wolfTPM2_CreateAuth
- For an index with TPMA_NV_OWNERWRITE or TPMA_NV_PPWRITE set nv->handle.hndl to TPM_RH_OWNER or TPM_RH_PLATFORM and nv->handle.auth to that hierarchy auth (after wolfTPM2_NVOpen)

29.9.6.144 function wolfTPM2_NVWriteAuthPolicy

```
WOLFTPM_API int wolfTPM2_NVWriteAuthPolicy(
    WOLFTPM2_DEV * dev,
    WOLFTPM2_SESSION * tpmSession,
    TPM_ALG_ID pcrAlg,
    byte * pcrArray,
    word32 pcrArraySz,
    WOLFTPM2_NV * nv,
    word32 nvIndex,
    byte * dataBuf,
    word32 dataSz,
    word32 offset
)
```

Stores user data to a NV Index, at a given offset. Allows using a policy session and PCR's for authentication.

Parameters:

- **dev** pointer to a TPM2_DEV struct
- **tpmSession** pointer to a WOLFTPM2_SESSION struct used with wolfTPM2_StartSession and wolfTPM2_SetAuthSession
- **pcrAlg** the hash algorithm to use with PCR policy
- **pcrArray** array of PCR Indexes to use when creating the policy
- **pcrArraySz** the number of PCR Indexes in the pcrArray
- **nv** pointer to a populated structure of WOLFTPM2_NV type
- **nvIndex** integer value, holding an existing NV Index Handle value
- **dataBuf** pointer to a byte buffer, containing the user data to be written to the TPM's NVRAM
- **dataSz** integer value, specifying the size of the user data buffer, in bytes
- **offset** integer value of word32 type, specifying the offset from the NV Index memory start, can be zero

See:

- wolfTPM2_NVReadAuth
- wolfTPM2_NVCreateAuth
- wolfTPM2_NVDeleteAuth
- wolfTPM2_NVWriteAuth

Return:

- TPM_RC_SUCCESS: successful
- TPM_RC_FAILURE: generic failure (check TPM IO and TPM return code)
- BAD_FUNC_ARG: check the provided arguments
- BUFFER_E: data too large for a policy without PCR's

Note:

- User data size should be less or equal to the NV Index maxSize specified using wolfTPM2_CreateAuth
- For an index with TPMA_NV_OWNERWRITE or TPMA_NV_PPWRITE set nv->handle.hndl to TPM_RH_OWNER or TPM_RH_PLATFORM and nv->handle.auth to that hierarchy auth (after wolfTPM2_NVOpen)
- pcrArray may be NULL with pcrArraySz 0 when the policy session was already satisfied by the caller (e.g. PolicySecret or PolicyCommandCode). The data must then fit in one MAX_NV_BUFFER_SIZE chunk, otherwise BUFFER_E is returned

29.9.6.145 function wolfTPM2_NVExtend

```
WOLFTPM_API int wolfTPM2_NVExtend(
    WOLFTPM2_DEV * dev,
    WOLFTPM2_NV * nv,
    word32 nvIndex,
    byte * dataBuf,
    word32 dataSz
)
```

Extend data to an NV index.

Parameters:

- **dev** pointer to a TPM2_DEV struct
- **nv** pointer to a populated structure of WOLFTPM2_NV type
- **nvIndex** integer value, holding an existing NV Index Handle value
- **dataBuf** pointer to a byte buffer, containing the user data to be written to the TPM's NVRAM
- **dataSz** integer value, specifying the size of the user data buffer, in bytes

See:

- wolfTPM2_NVReadAuth
- wolfTPM2_NVCreateAuth
- wolfTPM2_NVOpen
- wolfTPM2_NVDeleteAuth

Return:

- TPM_RC_SUCCESS: successful
- TPM_RC_FAILURE: generic failure (check TPM IO and TPM return code)
- BAD_FUNC_ARG: check the provided arguments

Note:

- When NV index is read it will return the digest
- For an index with TPMA_NV_OWNERWRITE or TPMA_NV_PPWRITE set nv->handle.hndl to TPM_RH_OWNER or TPM_RH_PLATFORM and nv->handle.auth to that hierarchy auth (after wolfTPM2_NVOpen)

29.9.6.146 function wolfTPM2_NVReadAuth

```
WOLFTPM_API int wolfTPM2_NVReadAuth(
    WOLFTPM2_DEV * dev,
    WOLFTPM2_NV * nv,
    word32 nvIndex,
    byte * dataBuf,
    word32 * pDataSz,
    word32 offset
)
```

Reads user data from a NV Index, starting at the given offset.

Parameters:

- **dev** pointer to a TPM2_DEV struct
- **nv** pointer to a populated structure of WOLFTPM2_NV type
- **nvIndex** integer value, holding an existing NV Index Handle value
- **dataBuf** pointer to an empty byte buffer, used to store the read data from the TPM's NVRAM
- **pDataSz** pointer to an integer variable, used to store the size of the data read from NVRAM, in bytes
- **offset** integer value of word32 type, specifying the offset from the NV Index memory start, can be zero

See:

- wolfTPM2_NVWriteAuth
- wolfTPM2_NVCreateAuth
- wolfTPM2_NVDeleteAuth
- wolfTPM2_NVReadAuthPolicy

Return:

- TPM_RC_SUCCESS: successful
- TPM_RC_FAILURE: generic failure (check TPM IO and TPM return code)
- BAD_FUNC_ARG: check the provided arguments

Note:

- User data size should be less or equal to the NV Index maxSize specified using wolfTPM2_CreateAuth
- For an index with TPMA_NV_OWNERREAD or TPMA_NV_PPREAD set nv->handle.hndl to TPM_RH_OWNER or TPM_RH_PLATFORM and nv->handle.auth to that hierarchy auth (after wolfTPM2_NVOpen)

29.9.6.147 function wolfTPM2_NVReadAuthPolicy

```
WOLFTPM_API int wolfTPM2_NVReadAuthPolicy(
    WOLFTPM2_DEV * dev,
    WOLFTPM2_SESSION * tpmSession,
    TPM_ALG_ID pcrAlg,
    byte * pcrArray,
    word32 pcrArraySz,
    WOLFTPM2_NV * nv,
    word32 nvIndex,
    byte * dataBuf,
    word32 * pDataSz,
    word32 offset
)
```

Reads user data from a NV Index, starting at the given offset. Allows using a policy session and PCR's for authentication.

Parameters:

- **dev** pointer to a TPM2_DEV struct
- **tpmSession** pointer to a WOLFTPM2_SESSION struct used with wolfTPM2_StartSession and wolfTPM2_SetAuthSession
- **pcrAlg** the hash algorithm to use with PCR policy
- **pcrArray** array of PCR Indexes to use when creating the policy
- **pcrArraySz** the number of PCR Indexes in the pcrArray
- **nv** pointer to a populated structure of WOLFTPM2_NV type
- **nvIndex** integer value, holding an existing NV Index Handle value
- **dataBuf** pointer to an empty byte buffer, used to store the read data from the TPM's NVRAM
- **pDataSz** pointer to an integer variable, used to store the size of the data read from NVRAM, in bytes
- **offset** integer value of word32 type, specifying the offset from the NV Index memory start, can be zero

See:

- wolfTPM2_NVWriteAuth
- wolfTPM2_NVCreateAuth
- wolfTPM2_NVDeleteAuth
- wolfTPM2_NVReadAuth

Return:

- TPM_RC_SUCCESS: successful
- TPM_RC_FAILURE: generic failure (check TPM IO and TPM return code)
- BAD_FUNC_ARG: check the provided arguments
- BUFFER_E: data too large for a policy without PCR's

Note:

- User data size should be less or equal to the NV Index maxSize specified using wolfTPM2_CreateAuth
- For an index with TPMA_NV_OWNERREAD or TPMA_NV_PPREAD set nv->handle.hndl to TPM_RH_OWNER or TPM_RH_PLATFORM and nv->handle.auth to that hierarchy auth (after wolfTPM2_NVOpen)
- pcrArray may be NULL with pcrArraySz 0 when the policy session was already satisfied by the caller (e.g. PolicySecret or PolicyCommandCode). The data must then fit in one MAX_NV_BUFFER_SIZE chunk, otherwise BUFFER_E is returned

29.9.6.148 function wolfTPM2_NVReadCert

```
WOLFTPM_API int wolfTPM2_NVReadCert(
    WOLFTPM2_DEV * dev,
    TPM_HANDLE handle,
    uint8_t * buffer,
    uint32_t * len
)
```

Helper to get size of NV and read buffer without authentication. Typically used for reading a certificate from an NV.

Parameters:

- **dev** pointer to a TPM2_DEV struct
- **handle** integer value, holding an existing NV Index Handle value
- **buffer** pointer to an empty byte buffer, used to store the read data from the TPM's NVRAM

- **len** pointer to an integer variable, used to store the size of the data read from NVRAM, in bytes

See:

- wolfTPM2_NVWriteAuth
- wolfTPM2_NVCreateAuth
- wolfTPM2_NVDeleteAuth

Return:

- TPM_RC_SUCCESS: successful
- TPM_RC_FAILURE: generic failure (check TPM IO and TPM return code)
- BAD_FUNC_ARG: check the provided arguments

29.9.6.149 function wolfTPM2_NVIncrement

```
WOLFTPM_API int wolfTPM2_NVIncrement(
    WOLFTPM2_DEV * dev,
    WOLFTPM2_NV * nv
)
```

Increments an NV one-way counter.

Parameters:

- **dev** pointer to a TPM2_DEV struct
- **nv** pointer to a populated structure of WOLFTPM2_NV type

See:

- wolfTPM2_NVOpen
- wolfTPM2_NVCreateAuth

Return:

- TPM_RC_SUCCESS: successful
- TPM_RC_FAILURE: generic failure (check TPM IO and TPM return code)
- BAD_FUNC_ARG: check the provided arguments

29.9.6.150 function wolfTPM2_NVOpen

```
WOLFTPM_API int wolfTPM2_NVOpen(
    WOLFTPM2_DEV * dev,
    WOLFTPM2_NV * nv,
    word32 nvIndex,
    const byte * auth,
    word32 authSz
)
```

Open an NV and populate the required authentication and name hash.

Parameters:

- **dev** pointer to a TPM2_DEV struct
- **nv** pointer to an empty structure of WOLFTPM2_NV type, to hold the new NV Index
- **nvIndex** integer value, holding the NV Index Handle given by the TPM upon success
- **auth** pointer to a string constant, specifying the password authorization for this NV Index
- **authSz** integer value, specifying the size of the password authorization, in bytes

See:

- wolfTPM2_NVCreateAuth

- wolfTPM2_UnloadHandle

Return:

- TPM_RC_SUCCESS: successful
- TPM_RC_FAILURE: generic failure (check TPM IO and TPM return code)
- BAD_FUNC_ARG: check the provided arguments

29.9.6.151 function wolfTPM2_NVWriteLock

```
WOLFTPM_API int wolfTPM2_NVWriteLock(
    WOLFTPM2_DEV * dev,
    WOLFTPM2_NV * nv
)
```

Lock writes on the specified NV Index.

Parameters:

- **dev** pointer to a TPM2_DEV struct
- **nv** pointer to an structure of WOLFTPM2_NV type loaded using wolfTPM2_NVOpen

See:

- wolfTPM2_NVOpen
- wolfTPM2_NVCreateAuth
- wolfTPM2_NVWriteAuth
- wolfTPM2_NVReadAuth

Return:

- TPM_RC_SUCCESS: successful
- TPM_RC_FAILURE: generic failure (check TPM IO and TPM return code)
- BAD_FUNC_ARG: check the provided arguments

29.9.6.152 function wolfTPM2_NVDeleteAuth

```
WOLFTPM_API int wolfTPM2_NVDeleteAuth(
    WOLFTPM2_DEV * dev,
    WOLFTPM2_HANDLE * parent,
    word32 nvIndex
)
```

Destroys an existing NV Index.

Parameters:

- **dev** pointer to a TPM2_DEV struct
- **parent** pointer to a WOLFTPM2_HANDLE, specifying the TPM hierarchy for the new NV Index
- **nvIndex** integer value, holding the NV Index Handle given by the TPM upon success

See:

- wolfTPM2_NVCreateAuth
- wolfTPM2_NVWriteAuth
- wolfTPM2_NVReadAuth

Return:

- TPM_RC_SUCCESS: successful
- TPM_RC_FAILURE: generic failure (check TPM IO and TPM return code)
- BAD_FUNC_ARG: check the provided arguments

29.9.6.153 function wolfTPM2_NVCreate

```
WOLFTPM_API int wolfTPM2_NVCreate(
    WOLFTPM2_DEV * dev,
    TPM_HANDLE authHandle,
    word32 nvIndex,
    word32 nvAttributes,
    word32 maxSize,
    const byte * auth,
    int authSz
)
```

Deprecated, use newer API.

See: wolfTPM2_NVCreateAuth

29.9.6.154 function wolfTPM2_NVWrite

```
WOLFTPM_API int wolfTPM2_NVWrite(
    WOLFTPM2_DEV * dev,
    TPM_HANDLE authHandle,
    word32 nvIndex,
    byte * dataBuf,
    word32 dataSz,
    word32 offset
)
```

Deprecated, use newer API.

See: wolfTPM2_NVWriteAuth

29.9.6.155 function wolfTPM2_NVRead

```
WOLFTPM_API int wolfTPM2_NVRead(
    WOLFTPM2_DEV * dev,
    TPM_HANDLE authHandle,
    word32 nvIndex,
    byte * dataBuf,
    word32 * dataSz,
    word32 offset
)
```

Deprecated, use newer API.

See: wolfTPM2_NVReadAuth

29.9.6.156 function wolfTPM2_NVDelete

```
WOLFTPM_API int wolfTPM2_NVDelete(
    WOLFTPM2_DEV * dev,
    TPM_HANDLE authHandle,
    word32 nvIndex
)
```

Deprecated, use newer API.

See: wolfTPM2_NVDeleteAuth

29.9.6.157 function wolfTPM2_NVReadPublic

```
WOLFTPM_API int wolfTPM2_NVReadPublic(
    WOLFTPM2_DEV * dev,
    word32 nvIndex,
    TPMS_NV_PUBLIC * nvPublic
)
```

Extracts the public information about an nvIndex, such as maximum size.

Parameters:

- **dev** pointer to a TPM2_DEV struct
- **nvIndex** integer value, holding the NV Index Handle given by the TPM upon success
- **nvPublic** pointer to a TPMS_NV_PUBLIC, used to store the extracted nvIndex public information

See:

- wolfTPM2_NVCreateAuth
- wolfTPM2_NVDeleteAuth
- wolfTPM2_NVWriteAuth
- wolfTPM2_NVReadAuth

Return:

- TPM_RC_SUCCESS: successful
- TPM_RC_FAILURE: generic failure (check TPM IO and TPM return code)
- BAD_FUNC_ARG: check the provided arguments

29.9.6.158 function wolfTPM2_NVStoreKey

```
WOLFTPM_API int wolfTPM2_NVStoreKey(
    WOLFTPM2_DEV * dev,
    TPM_HANDLE primaryHandle,
    WOLFTPM2_KEY * key,
    TPM_HANDLE persistentHandle
)
```

Helper function to store a TPM 2.0 Key into the TPM's NVRAM.

Parameters:

- **dev** pointer to a TPM2_DEV struct
- **primaryHandle** integer value, specifying a TPM 2.0 Hierarchy. typically TPM_RH_OWNER
- **key** pointer to a structure of WOLFTPM2_KEY type, containing the TPM 2.0 key for storing
- **persistentHandle** integer value, specifying an existing nvIndex

See:

- wolfTPM2_NVDeleteKey
- wolfTPM2_NVCreateAuth
- wolfTPM2_NVDeleteAuth

Return:

- TPM_RC_SUCCESS: successful
- TPM_RC_FAILURE: generic failure (check TPM IO and TPM return code)
- BAD_FUNC_ARG: check the provided arguments

29.9.6.159 function wolfTPM2_NVDeleteKey

```
WOLFTPM_API int wolfTPM2_NVDeleteKey(
    WOLFTPM2_DEV * dev,
    TPM_HANDLE primaryHandle,
    WOLFTPM2_KEY * key
)
```

Helper function to delete a TPM 2.0 Key from the TPM's NVRAM.

Parameters:

- **dev** pointer to a TPM2_DEV struct
- **primaryHandle** integer value, specifying a TPM 2.0 Hierarchy. typically TPM_RH_OWNER
- **key** pointer to a structure of WOLFTPM2_KEY type, containing the nvIndex handle value

See:

- wolfTPM2_NVDeleteKey
- wolfTPM2_NVCreateAuth
- wolfTPM2_NVDeleteAuth

Return:

- TPM_RC_SUCCESS: successful
- TPM_RC_FAILURE: generic failure (check TPM IO and TPM return code)
- BAD_FUNC_ARG: check the provided arguments

29.9.6.160 function wolfTPM2_GetRng

```
WOLFTPM_API struct WC_RNG * wolfTPM2_GetRng(
    WOLFTPM2_DEV * dev
)
```

Get the wolfcrypt RNG instance used for wolfTPM.

Parameters:

- **dev** pointer to a TPM2_DEV struct

See: wolfTPM2_GetRandom

Return:

- TPM_RC_SUCCESS: successful
- TPM_RC_FAILURE: generic failure (check TPM IO and TPM return code)
- BAD_FUNC_ARG: check the provided arguments

Note: Only if wolfcrypt is enabled and configured for use instead of the TPM RNG

29.9.6.161 function wolfTPM2_GetRandom

```
WOLFTPM_API int wolfTPM2_GetRandom(
    WOLFTPM2_DEV * dev,
    byte * buf,
    word32 len
)
```

Get a set of random number, generated with the TPM RNG or wolfcrypt RNG.

Parameters:

- **dev** pointer to a TPM2_DEV struct

- **buf** pointer to a byte buffer, used to store the generated random numbers
- **len** integer value of word32 type, used to store the size of the buffer, in bytes

See: wolfTPM2_GetRandom

Return:

- TPM_RC_SUCCESS: successful
- TPM_RC_FAILURE: generic failure (check TPM IO and TPM return code)
- BAD_FUNC_ARG: check the provided arguments

Note: Define WOLFTPM2_USE_HW_RNG to use the TPM RNG source.

Warning: When using WOLFTPM2_USE_HW_RNG, random data travels over the SPI/I2C bus unprotected. If this data is used for session nonces, an attacker with physical bus access can force known values. Use default local RNG unless the bus is physically secure.

29.9.6.162 function wolfTPM2_UnloadHandle

```
WOLFTPM_API int wolfTPM2_UnloadHandle(
    WOLFTPM2_DEV * dev,
    WOLFTPM2_HANDLE * handle
)
```

Use to discard any TPM loaded object.

Parameters:

- **dev** pointer to a TPM2_DEV struct
- **handle** pointer to a structure of WOLFTPM2_HANDLE type, with a valid TPM 2.0 handle value

See: wolfTPM2_UnloadHandles

Return:

- TPM_RC_SUCCESS: successful
- TPM_RC_FAILURE: generic failure (check TPM IO and TPM return code)
- BAD_FUNC_ARG: check the provided arguments

29.9.6.163 function wolfTPM2_Clear

```
WOLFTPM_API int wolfTPM2_Clear(
    WOLFTPM2_DEV * dev
)
```

Clears the TPM with TPM2_Clear, authorized by the lockout hierarchy. This removes the owner and endorsement hierarchy objects and the NV indices created by the owner, and resets their authorizations. NV indices created by the platform (TPMA_NV_PLATFORMCREATE) are preserved. It does not deinitialize wolfTPM; use wolfTPM2_Cleanup for that.

Parameters:

- **dev** pointer to a TPM2_DEV struct

See: wolfTPM2_Cleanup

Return:

- TPM_RC_SUCCESS: successful
- TPM_RC_FAILURE: generic failure (check TPM IO and TPM return code)
- BAD_FUNC_ARG: check the provided arguments

29.9.6.164 function wolfTPM2_DictionaryAttackLockReset

```
WOLFTPM_API int wolfTPM2_DictionaryAttackLockReset(
    WOLFTPM2_DEV * dev
)
```

Reset the dictionary attack lockout failed-tries counter.

Parameters:

- **dev** pointer to a TPM2_DEV struct

See: wolfTPM2_DictionaryAttackParameters

Return:

- TPM_RC_SUCCESS: successful
- TPM_RC_LOCKOUT: lockoutAuth is in its recovery interval
- BAD_FUNC_ARG: check the provided arguments

29.9.6.165 function wolfTPM2_DictionaryAttackParameters

```
WOLFTPM_API int wolfTPM2_DictionaryAttackParameters(
    WOLFTPM2_DEV * dev,
    word32 newMaxTries,
    word32 newRecoveryTime,
    word32 lockoutRecovery
)
```

Set the dictionary attack lockout parameters.

Parameters:

- **dev** pointer to a TPM2_DEV struct
- **newMaxTries** failed auths before lockout (must be non-zero)
- **newRecoveryTime** seconds to forgive one failed try (0 = reboot only)
- **lockoutRecovery** seconds before a failed lockoutAuth recovers

See: wolfTPM2_DictionaryAttackLockReset

Return:

- TPM_RC_SUCCESS: successful
- TPM_RC_LOCKOUT: lockoutAuth is in its recovery interval
- BAD_FUNC_ARG: check the provided arguments

29.9.6.166 function wolfTPM2_HashStart

```
WOLFTPM_API int wolfTPM2_HashStart(
    WOLFTPM2_DEV * dev,
    WOLFTPM2_HASH * hash,
    TPMI_ALG_HASH hashAlg,
    const byte * usageAuth,
    word32 usageAuthSz
)
```

Helper function to start a TPM generated hash.

Parameters:

- **dev** pointer to a TPM2_DEV struct
- **hash** pointer to a WOLFTPM2_HASH structure

- **hashAlg** integer value, specifying a valid TPM 2.0 hash algorithm
- **usageAuth** pointer to a string constant, specifying the authorization for subsequent use of the hash
- **usageAuthSz** integer value, specifying the size of the authorization, in bytes

See:

- wolfTPM2_HashUpdate
- wolfTPM2_HashFinish

Return:

- TPM_RC_SUCCESS: successful
- TPM_RC_FAILURE: generic failure (check TPM IO and TPM return code)
- BAD_FUNC_ARG: check the provided arguments

29.9.6.167 function wolfTPM2_HashUpdate

```
WOLFTPM_API int wolfTPM2_HashUpdate(
    WOLFTPM2_DEV * dev,
    WOLFTPM2_HASH * hash,
    const byte * data,
    word32 dataSz
)
```

Update a TPM generated hash with new user data.

Parameters:

- **dev** pointer to a TPM2_DEV struct
- **hash** pointer to a WOLFTPM2_HASH structure
- **data** pointer to a byte buffer, containing the user data to be added to the hash
- **dataSz** integer value of word32 type, specifying the size of the user data, in bytes

See:

- wolfTPM2_HashStart
- wolfTPM2_HashFinish

Return:

- TPM_RC_SUCCESS: successful
- TPM_RC_FAILURE: generic failure (check TPM IO and TPM return code)
- BAD_FUNC_ARG: check the provided arguments

Note: Make sure the auth is correctly set

29.9.6.168 function wolfTPM2_HashFinish

```
WOLFTPM_API int wolfTPM2_HashFinish(
    WOLFTPM2_DEV * dev,
    WOLFTPM2_HASH * hash,
    byte * digest,
    word32 * digestSz
)
```

Finalize a TPM generated hash and get the digest output in a user buffer.

Parameters:

- **dev** pointer to a TPM2_DEV struct
- **hash** pointer to a WOLFTPM2_HASH structure

- **digest** pointer to a byte buffer, used to store the resulting digest
- **digestSz** pointer to size of digest buffer, on return set to bytes stored in digest buffer

See:

- wolfTPM2_HashStart
- wolfTPM2_HashUpdate

Return:

- TPM_RC_SUCCESS: successful
- TPM_RC_FAILURE: generic failure (check TPM IO and TPM return code)
- BAD_FUNC_ARG: check the provided arguments

Note: Make sure the auth is correctly set

29.9.6.169 function wolfTPM2_LoadKeyedHashKey

```
WOLFTPM_API int wolfTPM2_LoadKeyedHashKey(
    WOLFTPM2_DEV * dev,
    WOLFTPM2_KEY * key,
    WOLFTPM2_HANDLE * parent,
    int hashAlg,
    const byte * keyBuf,
    word32 keySz,
    const byte * usageAuth,
    word32 usageAuthSz
)
```

Creates and loads a new TPM key of KeyedHash type, typically used for HMAC operations.

Parameters:

- **dev** pointer to a TPM2_DEV struct
- **key** pointer to an empty structure of WOLFTPM2_KEY type, to store the generated key
- **parent** pointer to a structure of WOLFTPM2_KEY type, containing a valid TPM handle of a primary key
- **hashAlg** integer value, specifying a valid TPM 2.0 hash algorithm
- **keyBuf** pointer to a byte array, containing derivation values for the new KeyedHash key
- **keySz** integer value, specifying the size of the derivation values stored in keyBuf, in bytes
- **usageAuth** pointer to a string constant, specifying the authorization of the new key
- **usageAuthSz** integer value, specifying the size of the authorization, in bytes

See:

- wolfTPM2_HmacStart
- wolfTPM2_HmacUpdate
- wolfTPM2_HmacFinish

Return:

- TPM_RC_SUCCESS: successful
- TPM_RC_FAILURE: generic failure (check TPM IO and TPM return code)
- BAD_FUNC_ARG: check the provided arguments

Note: To generate HMAC using the TPM it is recommended to use the wolfTPM2_Hmac wrappers

29.9.6.170 function wolfTPM2_HmacStart

```
WOLFTPM_API int wolfTPM2_HmacStart(
    WOLFTPM2_DEV * dev,
```

```
    WOLFTPM2_HMAC * hmac,
    WOLFTPM2_HANDLE * parent,
    TPMI_ALG_HASH hashAlg,
    const byte * keyBuf,
    word32 keySz,
    const byte * usageAuth,
    word32 usageAuthSz
)
```

Helper function to start a TPM generated hmac.

Parameters:

- **dev** pointer to a TPM2_DEV struct
- **hmac** pointer to a WOLFTPM2_HMAC structure
- **parent** pointer to a structure of WOLFTPM2_KEY type, containing a valid TPM handle of a primary key
- **hashAlg** integer value, specifying a valid TPM 2.0 hash algorithm
- **keyBuf** pointer to a byte array, containing derivation values for the new KeyedHash key
- **keySz** integer value, specifying the size of the derivation values stored in keyBuf, in bytes
- **usageAuth** pointer to a string constant, specifying the authorization for subsequent use of the hmac
- **usageAuthSz** integer value, specifying the size of the authorization, in bytes

See:

- wolfTPM2_HmacUpdate
- wolfTPM2_HmacFinish
- wolfTPM2_LoadKeyedHashKey

Return:

- TPM_RC_SUCCESS: successful
- TPM_RC_FAILURE: generic failure (check TPM IO and TPM return code)
- BAD_FUNC_ARG: check the provided arguments

29.9.6.171 function wolfTPM2_HmacUpdate

```
WOLFTPM_API int wolfTPM2_HmacUpdate(
    WOLFTPM2_DEV * dev,
    WOLFTPM2_HMAC * hmac,
    const byte * data,
    word32 dataSz
)
```

Update a TPM generated hmac with new user data.

Parameters:

- **dev** pointer to a TPM2_DEV struct
- **hmac** pointer to a WOLFTPM2_HMAC structure
- **data** pointer to a byte buffer, containing the user data to be added to the hmac
- **dataSz** integer value of word32 type, specifying the size of the user data, in bytes
- **dev** pointer to a TPM2_DEV struct
- **hmac** pointer to an active WOLFTPM2_HMAC structure
- **data** pointer to data to add to HMAC
- **dataSz** size of data in bytes

See:

- wolfTPM2_HmacStart
- wolfTPM2_HMACFinish
- wolfTPM2_HmacStart
- wolfTPM2_HmacFinish

Return:

- TPM_RC_SUCCESS: successful
- TPM_RC_FAILURE: generic failure (check TPM IO and TPM return code)
- BAD_FUNC_ARG: check the provided arguments
- TPM_RC_SUCCESS: successful
- TPM_RC_FAILURE: generic failure (check TPM IO and TPM return code)
- BAD_FUNC_ARG: check the provided arguments

Note:

- Make sure the TPM authorization is correctly set
- Adds data to an active HMAC sequence

Update an HMAC operation with data

29.9.6.172 function wolfTPM2_HmacFinish

```
WOLFTPM_API int wolfTPM2_HmacFinish(
    WOLFTPM2_DEV * dev,
    WOLFTPM2_HMAC * hmac,
    byte * digest,
    word32 * digestSz
)
```

Finalize a TPM generated hmac and get the digest output in a user buffer.

Parameters:

- **dev** pointer to a TPM2_DEV struct
- **hmac** pointer to a WOLFTPM2_HMAC structure
- **digest** pointer to a byte buffer, used to store the resulting hmac digest
- **digestSz** integer value of word32 type, specifying the size of the digest, in bytes

See:

- wolfTPM2_HmacStart
- wolfTPM2_HmacUpdate

Return:

- TPM_RC_SUCCESS: successful
- TPM_RC_FAILURE: generic failure (check TPM IO and TPM return code)
- BAD_FUNC_ARG: check the provided arguments

Note: Make sure the TPM authorization is correctly set

29.9.6.173 function wolfTPM2_LoadSymmetricKey

```
WOLFTPM_API int wolfTPM2_LoadSymmetricKey(
    WOLFTPM2_DEV * dev,
    WOLFTPM2_KEY * key,
    int alg,
    const byte * keyBuf,
    word32 keySz
)
```

Loads an external symmetric key into the TPM.

Parameters:

- **dev** pointer to a TPM2_DEV struct
- **key** pointer to an empty structure of WOLFTPM2_KEY type, to store the TPM handle and key information
- **alg** integer value, specifying a valid TPM 2.0 symmetric key algorithm, e.g. TPM_ALG_CFB for AES CFB
- **keyBuf** pointer to a byte array, containing private material of the symmetric key
- **keySz** integer value, specifying the size of the key material stored in keyBuf, in bytes
- **dev** pointer to a TPM2_DEV struct
- **key** pointer to an empty WOLFTPM2_KEY structure to store loaded key
- **alg** algorithm type (TPM_ALG_AES, etc)
- **keyBuf** pointer to key material
- **keySz** size of key material in bytes

See:

- wolfTPM2_EncryptDecryptBlock
- wolfTPM2_EncryptDecrypt
- TPM2_EncryptDecrypt2
- wolfTPM2_EncryptDecryptBlock
- wolfTPM2_EncryptDecrypt
- wolfTPM2_SetKeyAuthPassword

Return:

- TPM_RC_SUCCESS: successful
- TPM_RC_FAILURE: generic failure (check TPM IO and TPM return code)
- BAD_FUNC_ARG: check the provided arguments
- TPM_RC_SUCCESS: successful
- TPM_RC_FAILURE: generic failure (check TPM IO and TPM return code)
- BAD_FUNC_ARG: check the provided arguments

Note: Creates and loads a symmetric key for encryption/decryption operations

Warning: If key->handle.auth is empty the key is loaded with empty authorization and any caller holding the handle can use it. Set key->handle.auth (e.g. via wolfTPM2_SetKeyAuthPassword) before loading to require a password.

Load a symmetric key into the TPM

29.9.6.174 function wolfTPM2_EncryptDecryptBlock

```
WOLFTPM_API int wolfTPM2_EncryptDecryptBlock(
    WOLFTPM2_DEV * dev,
    WOLFTPM2_KEY * key,
    const byte * in,
    byte * out,
    word32 inOutSz,
    byte * iv,
    word32 ivSz,
    int isDecrypt
)
```

Encrypt or decrypt a single block of data with the symmetric key.

Parameters:

- **dev** pointer to a TPM2_DEV (must not be NULL)
- **key** handle/template of the loaded symmetric key
- **in** input buffer
- **out** output buffer (in-place is allowed)
- **inOutSz** length of `in` / `out` in bytes
- **iv** input/output IV buffer
- **ivSz** length of `iv` in bytes
- **isDecrypt** WOLFTPM2_ENCRYPT (0) for encrypt, WOLFTPM2_DECRYPT (1) for decrypt

Return:

- TPM_RC_SUCCESS on success.
- BAD_FUNC_ARG if dev/key/in/out is NULL, inOutSz is zero, the mode needs an IV but none was supplied, or a CBC/ECB length is not block-aligned.
- BUFFER_E if `inOutSz` exceeds the TPM input buffer size or `ivSz` exceeds the TPM IV buffer size.
- other TPM_RC_* values surfaced from TPM2_EncryptDecrypt2.

Wraps TPM2_EncryptDecrypt2 (TPM 2.0 Part 3 Section 15). The TPM symmetric mode is taken from `key->pub.publicArea.parameters.symDetail.sym.mode.aes`.

Length contract:

- TPM_ALG_CBC and TPM_ALG_ECB require `inOutSz` to be a whole multiple of MAX_AES_BLOCK_SIZE_BYTES; otherwise BAD_FUNC_ARG is returned.
- TPM_ALG_CFB, TPM_ALG_OFB, TPM_ALG_CTR accept any non-zero length and the TPM-returned IV reflects exactly the bytes consumed.
- TPM_ALG_NULL is forwarded to the TPM, which rejects it.

IV requirements: modes other than ECB and NULL require a non-NULL IV. On success the IV in `iv` is updated with the TPM's running IV so the next call can chain.

29.9.6.175 function wolfTPM2_EncryptDecrypt

```
WOLFTPM_API int wolfTPM2_EncryptDecrypt(
    WOLFTPM2_DEV * dev,
    WOLFTPM2_KEY * key,
    const byte * in,
    byte * out,
    word32 inOutSz,
    byte * iv,
    word32 ivSz,
    int isDecrypt
)
```

29.9.6.176 function wolfTPM2_SetCommand

```
WOLFTPM_API int wolfTPM2_SetCommand(
    WOLFTPM2_DEV * dev,
    TPM_CC commandCode,
    int enableFlag
)
```

Vendor specific TPM command, used to enable other restricted TPM commands.

Parameters:

- **dev** pointer to a TPM2_DEV struct
- **commandCode** integer value, representing a valid vendor command
- **enableFlag** integer value, non-zero values represent "to enable"

See: TPM2_GPIO_Config

Return:

- TPM_RC_SUCCESS: successful
- TPM_RC_FAILURE: generic failure (check TPM IO and TPM return code)
- BAD_FUNC_ARG: check the provided arguments

29.9.6.177 function wolfTPM2_Reset

```
WOLFTPM_API int wolfTPM2_Reset(
    WOLFTPM2_DEV * dev,
    int doShutdown,
    int doStartup
)
```

Helper function to shutdown, startup or reset the TPM.

Parameters:

- **dev** pointer to a TPM2_DEV struct
- **doStartup** integer value, non-zero values represent "perform Startup after Shutdown"
- **doShutdown** integer value, non-zero values represent "perform Shutdown"

See:

- wolfTPM2_Init
- wolfTPM2_Reset

Return:

- TPM_RC_SUCCESS: successful
- TPM_RC_FAILURE: generic failure (check TPM IO and TPM return code)
- NOT_COMPILED_IN: transport where the OS owns TPM startup state
- BAD_FUNC_ARG: check the provided arguments

Note:

- The behavior depends on the doStartup and doShutdown flags:
- – Both flags set to 1: Performs a full TPM restart (shutdown then startup)
- – Only doStartup=1: Just starts up the TPM
- – Only doShutdown=1: Just shuts down the TPM
- On the Linux kernel driver (/dev/tpmX) and Windows TBS the OS owns TPM startup state, so no TPM command is sent and NOT_COMPILED_IN is returned. A call requesting neither a shutdown nor a startup still returns TPM_RC_SUCCESS, since nothing was declined. See docs/DEVTPM.md.

29.9.6.178 function wolfTPM2_Shutdown

```
WOLFTPM_API int wolfTPM2_Shutdown(
    WOLFTPM2_DEV * dev,
    int doStartup
)
```

Helper function to shutdown or reset the TPM.

Parameters:

- **dev** pointer to a TPM2_DEV struct
- **doStartup** integer value, non-zero values represent "perform Startup after Shutdown"

See:

- wolfTPM2_Init
- wolfTPM2_Shutdown

Return:

- TPM_RC_SUCCESS: successful
- TPM_RC_FAILURE: generic failure (check TPM IO and TPM return code)
- NOT_COMPILED_IN: transport where the OS owns TPM startup state
- BAD_FUNC_ARG: check the provided arguments

Note:

- If doStartup is set, then TPM2_Startup is performed right after TPM2_Shutdown
- On the Linux kernel driver (/dev/tpmX) and Windows TBS the OS owns TPM startup state, so no TPM command is sent and NOT_COMPILED_IN is returned. A call requesting neither a shutdown nor a startup still returns TPM_RC_SUCCESS, since nothing was declined. See docs/DEVTPM.md.

29.9.6.179 function wolfTPM2_UnloadHandles

```
WOLFTPM_API int wolfTPM2_UnloadHandles(
    WOLFTPM2_DEV * dev,
    word32 handleStart,
    word32 handleCount
)
```

One-shot API to unload subsequent TPM handles.

Parameters:

- **dev** pointer to a TPM2_DEV struct
- **handleStart** integer value of word32 type, specifying the value of the first TPM handle
- **handleCount** integer value of word32 type, specifying the number of handles

See: wolfTPM2_Init

Return:

- TPM_RC_SUCCESS: successful
- TPM_RC_FAILURE: generic failure (check TPM IO and TPM return code)
- BAD_FUNC_ARG: check the provided arguments

29.9.6.180 function wolfTPM2_UnloadHandles_AllTransient

```
WOLFTPM_API int wolfTPM2_UnloadHandles_AllTransient(
    WOLFTPM2_DEV * dev
)
```

One-shot API to unload all transient TPM handles.

Parameters:

- **dev** pointer to a TPM2_DEV struct

See:

- wolfTPM2_UnloadHandles
- wolfTPM2_CreatePrimary

Return:

- TPM_RC_SUCCESS: successful

- TPM_RC_FAILURE: generic failure (check TPM IO and TPM return code)
- BAD_FUNC_ARG: check the provided arguments

Note:

- Flushes the transient handles the TPM reports as loaded, following TPM_CAP_HANDLES continuation pages, or the first MAX_HANDLE_NUM if the first TPM_CAP_HANDLES query fails. A failed query or flush is returned; a failed flush does not stop the remaining handles from being flushed
- If there are Primary Keys as transient objects, they need to be recreated before TPM keys can be used

29.9.6.181 function wolfTPM2_GetKeyTemplate_RSA

```
WOLFTPM_API int wolfTPM2_GetKeyTemplate_RSA(
    TPMT_PUBLIC * publicTemplate,
    TPMA_OBJECT objectAttributes
)
```

Prepares a TPM public template for new RSA key based on user selected object attributes.

Parameters:

- **publicTemplate** pointer to an empty structure of TPMT_PUBLIC type, to store the new RSA template
- **objectAttributes** integer value of TPMA_OBJECT type, can contain one or more attributes, e.g. TPMA_OBJECT_fixedTPM

See:

- wolfTPM2_GetKeyTemplate_RSA_ex
- wolfTPM2_GetKeyTemplate_ECC
- wolfTPM2_GetKeyTemplate_Symmetric
- wolfTPM2_GetKeyTemplate_KeyedHash
- wolfTPM2_GetKeyTemplate_KeySeal

Return:

- TPM_RC_SUCCESS: successful
- BAD_FUNC_ARG: check the provided arguments

29.9.6.182 function wolfTPM2_GetKeyTemplate_RSA_ex

```
WOLFTPM_API int wolfTPM2_GetKeyTemplate_RSA_ex(
    TPMT_PUBLIC * publicTemplate,
    TPM_ALG_ID nameAlg,
    TPMA_OBJECT objectAttributes,
    int keyBits,
    long exponent,
    TPM_ALG_ID sigScheme,
    TPM_ALG_ID sigHash
)
```

Prepares a TPM public template for new RSA key based on user selected object attributes.

Parameters:

- **publicTemplate** pointer to an empty structure of TPMT_PUBLIC type, to store the new RSA template
- **nameAlg** integer value of TPM_ALG_ID type, specifying a TPM supported hashing algorithm, typically TPM_ALG_SHA256 for SHA 256

- **objectAttributes** integer value of TPMA_OBJECT type, can contain one or more attributes, e.g. TPMA_OBJECT_fixedTPM
- **keyBits** integer value, specifying the size of the symmetric key, typically 128 or 256 bits
- **exponent** integer value of word32 type, specifying the RSA exponent
- **sigScheme** integer value of TPM_ALG_ID type, specifying a TPM supported signature scheme
- **sigHash** integer value of TPM_ALG_ID type, specifying a TPM supported signature hash scheme

See:

- wolfTPM2_GetKeyTemplate_RSA
- wolfTPM2_GetKeyTemplate_ECC
- wolfTPM2_GetKeyTemplate_ECC_ex
- wolfTPM2_GetKeyTemplate_Symmetric
- wolfTPM2_GetKeyTemplate_KeyedHash
- wolfTPM2_GetKeyTemplate_KeySeal

Return:

- TPM_RC_SUCCESS: successful
- BAD_FUNC_ARG: check the provided arguments

29.9.6.183 function wolfTPM2_GetKeyTemplate_ECC

```
WOLFTPM_API int wolfTPM2_GetKeyTemplate_ECC(
    TPMT_PUBLIC * publicTemplate,
    TPMA_OBJECT objectAttributes,
    TPM_ECC_CURVE curve,
    TPM_ALG_ID sigScheme
)
```

Prepares a TPM public template for new ECC key based on user selected object attributes.

Parameters:

- **publicTemplate** pointer to an empty structure of TPMT_PUBLIC type, to store the new ECC key template
- **objectAttributes** integer value of TPMA_OBJECT type, can contain one or more attributes, e.g. TPMA_OBJECT_fixedTPM
- **curve** integer value of TPM_ECC_CURVE type, specifying a TPM supported ECC curve ID; the requested curve is used exactly as given (the configurable WOLFTPM2_ECC_DEFAULT_CURVE applies only to the no-curve SRK/AIK templates, not to this explicit-curve API)
- **sigScheme** integer value of TPM_ALG_ID type, specifying a TPM supported signature scheme

See:

- wolfTPM2_GetKeyTemplate_ECC_ex
- wolfTPM2_GetKeyTemplate_RSA
- wolfTPM2_GetKeyTemplate_Symmetric
- wolfTPM2_GetKeyTemplate_KeyedHash
- wolfTPM2_GetKeyTemplate_KeySeal

Return:

- TPM_RC_SUCCESS: successful
- BAD_FUNC_ARG: check the provided arguments

29.9.6.184 function wolfTPM2_GetKeyTemplate_ECC_ex

```
WOLFTPM_API int wolfTPM2_GetKeyTemplate_ECC_ex(
    TPMT_PUBLIC * publicTemplate,
```

```
    TPM_ALG_ID nameAlg,
    TPMA_OBJECT objectAttributes,
    TPM_ECC_CURVE curve,
    TPM_ALG_ID sigScheme,
    TPM_ALG_ID sigHash
)
```

Prepares a TPM public template for new ECC key based on user selected object attributes.

Parameters:

- **publicTemplate** pointer to an empty structure of TPMT_PUBLIC type, to store the new ECC key template
- **nameAlg** integer value of TPM_ALG_ID type, specifying a TPM supported hashing algorithm, typically TPM_ALG_SHA256 for SHA 256
- **objectAttributes** integer value of TPMA_OBJECT type, can contain one or more attributes, e.g. TPMA_OBJECT_fixedTPM
- **curve** integer value of TPM_ECC_CURVE type, specifying a TPM supported ECC curve ID; the requested curve is used exactly as given (the configurable WOLFTPM2_ECC_DEFAULT_CURVE applies only to the no-curve SRK/AIK templates, not to this explicit-curve API)
- **sigScheme** integer value of TPM_ALG_ID type, specifying a TPM supported signature scheme
- **sigHash** integer value of TPM_ALG_ID type, specifying a TPM supported signature hash scheme

See:

- wolfTPM2_GetKeyTemplate_ECC
- wolfTPM2_GetKeyTemplate_RSA
- wolfTPM2_GetKeyTemplate_Symmetric
- wolfTPM2_GetKeyTemplate_KeyedHash
- wolfTPM2_GetKeyTemplate_KeySeal

Return:

- TPM_RC_SUCCESS: successful
- BAD_FUNC_ARG: check the provided arguments

29.9.6.185 function wolfTPM2_GetKeyTemplate_Symmetric

```
WOLFTPM_API int wolfTPM2_GetKeyTemplate_Symmetric(
    TPMT_PUBLIC * publicTemplate,
    int keyBits,
    TPM_ALG_ID algMode,
    int isSign,
    int isDecrypt
)
```

Prepares a TPM public template for new Symmetric key.

Parameters:

- **publicTemplate** pointer to an empty structure of TPMT_PUBLIC type, to store the new Symmetric key template
- **keyBits** integer value, specifying the size of the symmetric key, typically 128 or 256 bits
- **algMode** integer value of TPM_ALG_ID type, specifying a TPM supported symmetric algorithm, e.g. TPM_ALG_CFB for AES CFB
- **isSign** integer value, non-zero values represent “a signing key”
- **isDecrypt** integer value, non-zero values represent “a decryption key”

See:

- wolfTPM2_GetKeyTemplate_RSA
- wolfTPM2_GetKeyTemplate_ECC
- wolfTPM2_GetKeyTemplate_KeyedHash
- wolfTPM2_GetKeyTemplate_KeySeal

Return:

- TPM_RC_SUCCESS: successful
- BAD_FUNC_ARG: check the provided arguments

29.9.6.186 function wolfTPM2_GetKeyTemplate_KeyedHash

```
WOLFTPM_API int wolfTPM2_GetKeyTemplate_KeyedHash(
    TPMT_PUBLIC * publicTemplate,
    TPM_ALG_ID hashAlg,
    int isSign,
    int isDecrypt
)
```

Prepares a TPM public template for new KeyedHash key.

Parameters:

- **publicTemplate** pointer to an empty structure of TPMT_PUBLIC type, to store the new template
- **hashAlg** integer value of TPM_ALG_ID type, specifying a TPM supported hashing algorithm, e.g. TPM_ALG_SHA256 for SHA 256
- **isSign** integer value, non-zero values represent "a signing key"
- **isDecrypt** integer value, non-zero values represent "a decryption key"

See:

- wolfTPM2_GetKeyTemplate_RSA
- wolfTPM2_GetKeyTemplate_ECC
- wolfTPM2_GetKeyTemplate_Symmetric
- wolfTPM2_GetKeyTemplate_KeySeal

Return:

- TPM_RC_SUCCESS: successful
- BAD_FUNC_ARG: check the provided arguments

29.9.6.187 function wolfTPM2_GetKeyTemplate_KeySeal

```
WOLFTPM_API int wolfTPM2_GetKeyTemplate_KeySeal(
    TPMT_PUBLIC * publicTemplate,
    TPM_ALG_ID nameAlg
)
```

Prepares a TPM public template for new key for sealing secrets.

Parameters:

- **publicTemplate** pointer to an empty structure of TPMT_PUBLIC type, to store the new template
- **nameAlg** integer value of TPM_ALG_ID type, specifying a TPM supported hashing algorithm, typically TPM_ALG_SHA256 for SHA 256

See:

- wolfTPM2_GetKeyTemplate_ECC
- wolfTPM2_GetKeyTemplate_Symmetric
- wolfTPM2_GetKeyTemplate_KeyedHash

- wolfTPM2_GetKeyTemplate_KeySeal

Return:

- TPM_RC_SUCCESS: successful
- BAD_FUNC_ARG: check the provided arguments

Note: There are strict requirements for a Key Seal, therefore most of the key parameters are predetermined by the wrapper

29.9.6.188 function wolfTPM2_GetKeyTemplate_EK

```
WOLFTPM_API int wolfTPM2_GetKeyTemplate_EK(
    TPMT_PUBLIC * publicTemplate,
    TPM_ALG_ID alg,
    int keyBits,
    TPM_ECC_CURVE curveID,
    TPM_ALG_ID nameAlg,
    int highRange
)
```

Prepares a TPM public template for generating the TPM Endorsement Key.

Parameters:

- **publicTemplate** pointer to an empty structure of TPMT_PUBLIC type, to store the new template
- **alg** can be only TPM_ALG_RSA or TPM_ALG_ECC, see Note above
- **keyBits** integer value, specifying bits for the key, typically 2048 (RSA) or 256 (ECC)
- **curveID** use one of the accepted TPM_ECC_CURVE values like TPM_ECC_NIST_P256 (only used when alg=TPM_ALG_ECC)
- **nameAlg** integer value of TPMI_ALG_HASH type, specifying a valid TPM2 hashing algorithm (typically TPM_ALG_SHA256)
- **highRange** integer value: 0=low range, 1=high range

See:

- wolfTPM2_GetKeyTemplate_ECC_EK
- wolfTPM2_GetKeyTemplate_RSA_SRK
- wolfTPM2_GetKeyTemplate_RSA_AIK
- wolfTPM2_GetKeyTemplate_EKIndex

Return:

- TPM_RC_SUCCESS: successful
- BAD_FUNC_ARG: check the provided arguments

29.9.6.189 function wolfTPM2_GetKeyTemplate_EKIndex

```
WOLFTPM_API int wolfTPM2_GetKeyTemplate_EKIndex(
    word32 nvIndex,
    TPMT_PUBLIC * publicTemplate
)
```

Helper to get the Endorsement public key template by NV index.

Parameters:

- **nvIndex** handle for NV index. Typically starting from TPM_20_TCG_NV_SPACE
- **publicTemplate** pointer to an empty structure of TPMT_PUBLIC type, to store the new template

See:

- wolfTPM2_GetKeyTemplate_EK
- wolfTPM2_GetKeyTemplate_ECC_EK
- wolfTPM2_GetKeyTemplate_RSA_SRK
- wolfTPM2_GetKeyTemplate_RSA_AIK

Return:

- TPM_RC_SUCCESS: successful
- BAD_FUNC_ARG: check the provided arguments

29.9.6.190 function wolfTPM2_GetKeyTemplate_RSA_EK

```
WOLFTPM_API int wolfTPM2_GetKeyTemplate_RSA_EK(
    TPMT_PUBLIC * publicTemplate
)
```

Prepares a TPM public template for generating the TPM Endorsement Key of RSA type.

Parameters:

- **publicTemplate** pointer to an empty structure of TPMT_PUBLIC type, to store the new template

See:

- wolfTPM2_GetKeyTemplate_EK
- wolfTPM2_GetKeyTemplate_ECC_EK
- wolfTPM2_GetKeyTemplate_RSA_SRK
- wolfTPM2_GetKeyTemplate_RSA_AIK

Return:

- TPM_RC_SUCCESS: successful
- BAD_FUNC_ARG: check the provided arguments

29.9.6.191 function wolfTPM2_GetKeyTemplate_ECC_EK

```
WOLFTPM_API int wolfTPM2_GetKeyTemplate_ECC_EK(
    TPMT_PUBLIC * publicTemplate
)
```

Prepares a TPM public template for generating the TPM Endorsement Key of ECC type.

Parameters:

- **publicTemplate** pointer to an empty structure of TPMT_PUBLIC type, to store the new template

See:

- wolfTPM2_GetKeyTemplate_EK
- wolfTPM2_GetKeyTemplate_RSA_EK
- wolfTPM2_GetKeyTemplate_ECC_SRK
- wolfTPM2_GetKeyTemplate_ECC_AIK

Return:

- TPM_RC_SUCCESS: successful
- BAD_FUNC_ARG: check the provided arguments

29.9.6.192 function wolfTPM2_GetKeyTemplate_RSA_SRK

```
WOLFTPM_API int wolfTPM2_GetKeyTemplate_RSA_SRK(
    TPMT_PUBLIC * publicTemplate
)
```

Prepares a TPM public template for generating a new TPM Storage Key of RSA type.

Parameters:

- **publicTemplate** pointer to an empty structure of TPMT_PUBLIC type, to store the new template

See:

- wolfTPM2_GetKeyTemplate_ECC_SRK
- wolfTPM2_GetKeyTemplate_RSA_AIK
- wolfTPM2_GetKeyTemplate_RSA_EK

Return:

- TPM_RC_SUCCESS: successful
- BAD_FUNC_ARG: check the provided arguments

29.9.6.193 function wolfTPM2_GetKeyTemplate_ECC_SRK

```
WOLFTPM_API int wolfTPM2_GetKeyTemplate_ECC_SRK(
    TPMT_PUBLIC * publicTemplate
)
```

Prepares a TPM public template for generating a new TPM Storage Key of ECC type.

Parameters:

- **publicTemplate** pointer to an empty structure of TPMT_PUBLIC type, to store the new template

See:

- wolfTPM2_GetKeyTemplate_RSA_SRK
- wolfTPM2_GetKeyTemplate_ECC_AIK
- wolfTPM2_GetKeyTemplate_ECC_EK

Return:

- TPM_RC_SUCCESS: successful
- BAD_FUNC_ARG: check the provided arguments

29.9.6.194 function wolfTPM2_GetKeyTemplate_RSA_AIK

```
WOLFTPM_API int wolfTPM2_GetKeyTemplate_RSA_AIK(
    TPMT_PUBLIC * publicTemplate
)
```

Prepares a TPM public template for generating a new TPM Attestation Key of RSA type.

Parameters:

- **publicTemplate** pointer to an empty structure of TPMT_PUBLIC type, to store the new template

See:

- wolfTPM2_GetKeyTemplate_ECC_AIK
- wolfTPM2_GetKeyTemplate_RSA_SRK

Return:

- TPM_RC_SUCCESS: successful
- BAD_FUNC_ARG: check the provided arguments

29.9.6.195 function wolfTPM2_GetKeyTemplate_ECC_AIK

```
WOLFTPM_API int wolfTPM2_GetKeyTemplate_ECC_AIK(
    TPMT_PUBLIC * publicTemplate
)
```

Prepares a TPM public template for generating a new TPM Attestation Key of ECC type.

Parameters:

- **publicTemplate** pointer to an empty structure of TPMT_PUBLIC type, to store the new template

See:

- wolfTPM2_GetKeyTemplate_RSA_AIK
- wolfTPM2_GetKeyTemplate_ECC_SRK

Return:

- TPM_RC_SUCCESS: successful
- BAD_FUNC_ARG: check the provided arguments

29.9.6.196 function wolfTPM2_GetKeyTemplate_RSA_IAK

```
WOLFTPM_API int wolfTPM2_GetKeyTemplate_RSA_IAK(
    TPMT_PUBLIC * publicTemplate,
    int keyBits,
    TPM_ALG_ID hashAlg
)
```

29.9.6.197 function wolfTPM2_GetKeyTemplate_ECC_IAK

```
WOLFTPM_API int wolfTPM2_GetKeyTemplate_ECC_IAK(
    TPMT_PUBLIC * publicTemplate,
    TPM_ECC_CURVE curveID,
    TPM_ALG_ID hashAlg
)
```

29.9.6.198 function wolfTPM2_GetKeyTemplate_ECC_IDevID

```
WOLFTPM_API int wolfTPM2_GetKeyTemplate_ECC_IDevID(
    TPMT_PUBLIC * publicTemplate,
    TPM_ECC_CURVE curveID,
    TPM_ALG_ID hashAlg
)
```

29.9.6.199 function wolfTPM2_GetKeyTemplate_RSA_IDevID

```
WOLFTPM_API int wolfTPM2_GetKeyTemplate_RSA_IDevID(
    TPMT_PUBLIC * publicTemplate,
    int keyBits,
    TPM_ALG_ID hashAlg
)
```

29.9.6.200 function wolfTPM2_SetKeyTemplate_Unique

```
WOLFTPM_API int wolfTPM2_SetKeyTemplate_Unique(
    TPMT_PUBLIC * publicTemplate,
    const byte * unique,
    int uniqueSz
)
```

Sets the unique area of a public template used by Create or CreatePrimary.

Parameters:

- **publicTemplate** pointer to an empty structure of TPMT_PUBLIC type, to store the new template
- **unique** optional pointer to buffer to populate unique area of public template. If NULL, the buffer will be zeroized.
- **uniqueSz** size to fill the unique field. If zero the key size is used.

See:

- wolfTPM2_CreateKey
- wolfTPM2_CreatePrimaryKey

Return:

- TPM_RC_SUCCESS: successful
- BAD_FUNC_ARG: check the provided arguments

29.9.6.201 function wolfTPM2_GetNvAttributesTemplate

```
WOLFTPM_API int wolfTPM2_GetNvAttributesTemplate(
    TPM_HANDLE auth,
    word32 * nvAttributes
)
```

Prepares a TPM NV Index template.

Parameters:

- **auth** integer value, representing the TPM Hierarchy under which the new TPM NV index will be created
- **nvAttributes** pointer to an empty integer variable, to store the NV Attributes

See:

- wolfTPM2_CreateAuth
- wolfTPM2_WriteAuth
- wolfTPM2_ReadAuth
- wolfTPM2_DeleteAuth

Return:

- TPM_RC_SUCCESS: successful
- BAD_FUNC_ARG: check the provided arguments

29.9.6.202 function wolfTPM2_CreateEK

```
WOLFTPM_API int wolfTPM2_CreateEK(
    WOLFTPM2_DEV * dev,
    WOLFTPM2_KEY * ekKey,
    TPM_ALG_ID alg
)
```

Generates a new TPM Endorsement key, based on the user selected algorithm, RSA or ECC.

Parameters:

- **dev** pointer to a TPM2_DEV struct
- **ekKey** pointer to an empty WOLFTPM2_KEY structure, to store information about the new EK
- **alg** can be only TPM_ALG_RSA or TPM_ALG_ECC, see Note above

See:

- wolfTPM2_CreateSRK
- wolfTPM2_GetKeyTemplate_RSA_EK
- wolfTPM2_GetKeyTemplate_ECC_EK

Return:

- TPM_RC_SUCCESS: successful
- BAD_FUNC_ARG: check the provided arguments

Note: Although only RSA and ECC can be used for EK, symmetric keys can be created and used by the TPM

29.9.6.203 function wolfTPM2_CreateSRK

```
WOLFTPM_API int wolfTPM2_CreateSRK(
    WOLFTPM2_DEV * dev,
    WOLFTPM2_KEY * srkKey,
    TPM_ALG_ID alg,
    const byte * auth,
    int authSz
)
```

Generates a new TPM Primary Key that will be used as a Storage Key for other TPM keys.

Parameters:

- **dev** pointer to a TPM2_DEV struct
- **srkKey** pointer to an empty WOLFTPM2_KEY structure, to store information about the new EK
- **alg** can be only TPM_ALG_RSA or TPM_ALG_ECC, see Note above
- **auth** pointer to a string constant, specifying the password authorization for the TPM 2.0 Key
- **authSz** integer value, specifying the size of the password authorization, in bytes

See:

- wolfTPM2_CreateEK
- wolfTPM2_CreateAndLoadAIK
- wolfTPM2_GetKeyTemplate_RSA_SRK
- wolfTPM2_GetKeyTemplate_ECC_SRK

Return:

- TPM_RC_SUCCESS: successful
- TPM_RC_FAILURE: generic failure (check TPM IO and TPM return code)
- BAD_FUNC_ARG: check the provided arguments

Note: Although only RSA and ECC can be used for EK, symmetric keys can be created and used by the TPM

29.9.6.204 function wolfTPM2_CreateAndLoadAIK

```
WOLFTPM_API int wolfTPM2_CreateAndLoadAIK(
    WOLFTPM2_DEV * dev,
    WOLFTPM2_KEY * aikKey,
    TPM_ALG_ID alg,
    WOLFTPM2_KEY * srkKey,
    const byte * auth,
    int authSz
)
```

Generates a new TPM Attestation Key under the provided Storage Key.

Parameters:

- **dev** pointer to a TPM2_DEV struct
- **aikKey** pointer to an empty WOLFTPM2_KEY structure, to store the newly generated TPM key
- **alg** can be only TPM_ALG_RSA or TPM_ALG_ECC
- **srkKey** pointer to a WOLFTPM2_KEY structure, pointing to valid TPM handle of a loaded Storage Key
- **auth** pointer to a string constant, specifying the password authorization for the TPM 2.0 Key
- **authSz** integer value, specifying the size of the password authorization, in bytes

See:

- wolfTPM2_CreateSRK
- wolfTPM2_GetKeyTemplate_RSA_AIK
- wolfTPM2_GetKeyTemplate_ECC_AIK

Return:

- TPM_RC_SUCCESS: successful
- TPM_RC_FAILURE: generic failure (check TPM IO and TPM return code)
- BAD_FUNC_ARG: check the provided arguments

29.9.6.205 function wolfTPM2_GetTime

```
WOLFTPM_API int wolfTPM2_GetTime(
    WOLFTPM2_KEY * aikKey,
    GetTime_Out * getTimeOut
)
```

One-shot API to generate a TPM signed timestamp.

Parameters:

- **aikKey** pointer to a WOLFTPM2_KEY structure, containing valid TPM handle of a loaded attestation key
- **getTimeOut** pointer to an empty structure of GetTime_Out type, to store the output of the command

See:

- wolfTPM2_CreateSRK
- wolfTPM2_GetKeyTemplate_RSA_EK
- wolfTPM2_GetKeyTemplate_ECC_EK

Return:

- TPM_RC_SUCCESS: successful
- TPM_RC_FAILURE: generic failure (check TPM IO and TPM return code)
- BAD_FUNC_ARG: check the provided arguments

Note: The attestation key must be generated and loaded prior to this call

29.9.6.206 function wolfTPM2_CSR_SetCustomExt

```
WOLFTPM_API int wolfTPM2_CSR_SetCustomExt(
    WOLFTPM2_DEV * dev,
    WOLFTPM2_CSR * csr,
    int critical,
    const char * oid,
    const byte * der,
    word32 derSz
)
```

Helper for Certificate Signing Request (CSR) generation to set a custom request extension oid and value usage for a WOLFTPM2_CSR structure.

Parameters:

- **dev** pointer to a TPM2_DEV struct (not used)
- **csr** pointer to a WOLFTPM2_CSR structure
- **critical** If 0, the extension will not be marked critical, otherwise it will be marked critical.
- **oid** Dot separated oid as a string. For example "1.2.840.10045.3.1.7"
- **der** The der encoding of the content of the extension.
- **derSz** The size in bytes of the der encoding.

See:

- wolfTPM2_CSR_SetSubject
- wolfTPM2_CSR_SetKeyUsage
- wolfTPM2_CSR_MakeAndSign
- wolfTPM2_CSR_MakeAndSign_ex

Return:

- TPM_RC_SUCCESS: successful
- BAD_FUNC_ARG: check the provided arguments

29.9.6.207 function wolfTPM2_CSR_SetKeyUsage

```
WOLFTPM_API int wolfTPM2_CSR_SetKeyUsage(
    WOLFTPM2_DEV * dev,
    WOLFTPM2_CSR * csr,
    const char * keyUsage
)
```

Helper for Certificate Signing Request (CSR) generation to set a extended key usage or key usage for a WOLFTPM2_CSR structure. Pass either extended key usage or key usage values. Mixed string types are not supported, however you can call `wolfTPM2_CSR_SetKeyUsage` twice (once for extended key usage strings and once for standard key usage strings).

Parameters:

- **dev** pointer to a TPM2_DEV struct (not used)
- **csr** pointer to a WOLFTPM2_CSR structure
- **keyUsage** string list of comma separated key usage attributes. Possible Extended Key Usage values: any, serverAuth, clientAuth, codeSigning, emailProtection, timeStamping and OCSPSigning Possible Key Usage values: digitalSignature, nonRepudiation, contentCommitment, keyEncipherment, dataEncipherment, keyAgreement, keyCertSign, cRLSign, encipherOnly, decipherOnly Default: "serverAuth,clientAuth,codeSigning"

See:

- wolfTPM2_CSR_SetSubject

- wolfTPM2_CSR_SetCustomExt
- wolfTPM2_CSR_MakeAndSign
- wolfTPM2_CSR_MakeAndSign_ex

Return:

- TPM_RC_SUCCESS: successful
- BAD_FUNC_ARG: check the provided arguments

29.9.6.208 function wolfTPM2_CSR_SetSubject

```
WOLFTPM_API int wolfTPM2_CSR_SetSubject(
    WOLFTPM2_DEV * dev,
    WOLFTPM2_CSR * csr,
    const char * subject
)
```

Helper for Certificate Signing Request (CSR) generation to set a subject for a WOLFTPM2_CSR structure.

Parameters:

- **dev** pointer to a TPM2_DEV struct (not used)
- **csr** pointer to a WOLFTPM2_CSR structure
- **subject** distinguished name string using /CN= syntax. Example: "/C=US/ST=Washington/L=Seattle/O=wolfSSL/O

See:

- wolfTPM2_CSR_SetKeyUsage
- wolfTPM2_CSR_SetCustomExt
- wolfTPM2_CSR_MakeAndSign
- wolfTPM2_CSR_MakeAndSign_ex

Return:

- TPM_RC_SUCCESS: successful
- BAD_FUNC_ARG: check the provided arguments

29.9.6.209 function wolfTPM2_CSR_MakeAndSign_ex

```
WOLFTPM_API int wolfTPM2_CSR_MakeAndSign_ex(
    WOLFTPM2_DEV * dev,
    WOLFTPM2_CSR * csr,
    WOLFTPM2_KEY * key,
    int outFormat,
    byte * out,
    int outSz,
    int sigType,
    int selfSignCert,
    int devId
)
```

Helper for Certificate Signing Request (CSR) generation using a TPM based key (WOLFTPM2_KEY structure with subject and key usage already set.

Parameters:

- **dev** pointer to a TPM2_DEV struct
- **csr** pointer to a WOLFTPM2_CSR structure
- **key** WOLFTPM2_KEY structure
- **outFormat** CTC_FILETYPE_ASN1 or CTC_FILETYPE_PEM

- **out** destination buffer for CSR as ASN.1/DER or PEM
- **outSz** destination buffer maximum size
- **sigType** Use 0 to automatically select SHA2-256 based on keyType (CTC_SHA256wRSA or CTC_SHA256wECDSA). See wolfCrypt "enum Ctc_SigType" for list of possible values.
- **selfSignCert** If set to 1 (non-zero) then result will be a self signed certificate. Zero (0) will generate a CSR (Certificate Signing Request) to be used by a CA.
- **devId** The device identifier used when registering the crypto callback. Use INVALID_DEVID (-2) to automatically register the required crypto callback.

See:

- wolfTPM2_CSR_SetSubject
- wolfTPM2_CSR_SetKeyUsage
- wolfTPM2_CSR_SetCustomExt
- wolfTPM2_CSR_MakeAndSign

Return:

- Success: Positive integer (size of the output)
- TPM_RC_FAILURE: generic failure (check TPM IO and TPM return code)
- BAD_FUNC_ARG: check the provided arguments

29.9.6.210 function wolfTPM2_CSR_MakeAndSign

```
WOLFTPM_API int wolfTPM2_CSR_MakeAndSign(
    WOLFTPM2_DEV * dev,
    WOLFTPM2_CSR * csr,
    WOLFTPM2_KEY * key,
    int outFormat,
    byte * out,
    int outSz
)
```

Helper for Certificate Signing Request (CSR) generation using a TPM based key (WOLFTPM2_KEY structure with subject and key usage already set.

Parameters:

- **dev** pointer to a TPM2_DEV struct
- **csr** pointer to a WOLFTPM2_CSR structure
- **key** WOLFTPM2_KEY structure
- **outFormat** CTC_FILETYPE_ASN1 or CTC_FILETYPE_PEM
- **out** destination buffer for CSR as ASN.1/DER or PEM
- **outSz** destination buffer maximum size

See:

- wolfTPM2_CSR_SetSubject
- wolfTPM2_CSR_SetKeyUsage
- wolfTPM2_CSR_SetCustomExt
- wolfTPM2_CSR_MakeAndSign_ex

Return:

- Success: Positive integer (size of the output)
- TPM_RC_FAILURE: generic failure (check TPM IO and TPM return code)
- BAD_FUNC_ARG: check the provided arguments

29.9.6.211 function wolfTPM2_CSR_Generate_ex

```
WOLFTPM_API int wolfTPM2_CSR_Generate_ex(
    WOLFTPM2_DEV * dev,
    WOLFTPM2_KEY * key,
    const char * subject,
    const char * keyUsage,
    int outFormat,
    byte * out,
    int outSz,
    int sigType,
    int selfSignCert,
    int devId
)
```

Helper for Certificate Signing Request (CSR) generation using a TPM based key (WOLFTPM2_KEY). Single shot API for outputting a CSR or self-signed cert based on TPM key.

Parameters:

- **dev** pointer to a TPM2_DEV struct
- **key** pointer to a loaded WOLFTPM2_KEY structure
- **subject** distinguished name string using /CN= syntax. Example: "/C=US/ST=Washington/L=Seattle/O=wolfSSL/O
- **keyUsage** string list of comma separated key usage attributes. Possible values: any, serverAuth, clientAuth, codeSigning, emailProtection, timeStamping and OCSPSigning Default: "serverAuth,clientAuth,codeSigning"
- **outFormat** CTC_FILETYPE_ASN1 or CTC_FILETYPE_PEM
- **out** destination buffer for CSR as ASN.1/DER or PEM
- **outSz** destination buffer maximum size
- **sigType** Use 0 to automatically select SHA2-256 based on keyType (CTC_SHA256wRSA or CTC_SHA256wECDSA). See wolfCrypt "enum Ctc_SigType" for list of possible values.
- **selfSignCert** If set to 1 (non-zero) then result will be a self signed certificate. Zero (0) will generate a CSR (Certificate Signing Request) to be used by a CA.
- **devId** The device identifier used when registering the crypto callback. Use INVALID_DEVID (-2) to automatically register the required crypto callback.
- **dev** pointer to a TPM2_DEV struct
- **key** pointer to a loaded WOLFTPM2_KEY structure
- **subject** distinguished name string using /CN= syntax
- **keyUsage** string list of comma separated key usage attributes
- **outFormat** output format (CTC_FILETYPE_ASN1 or CTC_FILETYPE_PEM)
- **out** pointer to buffer for CSR/cert output
- **outSz** size of output buffer
- **sigType** signature algorithm (0 for default SHA2-256)
- **selfSignCert** If 1, generate self-signed cert; if 0, generate CSR
- **devId** device ID for crypto callback (-2 for auto-register)

See:

- wolfTPM2_SetCryptoDevCb
- wolfTPM2_CSR_Generate
- wolfTPM2_CSR_Generate
- wolfTPM2_CSR_MakeAndSign_ex

Return:

- Success: Positive integer (size of the output)
- TPM_RC_FAILURE: generic failure (check TPM IO and TPM return code)
- BAD_FUNC_ARG: check the provided arguments

- Success: Positive integer (size of the output)
- TPM_RC_FAILURE: generic failure (check TPM IO and TPM return code)
- BAD_FUNC_ARG: check the provided arguments

Note: Single shot API for outputting a CSR or self-signed cert based on TPM key

Generate a Certificate Signing Request (CSR) or self-signed certificate with extended options

29.9.6.212 function wolfTPM2_CSR_Generate

```
WOLFTPM_API int wolfTPM2_CSR_Generate(
    WOLFTPM2_DEV * dev,
    WOLFTPM2_KEY * key,
    const char * subject,
    const char * keyUsage,
    int outFormat,
    byte * out,
    int outSz
)
```

Helper for Certificate Signing Request (CSR) generation using a TPM based key (WOLFTPM2_KEY). Single shot API for outputting a CSR or self-signed cert based on TPM key.

Parameters:

- **dev** pointer to a TPM2_DEV struct
- **key** pointer to a loaded WOLFTPM2_KEY structure
- **subject** distinguished name string using /CN= syntax. Example: "/C=US/ST=Washington/L=Seattle/O=wolfSSL/O
- **keyUsage** string list of comma separated key usage attributes. Possible values: any, serverAuth, clientAuth, codeSigning, emailProtection, timeStamping and OCSPSigning Default: "serverAuth,clientAuth,codeSigning"
- **outFormat** CTC_FILETYPE_ASN1 or CTC_FILETYPE_PEM
- **out** destination buffer for CSR as ASN.1/DER or PEM
- **outSz** destination buffer maximum size

See:

- wolfTPM2_SetCryptoDevCb
- wolfTPM2_CSR_Generate_ex

Return:

- Success: Positive integer (size of the output)
- TPM_RC_FAILURE: generic failure (check TPM IO and TPM return code)
- BAD_FUNC_ARG: check the provided arguments

29.9.6.213 function wolfTPM2_ChangePlatformAuth

```
WOLFTPM_API int wolfTPM2_ChangePlatformAuth(
    WOLFTPM2_DEV * dev,
    WOLFTPM2_SESSION * session
)
```

Helper to set the platform hierarchy authentication value to random. Setting the platform auth to random value is used to prevent application from being able to use platform hierarchy. This is defined in section 10 of the TCG PC Client Platform specification.

Parameters:

- **dev** pointer to a TPM2_DEV struct

- **session** the current session, a session is required to protect the new platform auth

See:

- TPM2_HierarchyChangeAuth
- wolfTPM2_ChangeHierarchyAuth

Return:

- TPM_RC_SUCCESS (0) on success
- TPM_RC_FAILURE: generic failure (check TPM IO and TPM return code)
- BAD_FUNC_ARG: check the provided arguments

29.9.6.214 function wolfTPM2_ChangeHierarchyAuth

```
WOLFTPM_API int wolfTPM2_ChangeHierarchyAuth(
    WOLFTPM2_DEV * dev,
    WOLFTPM2_SESSION * session,
    TPMI_RH_HIERARCHY_AUTH authHandle
)
```

Helper to set the hierarchy authentication value to random. Setting the platform auth to random value is used to prevent application from being able to use platform hierarchy. This is defined in section 10 of the TCG PC Client Platform specification.

Parameters:

- **dev** pointer to a TPM2_DEV struct
- **session** the current session, a session is required to protect the new platform auth
- **authHandle** the auth hierarchy (example: TPM_RH_PLATFORM or TPM_RH_LOCKOUT)

See:

- TPM2_HierarchyChangeAuth
- wolfTPM2_ChangePlatformAuth

Return:

- TPM_RC_SUCCESS (0) on success
- TPM_RC_FAILURE: generic failure (check TPM IO and TPM return code)
- BAD_FUNC_ARG: check the provided arguments

29.9.6.215 function wolfTPM2_EncryptSecret

```
WOLFTPM_API int wolfTPM2_EncryptSecret(
    WOLFTPM2_DEV * dev,
    const WOLFTPM2_KEY * tpmKey,
    TPM2B_DATA * secret,
    TPM2B_ENCRYPTED_SECRET * encSecret,
    const char * label
)
```

29.9.6.216 function wolfTPM2_CryptoDevCb

```
WOLFTPM_API int wolfTPM2_CryptoDevCb(
    int devId,
    wc_CryptoInfo * info,
    void * ctx
)
```

A reference crypto callback API for using the TPM for crypto offload. This callback function is registered using wolfTPM2_SetCryptoDevCb or wc_CryptoDev_RegisterDevice.

Parameters:

- **devId** The devId used when registering the callback. Any signed integer value besides INVALID_DEVID
- **info** point to wc_CryptoInfo structure with detailed information about crypto type and parameters
- **ctx** The user context supplied when callback was registered with wolfTPM2_SetCryptoDevCb

See:

- wolfTPM2_SetCryptoDevCb
- wolfTPM2_ClearCryptoDevCb

Return:

- TPM_RC_SUCCESS: successful
- CRYPTOCB_UNAVAILABLE: Do not use TPM hardware, fall-back to default software crypto.
- WC_HW_E: generic hardware failure

Note: Key generation, signing and RSA operations on an ECC, RSA or ML-DSA key whose key id is a 4-byte big-endian persistent or transient handle use the TPM key at that handle with an empty authorization value. RSA public operations on an RsaKey that has a modulus use that modulus instead. The TPM key must be unrestricted and must fit the operation: RSA with the NULL scheme and the decrypt attribute, ECC with an ECDSA scheme or the NULL scheme, which signs with the curve's hash, where the hash is no shorter than the input and no longer than the curve, or pure ML-DSA with the parameter set given to wc_MlDsaKey_SetParams. The operation also fails when the wolfCrypt key holds a different public key or only a private key; without either, a transient id names whatever key is loaded at that handle. Key generation creates the key under storageKey with an empty auth and noDA. A persistent handle stores it there, which needs an empty owner authorization. WOLFTPM2_KEYID_TRANSIENT leaves it loaded at a handle the TPM picks and writes that handle to the key id; the caller unloads it with wolfTPM2_UnloadHandle. Key generation fails for any other key id, for a wolfCrypt key that already holds a key and for a curve shorter than its hash (P-192, P-224).

29.9.6.217 function wolfTPM2_SetCryptoDevCb

```
WOLFTPM_API int wolfTPM2_SetCryptoDevCb(
    WOLFTPM2_DEV * dev,
    CryptoDevCallbackFunc cb,
    TpmCryptoDevCtx * tpmCtx,
    int * pDevId
)
```

Register a crypto callback function and return assigned devId.

Parameters:

- **dev** pointer to a TPM2_DEV struct
- **cb** The wolfTPM2_CryptoDevCb API is a template, but you can also provide your own
- **tpmCtx** The user supplied context. For wolfTPM2_CryptoDevCb use TpmCryptoDevCtx, but can also be your own.
- **pDevId** Pointer to automatically assigned device ID.

See:

- wolfTPM2_CryptoDevCb
- wolfTPM2_ClearCryptoDevCb

Return:

- TPM_RC_SUCCESS: successful
- TPM_RC_FAILURE: generic failure (check TPM IO and TPM return code)
- BAD_FUNC_ARG: check the provided arguments

29.9.6.218 function wolfTPM2_ClearCryptoDevCb

```
WOLFTPM_API int wolfTPM2_ClearCryptoDevCb(
    WOLFTPM2_DEV * dev,
    int devId
)
```

Clears the registered crypto callback.

Parameters:

- **dev** pointer to a TPM2_DEV struct
- **devId** The devId used when registering the callback

See:

- wolfTPM2_CryptoDevCb
- wolfTPM2_SetCryptoDevCb

Return:

- TPM_RC_SUCCESS: successful
- TPM_RC_FAILURE: generic failure (check TPM IO and TPM return code)
- BAD_FUNC_ARG: check the provided arguments

29.9.6.219 function wolfTPM2_PK_RsaSign

```
WOLFTPM_API int wolfTPM2_PK_RsaSign(
    WOLFSSL * ssl,
    const unsigned char * in,
    unsigned int inSz,
    unsigned char * out,
    word32 * outSz,
    const unsigned char * keyDer,
    unsigned int keySz,
    void * ctx
)
```

29.9.6.220 function wolfTPM2_PK_RsaSignCheck

```
WOLFTPM_API int wolfTPM2_PK_RsaSignCheck(
    WOLFSSL * ssl,
    unsigned char * sig,
    unsigned int sigSz,
    unsigned char ** out,
    const unsigned char * keyDer,
    unsigned int keySz,
    void * ctx
)
```

29.9.6.221 function wolfTPM2_PK_RsaPssSign

```
WOLFTPM_API int wolfTPM2_PK_RsaPssSign(
    WOLFSSL * ssl,
    const unsigned char * in,
    unsigned int inSz,
    unsigned char * out,
    unsigned int * outSz,
    int hash,
    int mgf,
    const unsigned char * keyDer,
    unsigned int keySz,
    void * ctx
)
```

29.9.6.222 function wolfTPM2_PK_RsaPssSignCheck

```
WOLFTPM_API int wolfTPM2_PK_RsaPssSignCheck(
    WOLFSSL * ssl,
    unsigned char * sig,
    unsigned int sigSz,
    unsigned char ** out,
    int hash,
    int mgf,
    const unsigned char * keyDer,
    unsigned int keySz,
    void * ctx
)
```

29.9.6.223 function wolfTPM2_PK_EccSign

```
WOLFTPM_API int wolfTPM2_PK_EccSign(
    WOLFSSL * ssl,
    const unsigned char * in,
    unsigned int inSz,
    unsigned char * out,
    word32 * outSz,
    const unsigned char * keyDer,
    unsigned int keySz,
    void * ctx
)
```

29.9.6.224 function wolfTPM_PK_SetCb

```
WOLFTPM_API int wolfTPM_PK_SetCb(
    WOLFSSL_CTX * ctx
)
```

29.9.6.225 function wolfTPM_PK_SetCbCtx

```
WOLFTPM_API int wolfTPM_PK_SetCbCtx(
    WOLFSSL * ssl,
    void * userCtx
)
```

29.9.6.226 function wolfTPM2_New

```
WOLFTPM_API WOLFTPM2_DEV * wolfTPM2_New(
    void
)
```

Allocate and initialize a WOLFTPM2_DEV.

See:

- wolfTPM2_Free
- wolfTPM2_InitWithSpdmKey
- wolfTPM2_InitWithSpdmPsk

Return:

- pointer to new device struct
- NULL: on any error

Note: This convenience constructor uses uncredentialed wolfTPM2_Init. An SPDM-only TPM requires a caller-allocated WOLFTPM2_DEV initialized with wolfTPM2_InitWithSpdmKey, wolfTPM2_InitWithSpdmKey_ex, or wolfTPM2_InitWithSpdmPsk instead.

29.9.6.227 function wolfTPM2_Free

```
WOLFTPM_API int wolfTPM2_Free(
    WOLFTPM2_DEV * dev
)
```

Cleanup and Free a WOLFTPM2_DEV that was allocated by wolfTPM2_New.

Parameters:

- **dev** pointer to a TPM2_DEV struct

See: wolfTPM2_New

Return: TPM_RC_SUCCESS: successful

29.9.6.228 function wolfTPM2_NewKeyBlob

```
WOLFTPM_API WOLFTPM2_KEYBLOB * wolfTPM2_NewKeyBlob(
    void
)
```

Allocate and initialize a WOLFTPM2_KEYBLOB.

See: wolfTPM2_FreeKeyBlob

Return:

- pointer to newly initialized WOLFTPM2_KEYBLOB
- NULL on any error

29.9.6.229 function wolfTPM2_FreeKeyBlob

```
WOLFTPM_API int wolfTPM2_FreeKeyBlob(
    WOLFTPM2_KEYBLOB * blob
)
```

Free a WOLFTPM2_KEYBLOB that was allocated with wolfTPM2_NewKeyBlob.

Parameters:

- **blob** pointer to a WOLFTPM2_KEYBLOB that was allocated by wolfTPM2_NewKeyBlob

See: wolfTPM2_NewKeyBlob

Return: TPM_RC_SUCCESS: successful

29.9.6.230 function wolfTPM2_NewPublicTemplate

```
WOLFTPM_API TPMT_PUBLIC * wolfTPM2_NewPublicTemplate(
    void
)
```

Allocate and initialize a TPMT_PUBLIC.

See: wolfTPM2_FreePublicTemplate

Return:

- pointer to newly initialized
- NULL on any error

29.9.6.231 function wolfTPM2_FreePublicTemplate

```
WOLFTPM_API int wolfTPM2_FreePublicTemplate(
    TPMT_PUBLIC * PublicTemplate
)
```

Free a TPMT_PUBLIC that was allocated with wolfTPM2_NewPublicTemplate.

Parameters:

- **PublicTemplate** pointer to a TPMT_PUBLIC that was allocated with wolfTPM2_NewPublicTemplate

See: wolfTPM2_NewPublicTemplate

Return: TPM_RC_SUCCESS: successful

29.9.6.232 function wolfTPM2_NewKey

```
WOLFTPM_API WOLFTPM2_KEY * wolfTPM2_NewKey(
    void
)
```

Allocate and initialize a WOLFTPM2_KEY.

See: wolfTPM2_FreeKey

Return:

- pointer to newly initialized WOLFTPM2_KEY
- NULL on any error

29.9.6.233 function wolfTPM2_FreeKey

```
WOLFTPM_API int wolfTPM2_FreeKey(
    WOLFTPM2_KEY * key
)
```

Free a WOLFTPM2_KEY that was allocated with wolfTPM2_NewKey.

Parameters:

- **key** pointer to a WOLFTPM2_KEY that was allocated by wolfTPM2_NewKey

See: wolfTPM2_NewKey

Return: TPM_RC_SUCCESS: successful

29.9.6.234 function wolfTPM2_NewSession

```
WOLFTPM_API WOLFTPM2_SESSION * wolfTPM2_NewSession(
    void
)
```

Allocate and initialize a WOLFTPM2_SESSION.

See: wolfTPM2_FreeSession

Return:

- pointer to newly initialized WOLFTPM2_SESSION
- NULL on any error

29.9.6.235 function wolfTPM2_FreeSession

```
WOLFTPM_API int wolfTPM2_FreeSession(
    WOLFTPM2_SESSION * session
)
```

Free a WOLFTPM2_SESSION that was allocated with wolfTPM2_NewSession.

Parameters:

- **session** pointer to a WOLFTPM2_SESSION struct

See: wolfTPM2_NewSession

Return: TPM_RC_SUCCESS: successful

29.9.6.236 function wolfTPM2_NewCSR

```
WOLFTPM_API WOLFTPM2_CSR * wolfTPM2_NewCSR(
    void
)
```

Allocate and initialize a WOLFTPM2_CSR.

See: wolfTPM2_FreeCSR

Return:

- pointer to newly initialized WOLFTPM2_CSR
- NULL on any error

29.9.6.237 function wolfTPM2_FreeCSR

```
WOLFTPM_API int wolfTPM2_FreeCSR(
    WOLFTPM2_CSR * csr
)
```

Free a WOLFTPM2_CSR that was allocated with wolfTPM2_NewCSR.

Parameters:

- **csr** pointer to a WOLFTPM2_CSR that was allocated by wolfTPM2_NewCSR

See: wolfTPM2_NewCSR

Return: TPM_RC_SUCCESS: successful

29.9.6.238 function wolfTPM2_GetHandleRefFromKey

```
WOLFTPM_API WOLFTPM2_HANDLE * wolfTPM2_GetHandleRefFromKey(
    WOLFTPM2_KEY * key
)
```

Retrieve the WOLFTPM2_HANDLE from a WOLFTPM2_KEY.

Parameters:

- **key** pointer to a WOLFTPM2_KEY struct

Return:

- pointer to handle in the key structure
- NULL if key pointer is NULL

29.9.6.239 function wolfTPM2_GetHandleRefFromKeyBlob

```
WOLFTPM_API WOLFTPM2_HANDLE * wolfTPM2_GetHandleRefFromKeyBlob(
    WOLFTPM2_KEYBLOB * keyBlob
)
```

Retrieve the WOLFTPM2_HANDLE from a WOLFTPM2_KEYBLOB.

Parameters:

- **keyBlob** pointer to a WOLFTPM2_KEYBLOB struct

Return:

- pointer to handle in the key blob structure
- NULL if key pointer is NULL

29.9.6.240 function wolfTPM2_GetHandleRefFromSession

```
WOLFTPM_API WOLFTPM2_HANDLE * wolfTPM2_GetHandleRefFromSession(
    WOLFTPM2_SESSION * session
)
```

Retrieve the WOLFTPM2_HANDLE from a WOLFTPM2_SESSION.

Parameters:

- **session** pointer to a WOLFTPM2_SESSION struct

Return:

- pointer to handle in the session structure
- NULL if key pointer is NULL

29.9.6.241 function wolfTPM2_GetHandleValue

```
WOLFTPM_API TPM_HANDLE wolfTPM2_GetHandleValue(
    WOLFTPM2_HANDLE * handle
)
```

Get the 32-bit handle value from the WOLFTPM2_HANDLE.

Parameters:

- **handle** pointer to WOLFTPM2_HANDLE structure

Return: TPM_HANDLE value from TPM

29.9.6.242 function wolfTPM2_SetKeyAuthPassword

```
WOLFTPM_API int wolfTPM2_SetKeyAuthPassword(
    WOLFTPM2_KEY * key,
    const byte * auth,
    int authSz
)
```

Set the authentication data for a key.

Parameters:

- **key** pointer to wrapper key struct
- **auth** pointer to auth data
- **authSz** length in bytes of auth data

Return:

- TPM_RC_SUCCESS: successful
- BAD_FUNC_ARG: check the provided arguments

29.9.6.243 function wolfTPM2_GetKeyBlobAsBuffer

```
WOLFTPM_API int wolfTPM2_GetKeyBlobAsBuffer(
    byte * buffer,
    word32 bufferSz,
    WOLFTPM2_KEYBLOB * key
)
```

Marshal data from a keyblob to a binary buffer. This can be stored to disk for loading in a separate process or after power cycling. If buffer is not provided then size only will be returned.

Parameters:

- **buffer** pointer to buffer in which to store marshaled keyblob
- **bufferSz** size of the above buffer
- **key** pointer to keyblob to marshal

See: wolfTPM2_SetKeyBlobFromBuffer

Return:

- Positive integer (size of the output)
- BUFFER_E: insufficient space in provided buffer
- BAD_FUNC_ARG: check the provided arguments

29.9.6.244 function wolfTPM2_GetKeyBlobAsSeparateBuffers

```
WOLFTPM_API int wolfTPM2_GetKeyBlobAsSeparateBuffers(
    byte * pubBuffer,
    word32 * pubBufferSz,
    byte * privBuffer,
    word32 * privBufferSz,
```

```
    WOLFTPM2_KEYBLOB * key
)
```

Marshal data from a keyblob to a binary buffer. This can be stored to disk for loading in a separate process or after power cycling. If either buffer is NULL then the size will be returned for each part.

Parameters:

- **pubBuffer** pointer to buffer in which to store the public part of the marshaled keyblob
- **pubBufferSz** pointer to the size of the above buffer
- **privBuffer** pointer to buffer in which to store the private part of the marshaled keyblob
- **privBufferSz** pointer to the size of the above buffer
- **key** pointer to keyblob to marshal

See: wolfTPM2_GetKeyBlobAsSeparateBuffers

Return:

- TPM_RC_SUCCESS: successful
- BUFFER_E: insufficient space in provided buffer
- BAD_FUNC_ARG: check the provided arguments
- LENGTH_ONLY_E: Returning length only (when either of the buffers is NULL)

29.9.6.245 function wolfTPM2_SetKeyBlobFromBuffer

```
WOLFTPM_API int wolfTPM2_SetKeyBlobFromBuffer(
    WOLFTPM2_KEYBLOB * key,
    byte * buffer,
    word32 bufferSz
)
```

Unmarshal data into a WOLFTPM2_KEYBLOB struct. This can be used to load a keyblob that was previously marshaled by wolfTPM2_GetKeyBlobAsBuffer.

Parameters:

- **key** pointer to keyblob to load and unmarshall data into
- **buffer** pointer to buffer containing marshalled keyblob to load from
- **bufferSz** size of the above buffer

See: wolfTPM2_GetKeyBlobAsBuffer

Return:

- TPM_RC_SUCCESS: successful
- BUFFER_E: buffer is too small or there is extra data remaining and not unmarshalled
- BAD_FUNC_ARG: check the provided arguments

29.9.6.246 function wolfTPM2_PolicyRestart

```
WOLFTPM_API int wolfTPM2_PolicyRestart(
    WOLFTPM2_DEV * dev,
    TPM_HANDLE sessionHandle
)
```

Restart the policy digest for a policy session.

Parameters:

- **dev** pointer to a TPM2_DEV struct
- **sessionHandle** the handle of the current session, a session is required to use policy pcr

See:

- wolfTPM2_GetPolicyDigest
- wolfTPM2_PolicyPCR
- wolfTPM2_PolicyAuthorize

Return:

- TPM_RC_SUCCESS: successful
- INPUT_SIZE_E: policyDigestSz is too small to hold the returned digest
- BAD_FUNC_ARG: check the provided arguments

29.9.6.247 function wolfTPM2_GetPolicyDigest

```
WOLFTPM_API int wolfTPM2_GetPolicyDigest(
    WOLFTPM2_DEV * dev,
    TPM_HANDLE sessionHandle,
    byte * policyDigest,
    word32 * policyDigestSz
)
```

Get the policy digest of the session that was passed in wolfTPM2_GetPolicyDigest.

Parameters:

- **dev** pointer to a TPM2_DEV struct
- **sessionHandle** the handle of the current session, a session is required to use policy pcr
- **policyDigest** output digest of the policy
- **policyDigestSz** pointer to the size of the policyDigest

See:

- wolfTPM2_PolicyPCR
- wolfTPM2_PolicyAuthorize
- wolfTPM2_PolicyRestart

Return:

- TPM_RC_SUCCESS: successful
- INPUT_SIZE_E: policyDigestSz is too small to hold the returned digest
- BAD_FUNC_ARG: check the provided arguments

29.9.6.248 function wolfTPM2_PolicyPCR

```
WOLFTPM_API int wolfTPM2_PolicyPCR(
    WOLFTPM2_DEV * dev,
    TPM_HANDLE sessionHandle,
    TPM_ALG_ID pcrAlg,
    byte * pcrArray,
    word32 pcrArraySz
)
```

Apply the PCR's to the policy digest for the policy session.

Parameters:

- **dev** pointer to a TPM2_DEV struct
- **sessionHandle** the handle of the current policy session, a session is required to use policy PCR
- **pcrAlg** the hash algorithm to use with PCR policy
- **pcrArray** array of PCR Indexes to use when creating the policy
- **pcrArraySz** the number of PCR Indexes in the pcrArray

See:

- wolfTPM2_GetPolicyDigest
- wolfTPM2_PolicyPCR
- wolfTPM2_PolicyAuthorize
- wolfTPM2_PolicyRestart

Return:

- TPM_RC_SUCCESS: successful
- INPUT_SIZE_E: policyDigestSz is too small to hold the returned digest
- BAD_FUNC_ARG: check the provided arguments

29.9.6.249 function wolfTPM2_PolicyTransportSPDM

```
WOLFTPM_API int wolfTPM2_PolicyTransportSPDM(
    WOLFTPM2_DEV * dev,
    TPM_HANDLE sessionHandle,
    const TPM2B_NAME * reqKeyName,
    const TPM2B_NAME * tpmKeyName
)
```

Add TPM2_PolicyTransportSPDM (TPM 2.0 Library v1.84) to a policy session. An entity whose authPolicy contains this term can only be authorized by a command that arrives inside an SPDM session. Optional key names further restrict which SPDM session qualifies.

Parameters:

- **dev** pointer to a TPM2_DEV struct
- **sessionHandle** the handle of the current policy session
- **reqKeyName** requester SPDM key name to bind, or NULL for any
- **tpmKeyName** TPM SPDM key name to bind, or NULL for any

See:

- wolfTPM2_PolicyTransportSPDMMake
- wolfTPM2_GetCapability_SPDMSessionInfo
- wolfTPM2_GetPolicyDigest

Return:

- TPM_RC_SUCCESS: successful
- TPM_RC_VALUE: PolicyTransportSPDM already applied to this session
- TPM_RC_HASH: a name uses an unsupported hash algorithm
- TPM_RC_SIZE: a name is not the size its hash algorithm requires
- BUFFER_E: a name is larger than TPM2B_NAME can hold
- BAD_FUNC_ARG: check the provided arguments

29.9.6.250 function wolfTPM2_PolicyAuthorize

```
WOLFTPM_API int wolfTPM2_PolicyAuthorize(
    WOLFTPM2_DEV * dev,
    TPM_HANDLE sessionHandle,
    const TPM2B_PUBLIC * pub,
    const TPMT_TK_VERIFIED * checkTicket,
    const byte * pcrDigest,
    word32 pcrDigestSz,
    const byte * policyRef,
```

```
    word32 policyRefSz
)
```

Apply the PCR's to the policy digest for the policy session.

Parameters:

- **dev** pointer to a TPM2_DEV struct
- **sessionHandle** the handle of the current policy session, a session is required to use policy PCR
- **pub** pointer to a populated structure of TPM2B_PUBLIC type
- **checkTicket** returns the validation ticket proving the signature for digest was checked
- **pcrDigest** digest for the PCR(s) collected with wolfTPM2_PCRGetDigest
- **pcrDigestSz** size of the PCR digest
- **policyRef** optional nonce
- **policyRefSz** optional nonce size

See:

- wolfTPM2_GetPolicyDigest
- wolfTPM2_PolicyPCR
- wolfTPM2_PolicyAuthorize
- wolfTPM2_PolicyRestart
- wolfTPM2_PCRGetDigest

Return:

- TPM_RC_SUCCESS: successful
- INPUT_SIZE_E: policyDigestSz is too small to hold the returned digest
- BAD_FUNC_ARG: check the provided arguments

29.9.6.251 function wolfTPM2_PCRGetDigest

```
WOLFTPM_API int wolfTPM2_PCRGetDigest(
    WOLFTPM2_DEV * dev,
    TPM_ALG_ID pcrAlg,
    byte * pcrArray,
    word32 pcrArraySz,
    byte * pcrDigest,
    word32 * pcrDigestSz
)
```

Get a cumulative digest of the PCR's specified.

Parameters:

- **dev** pointer to a TPM2_DEV struct
- **pcrAlg** the hash algorithm to use with pcr policy
- **pcrArray** array of pcr Index to use when creating the policy
- **pcrArraySz** the number of Index in the pcrArray
- **pcrDigest** digest for the PCR(s) collected with wolfTPM2_PCRGetDigest
- **pcrDigestSz** size of the PCR digest

See:

- wolfTPM2_PolicyPCR
- wolfTPM2_PolicyAuthorize

Return:

- TPM_RC_SUCCESS: successful
- INPUT_SIZE_E: policyDigestSz is too small to hold the returned digest

- BAD_FUNC_ARG: check the provided arguments

29.9.6.252 function wolfTPM2_PolicyRefMake

```
WOLFTPM_API int wolfTPM2_PolicyRefMake(
    TPM_ALG_ID pcrAlg,
    byte * digest,
    word32 * digestSz,
    const byte * policyRef,
    word32 policyRefSz
)
```

Utility for generating a policy ref digest. If no policy reference (nonce) used then just rehash the provided digest again (update -> final)

Parameters:

- **pcrAlg** the hash algorithm to use with pcr policy
- **digest** input/out digest
- **digestSz** input/out digest size
- **policyRef** optional nonce
- **policyRefSz** optional nonce size

See:

- wolfTPM2_PolicyPCRMake
- wolfTPM2_PolicyAuthorizeMake

Return:

- TPM_RC_SUCCESS: successful
- INPUT_SIZE_E: policyDigestSz is too small to hold the returned digest
- BAD_FUNC_ARG: check the provided arguments

29.9.6.253 function wolfTPM2_PolicyPCRMake

```
WOLFTPM_API int wolfTPM2_PolicyPCRMake(
    TPM_ALG_ID pcrAlg,
    byte * pcrArray,
    word32 pcrArraySz,
    const byte * pcrDigest,
    word32 pcrDigestSz,
    byte * digest,
    word32 * digestSz
)
```

Utility for generating a policy PCR digest.

Parameters:

- **pcrAlg** the supported hash algorithm to use with the PCR policy
- **pcrArray** non-NULL array of PCRs to include in the policy
- **pcrArraySz** number of entries in pcrArray; must be greater than zero
- **pcrDigest** digest for the PCR(s) collected (can get using wolfTPM2_PCRGetDigest); required when pcrDigestSz is nonzero
- **pcrDigestSz** size of the PCR digest
- **digest** input/output policy digest buffer
- **digestSz** input/output: current digest size and buffer capacity on input, which must be at least the selected hash size; selected hash size on output

See:

- wolfTPM2_PolicyPCRMake
- wolfTPM2_PolicyAuthorizeMake
- wolfTPM2_PCRGetDigest

Return:

- TPM_RC_SUCCESS: successful
- BUFFER_E: digest is too small for the selected hash or the PCR policy input exceeds the internal assembly buffer
- BAD_FUNC_ARG: check the provided arguments

Note: To start a fresh policy chain, zero the digest buffer and set digestSz to the selected hash size before calling.

29.9.6.254 function wolfTPM2_PolicyTransportSPDMMake

```
WOLFTPM_API int wolfTPM2_PolicyTransportSPDMMake(
    TPM_ALG_ID hashAlg,
    const TPM2B_NAME * reqKeyName,
    const TPM2B_NAME * tpmKeyName,
    byte * digest,
    word32 * digestSz
)
```

Compute the policy digest that TPM2_PolicyTransportSPDM produces, without a TPM. Use it to build the authPolicy of an NV index or key that must only be reachable over an SPDM session.

Parameters:

- **hashAlg** the policy session hash algorithm
- **reqKeyName** requester SPDM key name to bind, or NULL for any
- **tpmKeyName** TPM SPDM key name to bind, or NULL for any
- **digest** input/output: policyDigestOld in (zeros for a fresh session), policyDigestNew out
- **digestSz** input/output: current digest size and buffer capacity on input, selected hash size on output

See:

- wolfTPM2_PolicyTransportSPDM
- wolfTPM2_PolicyHash
- wolfTPM2_NVCreateAuthPolicy

Return:

- TPM_RC_SUCCESS: successful
- BUFFER_E: digestSz is too small for the selected hash
- BAD_FUNC_ARG: check the provided arguments

29.9.6.255 function wolfTPM2_PolicyCommandCodeMake

```
WOLFTPM_API int wolfTPM2_PolicyCommandCodeMake(
    TPM_ALG_ID hashAlg,
    byte * digest,
    word32 * digestSz,
    TPM_CC cc
)
```

Compute the policy digest for PolicyCommandCode on a fresh session.

Parameters:

- **hashAlg** hash algorithm for the policy digest
- **digest** output policy digest buffer
- **digestSz** in/out: input buffer capacity, output digest size
- **cc** the command code to bind (for example TPM_CC_NV_Read)

See:

- wolfTPM2_PolicyCommandCode
- wolfTPM2_PolicyHash
- wolfTPM2_PolicyPCRMake

Return:

- TPM_RC_SUCCESS: successful
- BAD_FUNC_ARG: NULL digest/digestSz or unsupported hashAlg
- BUFFER_E: *digestSz (the buffer capacity) is smaller than the hash size for hashAlg

Note:

- policyDigest = hash(zeroDigest || TPM_CC_PolicyCommandCode || cc). Mirrors the running digest of a new policy session after wolfTPM2_PolicyCommandCode.
- digestSz is in/out. On input it is the capacity of the digest buffer in bytes, which must be at least the hash size or BUFFER_E is returned without writing to digest. On output it is the hash size written.

29.9.6.256 function wolfTPM2_PolicyHash

```
WOLFTPM_API int wolfTPM2_PolicyHash(
    TPM_ALG_ID hashAlg,
    byte * digest,
    word32 * digestSz,
    TPM_CC cc,
    const byte * input,
    word32 inputSz
)
```

Utility for creating a policy hash. Generic helper that takes command code and input array. policyDigestnew = hash(policyDigestOld || [cc] || [Input])

Parameters:

- **hashAlg** the hash algorithm to use with pcr policy
- **digest** input/out digest (input “old” / output “new”)
- **digestSz** input/out digest size
- **cc** is the command code used
- **input** pointer to a array to use (optional)
- **inputSz** size of input

See: wolfTPM2_PolicyPCRMake

Return:

- TPM_RC_SUCCESS: successful
- INPUT_SIZE_E: policyDigestSz is too small to hold the returned digest
- BAD_FUNC_ARG: check the provided arguments

29.9.6.257 function wolfTPM2_PolicyAuthorizeMake

```
WOLFTPM_API int wolfTPM2_PolicyAuthorizeMake(
    TPM_ALG_ID pcrAlg,
    const TPM2B_PUBLIC * pub,
    byte * digest,
    word32 * digestSz,
    const byte * policyRef,
    word32 policyRefSz
)
```

Utility for generating a policy authorization digest based on a public key.

Parameters:

- **pcrAlg** the hash algorithm to use with pcr policy
- **pub** pointer to a populated structure of TPM2B_PUBLIC type
- **digest** input/out digest
- **digestSz** input/out digest size
- **policyRef** optional nonce
- **policyRefSz** optional nonce size

See:

- wolfTPM2_PolicyPCRMake
- wolfTPM2_PolicyHash

Return:

- TPM_RC_SUCCESS: successful
- INPUT_SIZE_E: policyDigestSz is too small to hold the returned digest
- BAD_FUNC_ARG: check the provided arguments

29.9.6.258 function wolfTPM2_PolicyPassword

```
WOLFTPM_API int wolfTPM2_PolicyPassword(
    WOLFTPM2_DEV * dev,
    WOLFTPM2_SESSION * tpmSession,
    const byte * auth,
    int authSz
)
```

Wrapper for setting a policy password and calling TPM2_PolicyPassword. This will set a password (in clear) for the policy session instead of HMAC.

Parameters:

- **dev** pointer to a TPM2_DEV struct
- **tpmSession** pointer to a WOLFTPM2_SESSION struct used with wolfTPM2_StartSession and wolfTPM2_SetAuthSession
- **auth** pointer to a string constant, specifying the password authorization for the policy session
- **authSz** integer value, specifying the size of the password authorization, in bytes

See:

- wolfTPM2_PolicyAuthValue
- wolfTPM2_PolicyCommandCode

Return:

- TPM_RC_SUCCESS: successful
- BAD_FUNC_ARG: check the provided arguments

29.9.6.259 function wolfTPM2_PolicyAuthValue

```
WOLFTPM_API int wolfTPM2_PolicyAuthValue(
    WOLFTPM2_DEV * dev,
    WOLFTPM2_SESSION * tpmSession,
    const byte * auth,
    int authSz
)
```

Wrapper for setting a policy auth value that is added to the HMAC key for a policy session.

Parameters:

- **dev** pointer to a TPM2_DEV struct
- **tpmSession** pointer to a WOLFTPM2_SESSION struct used with wolfTPM2_StartSession and wolfTPM2_SetAuthSession
- **auth** pointer to a string constant, specifying the password authorization for the policy session
- **authSz** integer value, specifying the size of the password authorization, in bytes

See:

- wolfTPM2_PolicyPassword
- wolfTPM2_PolicyCommandCode

Return:

- TPM_RC_SUCCESS: successful
- BAD_FUNC_ARG: check the provided arguments

29.9.6.260 function wolfTPM2_PolicyCommandCode

```
WOLFTPM_API int wolfTPM2_PolicyCommandCode(
    WOLFTPM2_DEV * dev,
    WOLFTPM2_SESSION * tpmSession,
    TPM_CC cc
)
```

Wrapper for setting a policy command code.

Parameters:

- **dev** pointer to a TPM2_DEV struct
- **tpmSession** pointer to a WOLFTPM2_SESSION struct used with wolfTPM2_StartSession and wolfTPM2_SetAuthSession
- **cc** TPM_CC command code

See:

- wolfTPM2_PolicyPassword
- wolfTPM2_PolicyAuthValue

Return:

- TPM_RC_SUCCESS: successful
- BAD_FUNC_ARG: check the provided arguments

29.9.6.261 function wolfTPM2_PolicyOR

```
WOLFTPM_API int wolfTPM2_PolicyOR(
    WOLFTPM2_DEV * dev,
    WOLFTPM2_SESSION * tpmSession,
```

```
    const TPML_DIGEST * pHashList
)
```

Wrapper for satisfying a policy session with a compound OR of digests.

Parameters:

- **dev** pointer to a TPM2_DEV struct
- **tpmSession** pointer to a WOLFTPM2_SESSION struct used with wolfTPM2_StartSession and wolfTPM2_SetAuthSession
- **pHashList** list of pre-computed policy branch digests to OR together

See:

- wolfTPM2_PolicyPCR
- wolfTPM2_PolicyAuthorize
- wolfTPM2_GetPolicyDigest

Return:

- TPM_RC_SUCCESS: successful
- BAD_FUNC_ARG: bad pointer, count out of range (< 2 or > capacity), or a branch size that exceeds the digest buffer

Note:

- The digest list is hash-agnostic (each branch carries its own size), so it supports SHA2-256 through SHA2-512 policy branches. The number of branches (pHashList->count) must be between 2 and the TPML_DIGEST capacity (the digests[] array length), and each branch's size must not exceed the digest buffer length; branches beyond count are ignored.
- A minimum of two branches is required by TPM 2.0 Part 3 Sec.23.6. A one-branch list is rejected here with BAD_FUNC_ARG rather than sent to the TPM, which would answer TPM_RC_VALUE.

29.9.6.262 function wolfTPM2_SetPrimaryPolicy

```
WOLFTPM_API int wolfTPM2_SetPrimaryPolicy(
    WOLFTPM2_DEV * dev,
    TPMI_RH_HIERARCHY_AUTH authHandle,
    TPM_ALG_ID hashAlg,
    const byte * authPolicy,
    word32 authPolicySz
)
```

Set (or clear) the authPolicy of a hierarchy.

Parameters:

- **dev** pointer to a TPM2_DEV struct
- **authHandle** the hierarchy (for example TPM_RH_PLATFORM)
- **hashAlg** the policy digest hash algorithm (TPM_ALG_NULL to clear)
- **authPolicy** the policy digest to set (NULL to clear)
- **authPolicySz** size of the policy digest (0 to clear)

See:

- wolfTPM2_PolicyOR
- wolfTPM2_GetPolicyDigest

Return:

- TPM_RC_SUCCESS: successful

- BAD_FUNC_ARG: NULL dev, authPolicySz exceeds the digest buffer, or authPolicy is NULL with a non-zero authPolicySz

Note: Wraps TPM2_SetPrimaryPolicy for owner/endorsement/platform/lockout. Pass authPolicy=NULL, authPolicySz=0 and hashAlg=TPM_ALG_NULL to clear an existing policy. The command itself is authorized by the hierarchy's current auth (set it on the device's active session beforehand).

29.9.6.263 function wolfTPM2_SetIdentityAuth

```
WOLFTPM_API int wolfTPM2_SetIdentityAuth(
    WOLFTPM2_DEV * dev,
    WOLFTPM2_HANDLE * handle,
    uint8_t * masterPassword,
    uint16_t masterPasswordSz
)
```

Set authentication for pre-provisioned identity keys.

Parameters:

- **dev** pointer to a TPM2_DEV struct
- **handle** pointer to WOLFTPM2_HANDLE for the identity key
- **masterPassword** pointer to master password data (NULL uses the ST33 sample password only on ST33-capable builds)
- **masterPasswordSz** size of master password in bytes

See: wolfTPM2_CreateAndLoadAIK

Return:

- TPM_RC_SUCCESS: successful
- TPM_RC_FAILURE: generic failure (check TPM IO and TPM return code)
- BAD_FUNC_ARG: NULL handle, or NULL/empty masterPassword on a non-ST33 build

Note:

- Used with IAK and IDevID keys on ST33KTPM devices
- On ST33 (or autodetect) builds a NULL masterPassword derives auth from the public sample password; other targets require an explicit masterPassword and return BAD_FUNC_ARG when it is NULL/empty

29.9.6.264 function GetKeyTemplateRSA

```
WOLFTPM_LOCAL int GetKeyTemplateRSA(
    TPMT_PUBLIC * publicTemplate,
    TPM_ALG_ID nameAlg,
    TPMA_OBJECT objectAttributes,
    int keyBits,
    long exponent,
    TPM_ALG_ID sigScheme,
    TPM_ALG_ID sigHash
)
```

Internal helper to create RSA key template.

Parameters:

- **publicTemplate** pointer to TPMT_PUBLIC template to populate
- **nameAlg** hash algorithm for key name
- **objectAttributes** TPM object attributes

- **keyBits** RSA key size in bits
- **exponent** RSA public exponent
- **sigScheme** signature scheme algorithm
- **sigHash** hash algorithm for signatures

See: GetKeyTemplateECC

Return:

- TPM_RC_SUCCESS: successful
- BAD_FUNC_ARG: check the provided arguments

Note: Used internally by key creation functions

29.9.6.265 function GetKeyTemplateECC

```
WOLFTPM_LOCAL int GetKeyTemplateECC(
    TPMT_PUBLIC * publicTemplate,
    TPM_ALG_ID nameAlg,
    TPMA_OBJECT objectAttributes,
    TPM_ECC_CURVE curve,
    TPM_ALG_ID sigScheme,
    TPM_ALG_ID sigHash
)
```

Internal helper to create ECC key template.

Parameters:

- **publicTemplate** pointer to TPMT_PUBLIC template to populate
- **nameAlg** hash algorithm for key name
- **objectAttributes** TPM object attributes
- **curve** ECC curve identifier
- **sigScheme** signature scheme algorithm
- **sigHash** hash algorithm for signatures

See: GetKeyTemplateRSA

Return:

- TPM_RC_SUCCESS: successful
- BAD_FUNC_ARG: check the provided arguments

Note: Used internally by key creation functions

29.9.6.266 function wolfTPM2_EccZToBuffer

```
WOLFTPM_TEST_API int wolfTPM2_EccZToBuffer(
    byte * out,
    int * outSz,
    const TPM2B_ECC_PARAMETER * z
)
```

29.9.6.267 function wolfTPM2_FirmwareUpgradeHash

```
WOLFTPM_API int wolfTPM2_FirmwareUpgradeHash(
    WOLFTPM2_DEV * dev,
    TPM_ALG_ID hashAlg,
    uint8_t * manifest_hash,
    uint32_t manifest_hash_sz,
```

```
    uint8_t * manifest,
    uint32_t manifest_sz,
    wolfTPM2FwDataCb cb,
    void * cb_ctx
)
```

Perform TPM firmware upgrade with pre-computed manifest hash.

Parameters:

- **dev** pointer to a TPM2_DEV struct
- **hashAlg** hash algorithm to use (TPM_ALG_SHA384 or TPM_ALG_SHA512)
- **manifest_hash** buffer to store computed manifest hash
- **manifest_hash_sz** size of manifest hash buffer
- **manifest** pointer to firmware manifest data
- **manifest_sz** size of firmware manifest
- **cb** callback function for firmware data access
- **cb_ctx** context pointer passed to callback

See:

- wolfTPM2_FirmwareUpgrade
- wolfTPM2_FirmwareUpgradeRecover

Return:

- TPM_RC_SUCCESS: successful
- TPM_RC_FAILURE: generic failure (check TPM IO and TPM return code)
- BAD_FUNC_ARG: check the provided arguments

Note:

- Supports SHA2-384 or SHA2-512 for manifest hash
- For ST33KTPM: LMS vs non-LMS format is auto-detected from manifest size:
- 177 bytes: Non-LMS format (firmware < 512, e.g., 9.257)
- 2697 bytes: LMS format (firmware >= 512, e.g., 9.512)

29.9.6.268 function wolfTPM2_FirmwareUpgradeHash_ex

```
WOLFTPM_API int wolfTPM2_FirmwareUpgradeHash_ex(
    WOLFTPM2_DEV * dev,
    TPM_ALG_ID hashAlg,
    uint8_t * manifest_hash,
    uint32_t manifest_hash_sz,
    uint8_t * manifest,
    uint32_t manifest_sz,
    wolfTPM2FwDataCb cb,
    void * cb_ctx,
    WOLFTPM2_SESSION * startSession
)
```

Perform TPM firmware upgrade using a caller-supplied authorization session.

Parameters:

- **dev** pointer to a TPM2_DEV struct
- **hashAlg** hash algorithm to use (TPM_ALG_SHA384 or TPM_ALG_SHA512)
- **manifest_hash** buffer to store computed manifest hash
- **manifest_hash_sz** size of manifest hash buffer

- **manifest** pointer to firmware manifest data
- **manifest_sz** size of firmware manifest
- **cb** callback function for firmware data access
- **cb_ctx** context pointer passed to callback
- **startSession** optional caller-satisfied session authorizing the firmware-start command (NULL for library-managed authorization)

See:

- wolfTPM2_FirmwareUpgradeHash
- wolfTPM2_PolicyOR
- wolfTPM2_StartSession_ex

Return:

- TPM_RC_SUCCESS: successful
- TPM_RC_FAILURE: generic failure (check TPM IO and TPM return code)
- BAD_FUNC_ARG: check the provided arguments, or startSession is not an unsalted/unbound policy session with no auth value

Note:

- Identical to wolfTPM2_FirmwareUpgradeHash except the caller controls how the firmware-start command is authorized against the platform hierarchy.
- When startSession is NULL this behaves exactly like wolfTPM2_FirmwareUpgradeHash (library-managed platform authorization).
- When startSession is non-NULL the caller is responsible for having satisfied the platform authPolicy on that session (for example via wolfTPM2_PolicyPCR / wolfTPM2_PolicyAuthorize / wolfTPM2_PolicyOR using SHA2-256 or SHA2-512). For Infineon the platform primary policy is left untouched (the caller provisions it); for ST33 the session replaces the default TPM_RS_PW password authorization.
- The vendor firmware-start command carries a session handle with an empty nonceCaller, zero session attributes and an empty HMAC, so startSession must be an unsalted, unbound TPM_SE_POLICY session with no auth value and no parameter encryption. Sessions satisfied with wolfTPM2_PolicyAuthValue or wolfTPM2_PolicyPassword are NOT supported, because the required session HMAC is not serialized. Any such session is rejected with BAD_FUNC_ARG before the command is sent.
- On a successful firmware start the TPM consumes the session, and this function sets startSession->handle.hndl to TPM_RH_NULL to record that. The caller must not flush it; wolfTPM2_UnloadHandle is a no-op on TPM_RH_NULL, so an unconditional cleanup call remains safe.

29.9.6.269 function wolfTPM2_FirmwareUpgrade

```
WOLFTPM_API int wolfTPM2_FirmwareUpgrade(
    WOLFTPM2_DEV * dev,
    uint8_t * manifest,
    uint32_t manifest_sz,
    wolfTPM2FwDataCb cb,
    void * cb_ctx
)
```

Perform TPM firmware upgrade.

Parameters:

- **dev** pointer to a TPM2_DEV struct
- **manifest** pointer to firmware manifest data
- **manifest_sz** size of firmware manifest

- **cb** callback function for firmware data access
- **cb_ctx** context pointer passed to callback

See:

- wolfTPM2_FirmwareUpgradeHash
- wolfTPM2_FirmwareUpgradeRecover

Return:

- TPM_RC_SUCCESS: successful
- TPM_RC_FAILURE: generic failure (check TPM IO and TPM return code)
- BAD_FUNC_ARG: check the provided arguments
- NOT_COMPILED_IN: wolfSSL not built with WOLFSSL_SHA384

Note:

- Upgrades TPM firmware using provided manifest and data callback
- Requires WOLFSSL_SHA384; returns NOT_COMPILED_IN otherwise
- For ST33KTPM: LMS vs non-LMS format is auto-detected from manifest size:
- 177 bytes: Non-LMS format (firmware < 512, e.g., 9.257)
- 2697 bytes: LMS format (firmware >= 512, e.g., 9.512)

29.9.6.270 function wolfTPM2_FirmwareUpgrade_ex

```
WOLFTPM_API int wolfTPM2_FirmwareUpgrade_ex(
    WOLFTPM2_DEV * dev,
    uint8_t * manifest,
    uint32_t manifest_sz,
    wolfTPM2FwDataCb cb,
    void * cb_ctx,
    WOLFTPM2_SESSION * startSession
)
```

Perform TPM firmware upgrade using a caller-supplied authorization session.

Parameters:

- **dev** pointer to a TPM2_DEV struct
- **manifest** pointer to firmware manifest data
- **manifest_sz** size of firmware manifest
- **cb** callback function for firmware data access
- **cb_ctx** context pointer passed to callback
- **startSession** optional caller-satisfied session (NULL for default). See wolfTPM2_FirmwareUpgradeHash_ex for the supported session contract: unsalted, unbound policy session with no auth value.

See:

- wolfTPM2_FirmwareUpgrade
- wolfTPM2_FirmwareUpgradeHash_ex

Return:

- TPM_RC_SUCCESS: successful
- NOT_COMPILED_IN: wolfSSL not built with WOLFSSL_SHA384

Note: Same as wolfTPM2_FirmwareUpgrade but the caller controls how the firmware-start command is authorized (see wolfTPM2_FirmwareUpgradeHash_ex). startSession NULL preserves the default library-managed behavior.

29.9.6.271 function wolfTPM2_FirmwareUpgradeRecover

```
WOLFTPM_API int wolfTPM2_FirmwareUpgradeRecover(
    WOLFTPM2_DEV * dev,
    uint8_t * manifest,
    uint32_t manifest_sz,
    wolfTPM2FwDataCb cb,
    void * cb_ctx
)
```

Recover from failed TPM firmware upgrade.

Parameters:

- **dev** pointer to a TPM2_DEV struct
- **manifest** pointer to firmware manifest data
- **manifest_sz** size of firmware manifest
- **cb** callback function for firmware data access
- **cb_ctx** context pointer passed to callback

See:

- wolfTPM2_FirmwareUpgrade
- wolfTPM2_FirmwareUpgradeHash

Return:

- TPM_RC_SUCCESS: successful
- TPM_RC_FAILURE: generic failure (check TPM IO and TPM return code)
- BAD_FUNC_ARG: check the provided arguments

Note: Attempts to recover TPM after interrupted/failed upgrade

29.9.6.272 function wolfTPM2_FirmwareUpgradeRecover_ex

```
WOLFTPM_API int wolfTPM2_FirmwareUpgradeRecover_ex(
    WOLFTPM2_DEV * dev,
    uint8_t * manifest,
    uint32_t manifest_sz,
    wolfTPM2FwDataCb cb,
    void * cb_ctx,
    WOLFTPM2_SESSION * startSession
)
```

Recover from a failed firmware upgrade using a caller-supplied session.

Parameters:

- **dev** pointer to a TPM2_DEV struct
- **manifest** pointer to firmware manifest data
- **manifest_sz** size of firmware manifest
- **cb** callback function for firmware data access
- **cb_ctx** context pointer passed to callback
- **startSession** optional caller-satisfied session (NULL for default). See wolfTPM2_FirmwareUpgradeHash_ex for the supported session contract: unsalted, unbound policy session with no auth value.

See:

- wolfTPM2_FirmwareUpgradeRecover
- wolfTPM2_FirmwareUpgradeHash_ex

Return:

- TPM_RC_SUCCESS: successful
- BAD_FUNC_ARG: check the provided arguments

Note: Same as wolfTPM2_FirmwareUpgradeRecover but with caller-controlled authorization (see wolfTPM2_FirmwareUpgradeHash_ex). startSession NULL preserves the default library-managed behavior.

29.9.6.273 function wolfTPM2_FirmwareUpgradeCancel

```
WOLFTPM_API int wolfTPM2_FirmwareUpgradeCancel(
    WOLFTPM2_DEV * dev
)
```

Cancel ongoing TPM firmware upgrade.

Parameters:

- **dev** pointer to a TPM2_DEV struct

See:

- wolfTPM2_FirmwareUpgrade
- wolfTPM2_FirmwareUpgradeRecover

Return:

- TPM_RC_SUCCESS: successful
- TPM_RC_FAILURE: generic failure (check TPM IO and TPM return code)
- BAD_FUNC_ARG: check the provided arguments

Note: Aborts current firmware upgrade process

29.9.6.274 function wolfTPM2_ST33_ManifestVersion

```
WOLFTPM_API int wolfTPM2_ST33_ManifestVersion(
    const uint8_t * manifest,
    uint32_t manifest_sz,
    word16 * major,
    word16 * minor
)
```

Read the firmware version an ST33 firmware image upgrades to.

Parameters:

- **manifest** pointer to the manifest (blob0) bytes
- **manifest_sz** size of the manifest in bytes
- **major** pointer to store the target major version (optional, may be NULL)
- **minor** pointer to store the target minor version (optional, may be NULL)

See:

- wolfTPM2_ST33_FwUpgradeCommands
- wolfTPM2_FirmwareUpgrade

Return:

- TPM_RC_SUCCESS: successful
- BAD_FUNC_ARG: manifest is NULL or too small to hold the version

Note: The manifest (blob0) header opens with a zero byte followed by the target version in the TPM_PT_FIRMWARE_VERSION_1 layout: UINT16 major then UINT16 minor, big endian (00 | 00 02 02 00 is 2.512)

29.9.6.275 function wolfTPM2_ST33_CmdImplemented

```
WOLFTPM_API int wolfTPM2_ST33_CmdImplemented(
    TPM_CC cc,
    int * isImpl
)
```

Report whether the TPM implements a command code.

Parameters:

- **cc** command code to look for
- **isImpl** pointer to store 1 when implemented, 0 when not

See: wolfTPM2_ST33_GetFwUpgradeCommands

Return:

- TPM_RC_SUCCESS: query succeeded, isImpl set to 1 or 0
- BAD_FUNC_ARG: isImpl is NULL
- TPM_RC_VALUE: the TPM answered with a different capability

Note: Queries TPM_CAP_COMMANDS, which returns commands at or above the requested code, so a match at index 0 of a single-property query means it is implemented. Used to tell which field upgrade command pair an ST33 part supports

29.9.6.276 function wolfTPM2_ST33_GetFwUpgradeCommands

```
WOLFTPM_API int wolfTPM2_ST33_GetFwUpgradeCommands(
    const WOLFTPM2_CAPS * caps,
    word16 manifestMajor,
    TPM_CC * ccStart,
    TPM_CC * ccData,
    int * fromTpm
)
```

Choose the ST33 field upgrade command codes for the attached TPM.

Parameters:

- **caps** capabilities of the attached TPM, or NULL once it has entered firmware upgrade mode, where no capability query may be issued because the TPM accepts only FieldUpgradeData and any other command leaves the mode. With NULL the firmware version rule decides alone
- **manifestMajor** target major version from wolfTPM2_ST33_ManifestVersion
- **ccStart** pointer to store the field upgrade start command code
- **ccData** pointer to store the field upgrade data command code
- **fromTpm** pointer to store 1 when the TPM decided and 0 when the version rule did (optional, may be NULL)

See:

- wolfTPM2_ST33_CmdImplemented
- wolfTPM2_ST33_ManifestVersion

Return:

- TPM_RC_SUCCESS: successful
- BAD_FUNC_ARG: ccStart or ccData is NULL

Note: Asks the TPM which pair it implements, since the firmware version rule does not hold on every ST33 firmware line, and falls back to wolfTPM2_ST33_FwUpgradeCommands when the TPM does not

settle it. This is what the library sends, so a caller binding a PolicyCommandCode for the upgrade should bind the start code this returns

29.9.6.277 function wolfTPM2_ST33_FwUpgradeCommands

```
WOLFTPM_TEST_API int wolfTPM2_ST33_FwUpgradeCommands(
    word16 fwVerMinor,
    int haveFwVer,
    word16 manifestMajor,
    TPM_CC * ccStart,
    TPM_CC * ccData
)
```

Choose the ST33 field upgrade command codes from the version fields.

Parameters:

- **fwVerMinor** running firmware minor version from WOLFTPM2_CAPS
- **haveFwVer** set to 1 when fwVerMinor is known, 0 when it is not
- **manifestMajor** target major version from wolfTPM2_ST33_ManifestVersion
- **ccStart** pointer to store the field upgrade start command code
- **ccData** pointer to store the field upgrade data command code

See:

- wolfTPM2_ST33_ManifestVersion
- wolfTPM2_FirmwareUpgrade

Return:

- TPM_RC_SUCCESS: successful
- BAD_FUNC_ARG: ccStart or ccData is NULL

Note: ST33 implements the field upgrade with either the standard TPM_CC_FieldUpgradeStart/Data or the ST33KTPM vendor codes, and the wrong pair is answered with TPM_RC_COMMAND_CODE. This applies the rule ST's reference tool uses. Callers want wolfTPM2_ST33_GetFwUpgradeCommands instead, which asks the TPM first and only falls back to this. Pass haveFwVer = 0 when the running version is unknown, as it is once the TPM has entered upgrade mode

29.9.7 Attributes Documentation

29.9.7.1 variable C

```
C {
#endif


typedef struct WOLFTPM2_HANDLE {
    TPM_HANDLE      hndl;
    TPM2B_AUTH      auth;
    TPMT_SYM_DEF    symmetric;
    TPM2B_NAME      name;


    unsigned int    policyPass : 1;
    unsigned int    policyAuth : 1;
    unsigned int    nameLoaded : 1;
} WOLFTPM2_HANDLE;
```

29.9.8 Source code

```
/* tpm2_wrap.h
 *
 * Copyright (C) 2006-2026 wolfSSL Inc.
 *
 * This file is part of wolfTPM.
 *
 * wolfTPM is free software; you can redistribute it and/or modify
 * it under the terms of the GNU General Public License as published by
 * the Free Software Foundation; either version 3 of the License, or
 * (at your option) any later version.
 *
 * wolfTPM is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
 * GNU General Public License for more details.
 *
 * You should have received a copy of the GNU General Public License
 * along with this program; if not, write to the Free Software
 * Foundation, Inc., 51 Franklin Street, Fifth Floor, Boston, MA 02110-1335, USA
 */

#ifndef __TPM2_WRAP_H__
#define __TPM2_WRAP_H__

#include <wolftpm/tpm2.h>
#ifdef WOLFTPM_SPDM
#include <wolftpm/tpm2_spdm.h>
#endif

#ifdef __cplusplus
    extern "C" {
#endif


typedef struct WOLFTPM2_HANDLE {
    TPM_HANDLE      hndl;
    TPM2B_AUTH      auth;
    TPMT_SYM_DEF    symmetric;
    TPM2B_NAME      name;

    /* bit-fields */
    unsigned int    policyPass : 1;
    unsigned int    policyAuth : 1; /* Handle requires policy auth */
    unsigned int    nameLoaded : 1; /* flag to indicate if "name" was loaded
     ↪  and computed */
} WOLFTPM2_HANDLE;

#define TPM_SES_PWD 0xFF /* Session type for Password that fits in one byte */

typedef struct WOLFTPM2_SESSION {
    TPM_ST          type;         /* Trial, Policy or HMAC; or TPM_SES_PWD */
    WOLFTPM2_HANDLE handle;       /* Session handle from StartAuthSession */
```

```
    TPM2B_NONCE     nonceTPM;     /* Value from StartAuthSession */
    TPM2B_NONCE     nonceCaller;  /* Fresh nonce at each command */
    TPM2B_DIGEST    salt;         /* User defined */
    TPMI_ALG_HASH   authHash;
    TPMA_SESSION    sessionAttributes;
    TPM2B_AUTH*     bind;         /* pointer to bind auth password */
} WOLFTPM2_SESSION;

typedef struct WOLFTPM2_DEV {
    TPM2_CTX ctx;
    TPM2_AUTH_SESSION session[MAX_SESSION_NUM];
#ifdef WOLFTPM_SPDM
    WOLFTPM2_SPDM_CTX  spdmCtxData;
    WOLFTPM2_SPDM_CTX* spdmCtx; /* NULL = not initialized */
#endif
} WOLFTPM2_DEV;

/* Public Key with Handle.
 *   Must have "handle" and "pub" as first members */
typedef struct WOLFTPM2_KEY {
    WOLFTPM2_HANDLE   handle;
    TPM2B_PUBLIC      pub;
} WOLFTPM2_KEY;

/* Primary Key - From TPM2_CreatePrimary that include creation hash and ticket.
 * WOLFTPM2_PKEY can be cast to WOLFTPM2_KEY.
 *   Must have "handle" and "pub" as first members */
typedef struct WOLFTPM2_PKEY {
    WOLFTPM2_HANDLE   handle;
    TPM2B_PUBLIC      pub;

    TPM2B_DIGEST      creationHash;
    TPMT_TK_CREATION  creationTicket;
} WOLFTPM2_PKEY;

/* Private/Public Key:
 * WOLFTPM2_KEYBLOB can be cast to WOLFTPM2_KEY
 * Must have "handle" and "pub" as first members */
typedef struct WOLFTPM2_KEYBLOB {
    WOLFTPM2_HANDLE   handle;
    TPM2B_PUBLIC      pub;
    TPM2B_PRIVATE     priv;
    /* Note: Member "name" moved to "handle.name" */
} WOLFTPM2_KEYBLOB;

typedef struct WOLFTPM2_HASH {
    WOLFTPM2_HANDLE handle;
} WOLFTPM2_HASH;

typedef struct WOLFTPM2_NV {
    WOLFTPM2_HANDLE handle;
    TPMA_NV attributes;
} WOLFTPM2_NV;
```

```
typedef struct WOLFTPM2_HMAC {
    WOLFTPM2_HASH   hash;
    WOLFTPM2_KEY    key;

    /* option bits */
    word16 hmacKeyLoaded:1;
    word16 hmacKeyKeep:1;
} WOLFTPM2_HMAC;

#ifdef WOLFTPM2_CERT_GEN
typedef struct WOLFTPM2_CSR {
    Cert req;
} WOLFTPM2_CSR;

#ifdef WOLFSSL_CERT_SIGN_CB
typedef struct TpmSignCbCtx {
    WOLFTPM2_DEV* dev;
    WOLFTPM2_KEY* key;
} TpmSignCbCtx;
#endif /* WOLFSSL_CERT_SIGN_CB */
#endif /* WOLFTPM2_CERT_GEN */

/* buffer similar to TPM2B_MAX_BUFFER that can be used */
typedef struct WOLFTPM2_BUFFER {
    int size;
    byte buffer[MAX_DIGEST_BUFFER];
} WOLFTPM2_BUFFER;

typedef enum WOLFTPM2_MFG {
    TPM_MFG_UNKNOWN = 0,
    TPM_MFG_INFINEON,
    TPM_MFG_STM,
    TPM_MFG_MCHP,
    TPM_MFG_NUVOTON,
    TPM_MFG_NATIONTECH,
    TPM_MFG_SEALSQ,
    TPM_MFG_MSFT, /* Microsoft reference firmware TPM (e.g. OP-TEE fTPM TA) */
} WOLFTPM2_MFG;

typedef struct WOLFTPM2_CAPS {
    WOLFTPM2_MFG mfg;
    char mfgStr[4 + 1];
    char vendorStr[(4 * 4) + 1];
    word32 tpmType;
    word16 fwVerMajor;
    word16 fwVerMinor;
    word32 fwVerVendor;
#ifdef WOLFTPM_IFX_FW_UPGRADE
    word32 keyGroupId;
    word16 fwCounter;
    word16 fwCounterSame;
    byte   opMode;
#endif
```

```
    /* bits */
    word16 fips140_2 : 1; /* FIPS 140-2 mode (TPMA_MODES bit 0) */
    word16 cc_eal4   : 1; /* Common Criteria EAL4+ */
    word16 req_wait_state : 1; /* requires SPI wait state */
    word16 fips140_3 : 1; /* FIPS 140-3 mode (TPMA_MODES bit 1, spec v1.83+) */
#ifdef WOLFTPM_SPDM
    word32 acHandleCount;  /* Number of AC handles discovered */
    TPM_HANDLE acHandles[MAX_AC_HANDLES];  /* AC handles */
#endif
} WOLFTPM2_CAPS;


/* Wrapper API's to simplify TPM use */

/* For devtpm and swtpm builds, the ioCb and userCtx are not used and should be
 ↪  set to NULL */

WOLFTPM_API int wolfTPM2_Test(TPM2HalIoCb ioCb, void* userCtx, WOLFTPM2_CAPS*
 ↪  caps);

WOLFTPM_API int wolfTPM2_Init(WOLFTPM2_DEV* dev, TPM2HalIoCb ioCb, void*
 ↪  userCtx);

#if defined(WOLFTPM_SPDM) && defined(WOLFTPM_SPDM_TCG)
WOLFTPM_API int wolfTPM2_InitWithSpdmKey(WOLFTPM2_DEV* dev,
    TPM2HalIoCb ioCb, void* userCtx, const byte* rspPubKey,
    word32 rspPubKeySz);

WOLFTPM_API int wolfTPM2_InitWithSpdmKey_ex(WOLFTPM2_DEV* dev,
    TPM2HalIoCb ioCb, void* userCtx, const byte* rspPubKey,
    word32 rspPubKeySz, WOLFSPDM_MODE mode);

/* Test-visible helper used by automatic multi-vendor selection. */
WOLFTPM_TEST_API int wolfTPM2_SpdmModeFromDidVid(UINT32 didVid,
    WOLFSPDM_MODE* mode);
#endif

#if defined(WOLFTPM_SPDM) && defined(WOLFTPM_SPDM_PSK)
WOLFTPM_API int wolfTPM2_InitWithSpdmPsk(WOLFTPM2_DEV* dev,
    TPM2HalIoCb ioCb, void* userCtx, const byte* psk, word32 pskSz,
    const byte* hint, word32 hintSz);
#endif

WOLFTPM_API int wolfTPM2_OpenExisting(WOLFTPM2_DEV* dev, TPM2HalIoCb ioCb,
 ↪  void* userCtx);

WOLFTPM_API int wolfTPM2_Cleanup(WOLFTPM2_DEV* dev);

WOLFTPM_API int wolfTPM2_Cleanup_ex(WOLFTPM2_DEV* dev, int doShutdown);

WOLFTPM_API int wolfTPM2_GetTpmDevId(WOLFTPM2_DEV* dev);

WOLFTPM_API int wolfTPM2_SelfTest(WOLFTPM2_DEV* dev);
```

```
WOLFTPM_API int wolfTPM2_GetCapabilities(WOLFTPM2_DEV* dev, WOLFTPM2_CAPS*
 ↪  caps);

WOLFTPM_TEST_API int wolfTPM2_ParseCapabilities(WOLFTPM2_CAPS* caps,
    TPML_TAGGED_TPM_PROPERTY* props);

WOLFTPM_API int wolfTPM2_IsAlgSupported(WOLFTPM2_DEV* dev, TPM_ALG_ID alg,
    int* isSupported);

WOLFTPM_API int wolfTPM2_GetHandles(TPM_HANDLE handle, TPML_HANDLE* handles);

#ifdef WOLFTPM_SPDM
/* SPDM Secure Session Wrapper API
 *
 * These functions provide a high-level interface for SPDM secure sessions.
 * All SPDM protocol logic is implemented in the wolfSPDM library.
 */

WOLFTPM_API int wolfTPM2_SpdmInit(WOLFTPM2_DEV* dev);

WOLFTPM_API int wolfTPM2_SpdmConnect(WOLFTPM2_DEV* dev);

WOLFTPM_API int wolfTPM2_SpdmIsConnected(WOLFTPM2_DEV* dev);

WOLFTPM_API word32 wolfTPM2_SpdmGetSessionId(WOLFTPM2_DEV* dev);

WOLFTPM_API int wolfTPM2_GetCapability_SPDMSessionInfo(WOLFTPM2_DEV* dev,
    TPML_SPDM_SESSION_INFO* spdmSessionInfo);

WOLFTPM_API int wolfTPM2_SpdmDisconnect(WOLFTPM2_DEV* dev);

WOLFTPM_API int wolfTPM2_SpdmGetVersion(WOLFTPM2_DEV* dev);

WOLFTPM_API int wolfTPM2_SpdmCleanup(WOLFTPM2_DEV* dev);

#ifdef WOLFTPM_SPDM_TCG
WOLFTPM_API int wolfTPM2_SpdmSetResponderPubKey(WOLFTPM2_DEV* dev,
    const byte* pubKey, word32 pubKeySz);

WOLFTPM_API int wolfTPM2_SpdmGetPubKey(WOLFTPM2_DEV* dev,
    byte* pubKey, word32* pubKeySz);
#endif /* WOLFTPM_SPDM_TCG */

#ifdef WOLFSPDM_NUVOTON
/* Nuvoton-specific SPDM functions (requires wolfSPDM with --enable-nuvoton) */

WOLFTPM_API int wolfTPM2_SpdmSetNuvotonMode(WOLFTPM2_DEV* dev);

WOLFTPM_API int wolfTPM2_SpdmEnable(WOLFTPM2_DEV* dev);

WOLFTPM_API int wolfTPM2_SpdmDisable(WOLFTPM2_DEV* dev);

WOLFTPM_API int wolfTPM2_SpdmConnectNuvoton(WOLFTPM2_DEV* dev,
    const byte* reqPubKey, word32 reqPubKeySz,
```

```
    const byte* reqPrivKey, word32 reqPrivKeySz);

WOLFTPM_API int wolfTPM2_SpdmGetStatus(WOLFTPM2_DEV* dev,
    WOLFSPDM_NUVOTON_STATUS* status);

WOLFTPM_API int wolfTPM2_SpdmSetOnlyMode(WOLFTPM2_DEV* dev, int lock);

#endif /* WOLFSPDM_NUVOTON */

#ifdef WOLFSPDM_NATIONS
/* Nations Technology NS350 SPDM functions (requires --enable-nations
 ↪  --enable-spdm) */

WOLFTPM_API int wolfTPM2_SpdmSetNationsMode(WOLFTPM2_DEV* dev);

WOLFTPM_API int wolfTPM2_SpdmConnectNations(WOLFTPM2_DEV* dev,
    const byte* reqPubKey, word32 reqPubKeySz,
    const byte* reqPrivKey, word32 reqPrivKeySz);

WOLFTPM_API int wolfTPM2_SpdmNationsIdentityKeySet(WOLFTPM2_DEV* dev, int set);

/* wolfTPM2_SpdmConnectNationsPsk is now an alias for the spec-flag-gated
 * wolfTPM2_SpdmConnectPsk (declared below). Kept for source compatibility. */
#define wolfTPM2_SpdmConnectNationsPsk wolfTPM2_SpdmConnectPsk

WOLFTPM_API int wolfTPM2_SpdmNationsGetStatus(WOLFTPM2_DEV* dev,
    WOLFSPDM_NATIONS_STATUS* status);

WOLFTPM_API int wolfTPM2_SpdmNationsSetOnlyMode(WOLFTPM2_DEV* dev, int lock);

WOLFTPM_API int wolfTPM2_SpdmNationsPskSet(WOLFTPM2_DEV* dev,
    const byte* psk, word32 pskSz);

WOLFTPM_API int wolfTPM2_SpdmNationsPskClear(WOLFTPM2_DEV* dev,
    const byte* clearAuth, word32 clearAuthSz);

#endif /* WOLFSPDM_NATIONS */

#ifdef WOLFSPDM_INFINEON
WOLFTPM_API int wolfTPM2_SpdmSetInfineonMode(WOLFTPM2_DEV* dev);

WOLFTPM_API int wolfTPM2_SpdmConnectInfineonTcg(WOLFTPM2_DEV* dev,
    const byte* reqPubKey, word32 reqPubKeySz,
    const byte* reqPrivKey, word32 reqPrivKeySz);

/* wolfTPM2_SpdmConnectInfineon establishes the SLB9678 SPDM secure session
 * via the DSP0274 PSK handshake. This compatibility name aliases the
 * spec-pure wolfTPM2_SpdmConnectPsk (declared below). */
#define wolfTPM2_SpdmConnectInfineon wolfTPM2_SpdmConnectPsk

WOLFTPM_API int wolfTPM2_SpdmInfineonGetStatus(WOLFTPM2_DEV* dev,
    WOLFSPDM_INFINEON_STATUS* status);

WOLFTPM_API int wolfTPM2_SpdmInfineonSetOnlyMode(WOLFTPM2_DEV* dev, int lock);
```

```
WOLFTPM_API int wolfTPM2_SpdmInfineonEnable(WOLFTPM2_DEV* dev,
    int enable, int lock);

#endif /* WOLFSPDM_INFINEON */

#ifdef WOLFTPM_SPDM_PSK
WOLFTPM_API int wolfTPM2_SpdmConnectPsk(WOLFTPM2_DEV* dev,
    const byte* psk, word32 pskSz,
    const byte* hint, word32 hintSz);
#endif /* WOLFTPM_SPDM_PSK */

#endif /* WOLFTPM_SPDM */

WOLFTPM_API int wolfTPM2_UnsetAuth(WOLFTPM2_DEV* dev, int index);

WOLFTPM_API int wolfTPM2_UnsetAuthSession(WOLFTPM2_DEV* dev, int index,
 ↪  WOLFTPM2_SESSION* session);

WOLFTPM_API int wolfTPM2_SetAuth(WOLFTPM2_DEV* dev, int index,
    TPM_HANDLE sessionHandle, const TPM2B_AUTH* auth, TPMA_SESSION
 ↪  sessionAttributes,
    const TPM2B_NAME* name);

WOLFTPM_API int wolfTPM2_SetAuthPassword(WOLFTPM2_DEV* dev, int index, const
 ↪  TPM2B_AUTH* auth);

WOLFTPM_API int wolfTPM2_SetAuthHandle(WOLFTPM2_DEV* dev, int index, const
 ↪  WOLFTPM2_HANDLE* handle);

WOLFTPM_API int wolfTPM2_SetAuthSession(WOLFTPM2_DEV* dev, int index,
    WOLFTPM2_SESSION* tpmSession, TPMA_SESSION sessionAttributes);

WOLFTPM_API int wolfTPM2_SetSessionHandle(WOLFTPM2_DEV* dev, int index,
    WOLFTPM2_SESSION* tpmSession);

WOLFTPM_API int wolfTPM2_SetAuthHandleName(WOLFTPM2_DEV* dev, int index, const
 ↪  WOLFTPM2_HANDLE* handle);

WOLFTPM_API int wolfTPM2_StartSession(WOLFTPM2_DEV* dev,
    WOLFTPM2_SESSION* session, WOLFTPM2_KEY* tpmKey,
    WOLFTPM2_HANDLE* bind, TPM_SE sesType, int encDecAlg);

WOLFTPM_API int wolfTPM2_StartSession_ex(WOLFTPM2_DEV* dev,
    WOLFTPM2_SESSION* session, WOLFTPM2_KEY* tpmKey,
    WOLFTPM2_HANDLE* bind, TPM_SE sesType, int encDecAlg,
    TPMI_ALG_HASH authHash);

WOLFTPM_API int wolfTPM2_CreateAuthSession_EkPolicy(WOLFTPM2_DEV* dev,
                                                    WOLFTPM2_SESSION* tpmSession);

WOLFTPM_API int wolfTPM2_CreateAuthSession_EkPolicy_ex(WOLFTPM2_DEV* dev,
                                                      WOLFTPM2_SESSION* tpmSession,
                                                        TPMI_ALG_HASH authHash);
```

```
WOLFTPM_API int wolfTPM2_CreatePrimaryKey(WOLFTPM2_DEV* dev,
    WOLFTPM2_KEY* key, TPM_HANDLE primaryHandle, TPMT_PUBLIC* publicTemplate,
    const byte* auth, int authSz);

WOLFTPM_API int wolfTPM2_CreatePrimaryKey_ex(WOLFTPM2_DEV* dev, WOLFTPM2_PKEY*
 ↪  pkey,
    TPM_HANDLE primaryHandle, TPMT_PUBLIC* publicTemplate,
    const byte* auth, int authSz);

WOLFTPM_API int wolfTPM2_ChangeAuthKey(WOLFTPM2_DEV* dev, WOLFTPM2_KEY* key,
    WOLFTPM2_HANDLE* parent, const byte* auth, int authSz);

WOLFTPM_API int wolfTPM2_CreateKey(WOLFTPM2_DEV* dev,
    WOLFTPM2_KEYBLOB* keyBlob, WOLFTPM2_HANDLE* parent,
    TPMT_PUBLIC* publicTemplate, const byte* auth, int authSz);

WOLFTPM_API int wolfTPM2_LoadKey(WOLFTPM2_DEV* dev,
    WOLFTPM2_KEYBLOB* keyBlob, WOLFTPM2_HANDLE* parent);

WOLFTPM_API int wolfTPM2_CreateAndLoadKey(WOLFTPM2_DEV* dev,
    WOLFTPM2_KEY* key, WOLFTPM2_HANDLE* parent, TPMT_PUBLIC* publicTemplate,
    const byte* auth, int authSz);

WOLFTPM_API int wolfTPM2_CreateLoadedKey(WOLFTPM2_DEV* dev, WOLFTPM2_KEYBLOB*
 ↪  keyBlob,
    WOLFTPM2_HANDLE* parent, TPMT_PUBLIC* publicTemplate,
    const byte* auth, int authSz);

WOLFTPM_API int wolfTPM2_LoadPublicKey(WOLFTPM2_DEV* dev, WOLFTPM2_KEY* key,
    const TPM2B_PUBLIC* pub);

/* Same as wolfTPM2_LoadPublicKey, but adds hierarchy option (default is owner)
 ↪  */
WOLFTPM_API int wolfTPM2_LoadPublicKey_ex(WOLFTPM2_DEV* dev, WOLFTPM2_KEY* key,
    const TPM2B_PUBLIC* pub, TPM_HANDLE hierarchy);

WOLFTPM_API int wolfTPM2_LoadPrivateKey(WOLFTPM2_DEV* dev,
    const WOLFTPM2_KEY* parentKey, WOLFTPM2_KEY* key, const TPM2B_PUBLIC* pub,
    TPM2B_SENSITIVE* sens);

WOLFTPM_API int wolfTPM2_ImportPrivateKey(WOLFTPM2_DEV* dev,
    const WOLFTPM2_KEY* parentKey, WOLFTPM2_KEYBLOB* keyBlob, const
     ↪  TPM2B_PUBLIC* pub,
    TPM2B_SENSITIVE* sens);

WOLFTPM_API int wolfTPM2_LoadRsaPublicKey(WOLFTPM2_DEV* dev, WOLFTPM2_KEY* key,
    const byte* rsaPub, word32 rsaPubSz, word32 exponent);

WOLFTPM_API int wolfTPM2_LoadRsaPublicKey_ex(WOLFTPM2_DEV* dev, WOLFTPM2_KEY*
 ↪  key,
    const byte* rsaPub, word32 rsaPubSz, word32 exponent,
    TPMI_ALG_RSA_SCHEME scheme, TPMI_ALG_HASH hashAlg);
```

```
WOLFTPM_API int wolfTPM2_ImportRsaPrivateKey(WOLFTPM2_DEV* dev,
    const WOLFTPM2_KEY* parentKey, WOLFTPM2_KEYBLOB* keyBlob,
    const byte* rsaPub, word32 rsaPubSz, word32 exponent,
    const byte* rsaPriv, word32 rsaPrivSz,
    TPMI_ALG_RSA_SCHEME scheme, TPMI_ALG_HASH hashAlg);

WOLFTPM_API int wolfTPM2_ImportRsaPrivateKeySeed(WOLFTPM2_DEV* dev,
    const WOLFTPM2_KEY* parentKey, WOLFTPM2_KEYBLOB* keyBlob,
    const byte* rsaPub, word32 rsaPubSz, word32 exponent,
    const byte* rsaPriv, word32 rsaPrivSz,
    TPMI_ALG_RSA_SCHEME scheme, TPMI_ALG_HASH hashAlg,
    TPMA_OBJECT attributes, byte* seed, word32 seedSz);

WOLFTPM_API int wolfTPM2_LoadRsaPrivateKey(WOLFTPM2_DEV* dev,
    const WOLFTPM2_KEY* parentKey, WOLFTPM2_KEY* key,
    const byte* rsaPub, word32 rsaPubSz, word32 exponent,
    const byte* rsaPriv, word32 rsaPrivSz);

WOLFTPM_API int wolfTPM2_LoadRsaPrivateKey_ex(WOLFTPM2_DEV* dev,
    const WOLFTPM2_KEY* parentKey, WOLFTPM2_KEY* key,
    const byte* rsaPub, word32 rsaPubSz, word32 exponent,
    const byte* rsaPriv, word32 rsaPrivSz,
    TPMI_ALG_RSA_SCHEME scheme, TPMI_ALG_HASH hashAlg);

WOLFTPM_API int wolfTPM2_LoadEccPublicKey(WOLFTPM2_DEV* dev, WOLFTPM2_KEY* key,
    int curveId, const byte* eccPubX, word32 eccPubXSz,
    const byte* eccPubY, word32 eccPubYSz);

WOLFTPM_API int wolfTPM2_LoadEccPublicKey_ex(WOLFTPM2_DEV* dev,
    WOLFTPM2_KEY* key, int curveId,
    const byte* eccPubX, word32 eccPubXSz,
    const byte* eccPubY, word32 eccPubYSz,
    TPMI_ALG_ECC_SCHEME scheme, TPMI_ALG_HASH hashAlg,
    TPMA_OBJECT objectAttributes);

WOLFTPM_API int wolfTPM2_ImportEccPrivateKey(WOLFTPM2_DEV* dev,
    const WOLFTPM2_KEY* parentKey, WOLFTPM2_KEYBLOB* keyBlob, int curveId,
    const byte* eccPubX, word32 eccPubXSz,
    const byte* eccPubY, word32 eccPubYSz,
    const byte* eccPriv, word32 eccPrivSz);

WOLFTPM_API int wolfTPM2_ImportEccPrivateKeySeed(WOLFTPM2_DEV* dev,
    const WOLFTPM2_KEY* parentKey, WOLFTPM2_KEYBLOB* keyBlob, int curveId,
    const byte* eccPubX, word32 eccPubXSz,
    const byte* eccPubY, word32 eccPubYSz,
    const byte* eccPriv, word32 eccPrivSz,
    TPMA_OBJECT attributes, byte* seed, word32 seedSz);

WOLFTPM_API int wolfTPM2_LoadEccPrivateKey(WOLFTPM2_DEV* dev,
    const WOLFTPM2_KEY* parentKey, WOLFTPM2_KEY* key,
    int curveId, const byte* eccPubX, word32 eccPubXSz,
    const byte* eccPubY, word32 eccPubYSz,
    const byte* eccPriv, word32 eccPrivSz);
```

```
WOLFTPM_API int wolfTPM2_ReadPublicKey(WOLFTPM2_DEV* dev, WOLFTPM2_KEY* key,
    const TPM_HANDLE handle);

WOLFTPM_API int wolfTPM2_CreateKeySeal(WOLFTPM2_DEV* dev,
    WOLFTPM2_KEYBLOB* keyBlob, WOLFTPM2_HANDLE* parent,
    TPMT_PUBLIC* publicTemplate, const byte* auth, int authSz,
    const byte* sealData, int sealSize);

WOLFTPM_API int wolfTPM2_CreateKeySeal_ex(WOLFTPM2_DEV* dev,
    WOLFTPM2_KEYBLOB* keyBlob, WOLFTPM2_HANDLE* parent,
    TPMT_PUBLIC* publicTemplate, const byte* auth, int authSz,
    TPM_ALG_ID pcrAlg, byte* pcrArray, word32 pcrArraySz,
    const byte* sealData, int sealSize);

WOLFTPM_API int wolfTPM2_ComputeName(const TPM2B_PUBLIC* pub, TPM2B_NAME* out);

WOLFTPM_API int wolfTPM2_SensitiveToPrivate(TPM2B_SENSITIVE* sens,
 ↪  TPM2B_PRIVATE* priv,
    TPMI_ALG_HASH nameAlg, TPM2B_NAME* name, const WOLFTPM2_KEY* parentKey,
    TPMT_SYM_DEF_OBJECT* sym, TPM2B_DATA* symSeed);

WOLFTPM_API word32 wolfTPM2_RsaKey_Exponent(const byte* e, word32 eSz);

#ifndef WOLFTPM2_NO_WOLFCRYPT
WOLFTPM_API int wolfTPM2_ImportPrivateKeyBuffer(WOLFTPM2_DEV* dev,
    const WOLFTPM2_KEY* parentKey, int keyType, WOLFTPM2_KEYBLOB* keyBlob,
    int encodingType, const char* input, word32 inSz, const char* pass,
    TPMA_OBJECT objectAttributes, byte* seed, word32 seedSz);

WOLFTPM_API int wolfTPM2_ImportPublicKeyBuffer(WOLFTPM2_DEV* dev, int keyType,
    WOLFTPM2_KEY* key, int encodingType, const char* input, word32 inSz,
    TPMA_OBJECT objectAttributes);

WOLFTPM_API int wolfTPM2_ExportPublicKeyBuffer(WOLFTPM2_DEV* dev,
 ↪  WOLFTPM2_KEY* tpmKey,
    int encodingType, byte* out, word32* outSz);

#ifndef NO_RSA

WOLFTPM_API int wolfTPM2_RsaPrivateKeyImportDer(WOLFTPM2_DEV* dev,
    const WOLFTPM2_KEY* parentKey, WOLFTPM2_KEYBLOB* keyBlob, const byte*
     ↪  input,
    word32 inSz, TPMI_ALG_RSA_SCHEME scheme, TPMI_ALG_HASH hashAlg);

WOLFTPM_API int wolfTPM2_RsaPrivateKeyImportPem(WOLFTPM2_DEV* dev,
    const WOLFTPM2_KEY* parentKey, WOLFTPM2_KEYBLOB* keyBlob,
    const char* input, word32 inSz, char* pass,
    TPMI_ALG_RSA_SCHEME scheme, TPMI_ALG_HASH hashAlg);

WOLFTPM_API int wolfTPM2_RsaKey_TpmToWolf(WOLFTPM2_DEV* dev, WOLFTPM2_KEY*
 ↪  tpmKey,
    RsaKey* wolfKey);

WOLFTPM_API int wolfTPM2_RsaKey_TpmToPemPub(WOLFTPM2_DEV* dev,
```

```
                                        WOLFTPM2_KEY* keyBlob,
                                        byte* pem, word32* pemSz);

WOLFTPM_API int wolfTPM2_RsaKey_WolfToTpm(WOLFTPM2_DEV* dev, RsaKey* wolfKey,
    WOLFTPM2_KEY* tpmKey);

WOLFTPM_API int wolfTPM2_RsaKey_WolfToTpm_ex(WOLFTPM2_DEV* dev,
    const WOLFTPM2_KEY* parentKey, RsaKey* wolfKey, WOLFTPM2_KEY* tpmKey);

WOLFTPM_API int wolfTPM2_CreateRsaKeyBlob(WOLFTPM2_DEV* dev, const
 ↪  WOLFTPM2_KEY* parentKey,
    RsaKey* wolfKey, WOLFTPM2_KEYBLOB* tpmKey);

WOLFTPM_API int wolfTPM2_RsaKey_PubPemToTpm(WOLFTPM2_DEV* dev,
    WOLFTPM2_KEY* tpmKey, const byte* pem, word32 pemSz);

WOLFTPM_API int wolfTPM2_DecodeRsaDer(const byte* der, word32 derSz,
    TPM2B_PUBLIC* pub, TPM2B_SENSITIVE* sens, TPMA_OBJECT attributes);
#endif /* !NO_RSA */

#ifdef HAVE_ECC
WOLFTPM_API int wolfTPM2_EccKey_TpmToWolf(WOLFTPM2_DEV* dev, WOLFTPM2_KEY*
 ↪  tpmKey,
    ecc_key* wolfKey);

WOLFTPM_API int wolfTPM2_EccKey_WolfToTpm(WOLFTPM2_DEV* dev, ecc_key* wolfKey,
    WOLFTPM2_KEY* tpmKey);

WOLFTPM_API int wolfTPM2_EccKey_WolfToTpm_ex(WOLFTPM2_DEV* dev, WOLFTPM2_KEY*
 ↪  parentKey,
    ecc_key* wolfKey, WOLFTPM2_KEY* tpmKey);

WOLFTPM_API int wolfTPM2_CreateEccKeyBlob(WOLFTPM2_DEV* dev, WOLFTPM2_KEY*
 ↪  parentKey,
    ecc_key* wolfKey, WOLFTPM2_KEYBLOB* tpmKey);

WOLFTPM_API int wolfTPM2_EccKey_WolfToPubPoint(WOLFTPM2_DEV* dev, ecc_key*
 ↪  wolfKey,
    TPM2B_ECC_POINT* pubPoint);

WOLFTPM_API int wolfTPM2_DecodeEccDer(const byte* der, word32 derSz,
    TPM2B_PUBLIC* pub, TPM2B_SENSITIVE* sens, TPMA_OBJECT attributes);
#endif /* HAVE_ECC */
#endif /* !WOLFTPM2_NO_WOLFCRYPT */

WOLFTPM_API int wolfTPM2_SignHash(WOLFTPM2_DEV* dev, WOLFTPM2_KEY* key,
    const byte* digest, int digestSz, byte* sig, int* sigSz);

WOLFTPM_API int wolfTPM2_SignHashScheme(WOLFTPM2_DEV* dev, WOLFTPM2_KEY* key,
    const byte* digest, int digestSz, byte* sig, int* sigSz,
    TPMI_ALG_SIG_SCHEME sigAlg, TPMI_ALG_HASH hashAlg);

WOLFTPM_API int wolfTPM2_VerifyHash(WOLFTPM2_DEV* dev, WOLFTPM2_KEY* key,
    const byte* sig, int sigSz, const byte* digest, int digestSz);
```

```
WOLFTPM_API int wolfTPM2_VerifyHash_ex(WOLFTPM2_DEV* dev, WOLFTPM2_KEY* key,
    const byte* sig, int sigSz, const byte* digest, int digestSz,
    int hashAlg);

WOLFTPM_API int wolfTPM2_VerifyHashScheme(WOLFTPM2_DEV* dev, WOLFTPM2_KEY* key,
    const byte* sig, int sigSz, const byte* digest, int digestSz,
    TPMI_ALG_SIG_SCHEME sigAlg, TPMI_ALG_HASH hashAlg);

WOLFTPM_API int wolfTPM2_VerifyHashTicket(WOLFTPM2_DEV* dev,
    WOLFTPM2_KEY* key, const byte* sig, int sigSz, const byte* digest,
    int digestSz, TPMI_ALG_SIG_SCHEME sigAlg, TPMI_ALG_HASH hashAlg,
    TPMT_TK_VERIFIED* checkTicket);

#ifdef WOLFTPM_PQC
WOLFTPM_API int wolfTPM2_SignSequenceStart(WOLFTPM2_DEV* dev, WOLFTPM2_KEY*
 ↪  key,
    const byte* context, int contextSz, TPM_HANDLE* sequenceHandle);

WOLFTPM_API int wolfTPM2_SignSequenceUpdate(WOLFTPM2_DEV* dev,
    TPM_HANDLE sequenceHandle, const byte* data, int dataSz);

WOLFTPM_API int wolfTPM2_SignSequenceComplete(WOLFTPM2_DEV* dev,
    TPM_HANDLE sequenceHandle, WOLFTPM2_KEY* key, const byte* data, int dataSz,
    byte* sig, int* sigSz);

WOLFTPM_API int wolfTPM2_VerifySequenceStart(WOLFTPM2_DEV* dev, WOLFTPM2_KEY*
 ↪  key,
    const byte* context, int contextSz, TPM_HANDLE* sequenceHandle);

WOLFTPM_API int wolfTPM2_VerifySequenceUpdate(WOLFTPM2_DEV* dev,
    TPM_HANDLE sequenceHandle, const byte* data, int dataSz);

WOLFTPM_API int wolfTPM2_VerifySequenceComplete(WOLFTPM2_DEV* dev,
    TPM_HANDLE sequenceHandle, WOLFTPM2_KEY* key, const byte* data, int dataSz,
    const byte* sig, int sigSz, TPMT_TK_VERIFIED* validation);

WOLFTPM_API int wolfTPM2_SignDigest(WOLFTPM2_DEV* dev, WOLFTPM2_KEY* key,
    const byte* digest, int digestSz, const byte* context, int contextSz,
    byte* sig, int* sigSz);

WOLFTPM_API int wolfTPM2_VerifyDigestSignature(WOLFTPM2_DEV* dev,
 ↪  WOLFTPM2_KEY* key,
    const byte* digest, int digestSz, const byte* sig, int sigSz,
    const byte* context, int contextSz, TPMT_TK_VERIFIED* validation);

WOLFTPM_API int wolfTPM2_Encapsulate(WOLFTPM2_DEV* dev, WOLFTPM2_KEY* key,
    byte* ciphertext, int* ciphertextSz, byte* sharedSecret, int*
 ↪  sharedSecretSz);

WOLFTPM_API int wolfTPM2_Decapsulate(WOLFTPM2_DEV* dev, WOLFTPM2_KEY* key,
    const byte* ciphertext, int ciphertextSz, byte* sharedSecret, int*
     ↪  sharedSecretSz);
```

```
WOLFTPM_API int wolfTPM2_GetKeyTemplate_MLDSA(TPMT_PUBLIC* publicTemplate,
    TPMA_OBJECT objectAttributes, TPMI_MLDSA_PARAMETER_SET parameterSet,
    int allowExternalMu);

#ifdef WOLFTPM_MLDSA
WOLFTPM_API int wolfTPM2_GetKeyTemplate_MLDSA_ex(TPMT_PUBLIC* publicTemplate,
    TPMA_OBJECT objectAttributes, TPMI_MLDSA_PARAMETER_SET parameterSet,
    int allowExternalMu, TPMI_ALG_HASH nameAlg);
#endif

WOLFTPM_API int wolfTPM2_GetKeyTemplate_HASH_MLDSA(TPMT_PUBLIC* publicTemplate,
    TPMA_OBJECT objectAttributes, TPMI_MLDSA_PARAMETER_SET parameterSet,
    TPMI_ALG_HASH hashAlg);

#ifdef WOLFTPM_HASH_MLDSA
WOLFTPM_API int wolfTPM2_GetKeyTemplate_HASH_MLDSA_ex(TPMT_PUBLIC*
 ↪  publicTemplate,
    TPMA_OBJECT objectAttributes, TPMI_MLDSA_PARAMETER_SET parameterSet,
    TPMI_ALG_HASH hashAlg, TPMI_ALG_HASH nameAlg);
#endif

WOLFTPM_API int wolfTPM2_GetKeyTemplate_MLKEM(TPMT_PUBLIC* publicTemplate,
    TPMA_OBJECT objectAttributes, TPMI_MLKEM_PARAMETER_SET parameterSet);

#ifdef WOLFTPM_MLKEM
WOLFTPM_API int wolfTPM2_GetKeyTemplate_MLKEM_ex(TPMT_PUBLIC* publicTemplate,
    TPMA_OBJECT objectAttributes, TPMI_MLKEM_PARAMETER_SET parameterSet,
    TPMI_ALG_HASH nameAlg);
#endif
#endif /* WOLFTPM_PQC */

WOLFTPM_API int wolfTPM2_ECDHGenKey(WOLFTPM2_DEV* dev, WOLFTPM2_KEY* ecdhKey,
    int curve_id, const byte* auth, int authSz);

WOLFTPM_API int wolfTPM2_ECDHGen(WOLFTPM2_DEV* dev, WOLFTPM2_KEY* privKey,
    TPM2B_ECC_POINT* pubPoint, byte* out, int* outSz);

WOLFTPM_API int wolfTPM2_ECDHGenZ(WOLFTPM2_DEV* dev, WOLFTPM2_KEY* privKey,
    const TPM2B_ECC_POINT* pubPoint, byte* out, int* outSz);

WOLFTPM_API int wolfTPM2_ECDHEGenKey(WOLFTPM2_DEV* dev, WOLFTPM2_KEY* ecdhKey,
    int curve_id);

WOLFTPM_API int wolfTPM2_ECDHEGenZ(WOLFTPM2_DEV* dev, WOLFTPM2_KEY* parentKey,
    WOLFTPM2_KEY* ecdhKey, const TPM2B_ECC_POINT* pubPoint,
    byte* out, int* outSz);

WOLFTPM_API int wolfTPM2_RsaEncrypt(WOLFTPM2_DEV* dev, WOLFTPM2_KEY* key,
    TPM_ALG_ID padScheme, const byte* msg, int msgSz, byte* out, int* outSz);

WOLFTPM_API int wolfTPM2_RsaEncrypt_ex(WOLFTPM2_DEV* dev, WOLFTPM2_KEY* key,
    TPM_ALG_ID padScheme, const byte* msg, int msgSz, byte* out, int* outSz,
    TPMI_ALG_HASH hashAlg);
```

```
WOLFTPM_API int wolfTPM2_RsaDecrypt(WOLFTPM2_DEV* dev, WOLFTPM2_KEY* key,
    TPM_ALG_ID padScheme, const byte* in, int inSz, byte* msg, int* msgSz);

WOLFTPM_API int wolfTPM2_RsaDecrypt_ex(WOLFTPM2_DEV* dev, WOLFTPM2_KEY* key,
    TPM_ALG_ID padScheme, const byte* in, int inSz, byte* msg, int* msgSz,
    TPMI_ALG_HASH hashAlg);


WOLFTPM_API int wolfTPM2_ReadPCR(WOLFTPM2_DEV* dev,
    int pcrIndex, int hashAlg, byte* digest, int* pDigestLen);

WOLFTPM_API int wolfTPM2_ResetPCR(WOLFTPM2_DEV* dev, int pcrIndex);

WOLFTPM_API int wolfTPM2_SetLocality(WOLFTPM2_DEV* dev, int locality);

WOLFTPM_API int wolfTPM2_ExtendPCR(WOLFTPM2_DEV* dev, int pcrIndex, int
 ↪  hashAlg,
    const byte* digest, int digestLen);

WOLFTPM_API int wolfTPM2_AllocatePCRBanks(WOLFTPM2_DEV* dev,
    const TPM_ALG_ID* hashAlgs, int hashAlgCount, PCR_Allocate_Out* allocOut);

WOLFTPM_API int wolfTPM2_AllocatePCRBanks_ex(WOLFTPM2_DEV* dev,
    WOLFTPM2_SESSION* session, const TPM_ALG_ID* hashAlgs, int hashAlgCount,
    PCR_Allocate_Out* allocOut);

/* Newer API's that use WOLFTPM2_NV context and support auth */

WOLFTPM_API int wolfTPM2_NVCreateAuth(WOLFTPM2_DEV* dev, WOLFTPM2_HANDLE*
 ↪  parent,
    WOLFTPM2_NV* nv, word32 nvIndex, word32 nvAttributes, word32 maxSize,
    const byte* auth, int authSz);

WOLFTPM_API int wolfTPM2_NVCreateAuthPolicy(WOLFTPM2_DEV* dev,
 ↪  WOLFTPM2_HANDLE* parent,
    WOLFTPM2_NV* nv, word32 nvIndex, word32 nvAttributes, word32 maxSize,
    const byte* auth, int authSz, const byte* authPolicy, int authPolicySz);

WOLFTPM_API int wolfTPM2_NVCreateAuthPolicy_ex(WOLFTPM2_DEV* dev,
    WOLFTPM2_HANDLE* parent, WOLFTPM2_NV* nv, word32 nvIndex,
    word32 nvAttributes, word32 maxSize, const byte* auth, int authSz,
    const byte* authPolicy, int authPolicySz, TPMI_ALG_HASH nameAlg);

WOLFTPM_API int wolfTPM2_NVWriteAuth(WOLFTPM2_DEV* dev, WOLFTPM2_NV* nv,
    word32 nvIndex, byte* dataBuf, word32 dataSz, word32 offset);

WOLFTPM_API int wolfTPM2_NVWriteAuthPolicy(WOLFTPM2_DEV* dev,
 ↪  WOLFTPM2_SESSION* tpmSession,
    TPM_ALG_ID pcrAlg, byte* pcrArray, word32 pcrArraySz, WOLFTPM2_NV* nv,
    word32 nvIndex, byte* dataBuf, word32 dataSz, word32 offset);

WOLFTPM_API int wolfTPM2_NVExtend(WOLFTPM2_DEV* dev, WOLFTPM2_NV* nv,
    word32 nvIndex, byte* dataBuf, word32 dataSz);
```

```
WOLFTPM_API int wolfTPM2_NVReadAuth(WOLFTPM2_DEV* dev, WOLFTPM2_NV* nv,
    word32 nvIndex, byte* dataBuf, word32* pDataSz, word32 offset);

WOLFTPM_API int wolfTPM2_NVReadAuthPolicy(WOLFTPM2_DEV* dev, WOLFTPM2_SESSION*
 ↪  tpmSession,
    TPM_ALG_ID pcrAlg, byte* pcrArray, word32 pcrArraySz, WOLFTPM2_NV* nv,
    word32 nvIndex, byte* dataBuf, word32* pDataSz, word32 offset);

WOLFTPM_API int wolfTPM2_NVReadCert(WOLFTPM2_DEV* dev, TPM_HANDLE handle,
    uint8_t* buffer, uint32_t* len);

WOLFTPM_API int wolfTPM2_NVIncrement(WOLFTPM2_DEV* dev, WOLFTPM2_NV* nv);

WOLFTPM_API int wolfTPM2_NVOpen(WOLFTPM2_DEV* dev, WOLFTPM2_NV* nv,
    word32 nvIndex, const byte* auth, word32 authSz);

WOLFTPM_API int wolfTPM2_NVWriteLock(WOLFTPM2_DEV* dev, WOLFTPM2_NV* nv);

WOLFTPM_API int wolfTPM2_NVDeleteAuth(WOLFTPM2_DEV* dev, WOLFTPM2_HANDLE*
 ↪  parent,
    word32 nvIndex);

/* older API's with improper auth support, kept only for backwards
 ↪  compatibility */
WOLFTPM_API int wolfTPM2_NVCreate(WOLFTPM2_DEV* dev, TPM_HANDLE authHandle,
    word32 nvIndex, word32 nvAttributes, word32 maxSize, const byte* auth, int
 ↪  authSz);
WOLFTPM_API int wolfTPM2_NVWrite(WOLFTPM2_DEV* dev, TPM_HANDLE authHandle,
    word32 nvIndex, byte* dataBuf, word32 dataSz, word32 offset);
WOLFTPM_API int wolfTPM2_NVRead(WOLFTPM2_DEV* dev, TPM_HANDLE authHandle,
    word32 nvIndex, byte* dataBuf, word32* dataSz, word32 offset);
WOLFTPM_API int wolfTPM2_NVDelete(WOLFTPM2_DEV* dev, TPM_HANDLE authHandle,
    word32 nvIndex);

WOLFTPM_API int wolfTPM2_NVReadPublic(WOLFTPM2_DEV* dev, word32 nvIndex,
    TPMS_NV_PUBLIC* nvPublic);

WOLFTPM_API int wolfTPM2_NVStoreKey(WOLFTPM2_DEV* dev, TPM_HANDLE
 ↪  primaryHandle,
    WOLFTPM2_KEY* key, TPM_HANDLE persistentHandle);

WOLFTPM_API int wolfTPM2_NVDeleteKey(WOLFTPM2_DEV* dev, TPM_HANDLE
 ↪  primaryHandle,
    WOLFTPM2_KEY* key);

WOLFTPM_API struct WC_RNG* wolfTPM2_GetRng(WOLFTPM2_DEV* dev);

WOLFTPM_API int wolfTPM2_GetRandom(WOLFTPM2_DEV* dev, byte* buf, word32 len);

WOLFTPM_API int wolfTPM2_UnloadHandle(WOLFTPM2_DEV* dev, WOLFTPM2_HANDLE*
 ↪  handle);

WOLFTPM_API int wolfTPM2_Clear(WOLFTPM2_DEV* dev);
```

```
WOLFTPM_API int wolfTPM2_DictionaryAttackLockReset(WOLFTPM2_DEV* dev);

WOLFTPM_API int wolfTPM2_DictionaryAttackParameters(WOLFTPM2_DEV* dev,
    word32 newMaxTries, word32 newRecoveryTime, word32 lockoutRecovery);

WOLFTPM_API int wolfTPM2_HashStart(WOLFTPM2_DEV* dev, WOLFTPM2_HASH* hash,
    TPMI_ALG_HASH hashAlg, const byte* usageAuth, word32 usageAuthSz);

WOLFTPM_API int wolfTPM2_HashUpdate(WOLFTPM2_DEV* dev, WOLFTPM2_HASH* hash,
    const byte* data, word32 dataSz);

WOLFTPM_API int wolfTPM2_HashFinish(WOLFTPM2_DEV* dev, WOLFTPM2_HASH* hash,
    byte* digest, word32* digestSz);

WOLFTPM_API int wolfTPM2_LoadKeyedHashKey(WOLFTPM2_DEV* dev, WOLFTPM2_KEY* key,
    WOLFTPM2_HANDLE* parent, int hashAlg, const byte* keyBuf, word32 keySz,
    const byte* usageAuth, word32 usageAuthSz);

WOLFTPM_API int wolfTPM2_HmacStart(WOLFTPM2_DEV* dev, WOLFTPM2_HMAC* hmac,
    WOLFTPM2_HANDLE* parent, TPMI_ALG_HASH hashAlg, const byte* keyBuf, word32
 ↪  keySz,
    const byte* usageAuth, word32 usageAuthSz);

WOLFTPM_API int wolfTPM2_HmacUpdate(WOLFTPM2_DEV* dev, WOLFTPM2_HMAC* hmac,
    const byte* data, word32 dataSz);

WOLFTPM_API int wolfTPM2_HmacFinish(WOLFTPM2_DEV* dev, WOLFTPM2_HMAC* hmac,
    byte* digest, word32* digestSz);

WOLFTPM_API int wolfTPM2_LoadSymmetricKey(WOLFTPM2_DEV* dev,
    WOLFTPM2_KEY* key, int alg, const byte* keyBuf, word32 keySz);

#define WOLFTPM2_ENCRYPT NO
#define WOLFTPM2_DECRYPT YES
WOLFTPM_API int wolfTPM2_EncryptDecryptBlock(WOLFTPM2_DEV* dev, WOLFTPM2_KEY*
 ↪  key,
    const byte* in, byte* out, word32 inOutSz, byte* iv, word32 ivSz,
    int isDecrypt);
WOLFTPM_API int wolfTPM2_EncryptDecrypt(WOLFTPM2_DEV* dev, WOLFTPM2_KEY* key,
    const byte* in, byte* out, word32 inOutSz,
    byte* iv, word32 ivSz, int isDecrypt);

WOLFTPM_API int wolfTPM2_SetCommand(WOLFTPM2_DEV* dev, TPM_CC commandCode,
    int enableFlag);

WOLFTPM_API int wolfTPM2_Reset(WOLFTPM2_DEV* dev, int doShutdown, int
 ↪  doStartup);

WOLFTPM_API int wolfTPM2_Shutdown(WOLFTPM2_DEV* dev, int doStartup);

WOLFTPM_API int wolfTPM2_UnloadHandles(WOLFTPM2_DEV* dev, word32 handleStart,
    word32 handleCount);

WOLFTPM_API int wolfTPM2_UnloadHandles_AllTransient(WOLFTPM2_DEV* dev);
```

```
/* Utility functions */

WOLFTPM_API int wolfTPM2_GetKeyTemplate_RSA(TPMT_PUBLIC* publicTemplate,
    TPMA_OBJECT objectAttributes);

WOLFTPM_API int wolfTPM2_GetKeyTemplate_RSA_ex(TPMT_PUBLIC* publicTemplate,
    TPM_ALG_ID nameAlg, TPMA_OBJECT objectAttributes, int keyBits, long
 ↪  exponent,
    TPM_ALG_ID sigScheme, TPM_ALG_ID sigHash);

WOLFTPM_API int wolfTPM2_GetKeyTemplate_ECC(TPMT_PUBLIC* publicTemplate,
    TPMA_OBJECT objectAttributes, TPM_ECC_CURVE curve, TPM_ALG_ID sigScheme);

WOLFTPM_API int wolfTPM2_GetKeyTemplate_ECC_ex(TPMT_PUBLIC* publicTemplate,
    TPM_ALG_ID nameAlg, TPMA_OBJECT objectAttributes, TPM_ECC_CURVE curve,
    TPM_ALG_ID sigScheme, TPM_ALG_ID sigHash);

WOLFTPM_API int wolfTPM2_GetKeyTemplate_Symmetric(TPMT_PUBLIC* publicTemplate,
    int keyBits, TPM_ALG_ID algMode, int isSign, int isDecrypt);

WOLFTPM_API int wolfTPM2_GetKeyTemplate_KeyedHash(TPMT_PUBLIC* publicTemplate,
    TPM_ALG_ID hashAlg, int isSign, int isDecrypt);

WOLFTPM_API int wolfTPM2_GetKeyTemplate_KeySeal(TPMT_PUBLIC* publicTemplate,
 ↪  TPM_ALG_ID nameAlg);

WOLFTPM_API int wolfTPM2_GetKeyTemplate_EK(TPMT_PUBLIC* publicTemplate,
 ↪  TPM_ALG_ID alg,
    int keyBits, TPM_ECC_CURVE curveID, TPM_ALG_ID nameAlg, int highRange);

WOLFTPM_API int wolfTPM2_GetKeyTemplate_EKIndex(word32 nvIndex,
    TPMT_PUBLIC* publicTemplate);

WOLFTPM_API int wolfTPM2_GetKeyTemplate_RSA_EK(TPMT_PUBLIC* publicTemplate);

WOLFTPM_API int wolfTPM2_GetKeyTemplate_ECC_EK(TPMT_PUBLIC* publicTemplate);

WOLFTPM_API int wolfTPM2_GetKeyTemplate_RSA_SRK(TPMT_PUBLIC* publicTemplate);

WOLFTPM_API int wolfTPM2_GetKeyTemplate_ECC_SRK(TPMT_PUBLIC* publicTemplate);

WOLFTPM_API int wolfTPM2_GetKeyTemplate_RSA_AIK(TPMT_PUBLIC* publicTemplate);

WOLFTPM_API int wolfTPM2_GetKeyTemplate_ECC_AIK(TPMT_PUBLIC* publicTemplate);

#ifdef WOLFTPM_PROVISIONING
WOLFTPM_API int wolfTPM2_GetKeyTemplate_RSA_IAK(TPMT_PUBLIC* publicTemplate,
 ↪  int keyBits,
    TPM_ALG_ID hashAlg);
WOLFTPM_API int wolfTPM2_GetKeyTemplate_ECC_IAK(TPMT_PUBLIC* publicTemplate,
    TPM_ECC_CURVE curveID, TPM_ALG_ID hashAlg);

WOLFTPM_API int wolfTPM2_GetKeyTemplate_ECC_IDevID(TPMT_PUBLIC* publicTemplate,
```

```
    TPM_ECC_CURVE curveID, TPM_ALG_ID hashAlg);
WOLFTPM_API int wolfTPM2_GetKeyTemplate_RSA_IDevID(TPMT_PUBLIC*
 ↪  publicTemplate, int keyBits,
    TPM_ALG_ID hashAlg);
#endif /* WOLFTPM_PROVISIONING */

WOLFTPM_API int wolfTPM2_SetKeyTemplate_Unique(TPMT_PUBLIC* publicTemplate,
 ↪  const byte* unique, int uniqueSz);

WOLFTPM_API int wolfTPM2_GetNvAttributesTemplate(TPM_HANDLE auth, word32*
 ↪  nvAttributes);

WOLFTPM_API int wolfTPM2_CreateEK(WOLFTPM2_DEV* dev, WOLFTPM2_KEY* ekKey,
 ↪  TPM_ALG_ID alg);

WOLFTPM_API int wolfTPM2_CreateSRK(WOLFTPM2_DEV* dev, WOLFTPM2_KEY* srkKey,
 ↪  TPM_ALG_ID alg,
    const byte* auth, int authSz);
WOLFTPM_API int wolfTPM2_CreateAndLoadAIK(WOLFTPM2_DEV* dev, WOLFTPM2_KEY*
 ↪  aikKey,
    TPM_ALG_ID alg, WOLFTPM2_KEY* srkKey, const byte* auth, int authSz);

WOLFTPM_API int wolfTPM2_GetTime(WOLFTPM2_KEY* aikKey, GetTime_Out*
 ↪  getTimeOut);


#ifdef WOLFTPM2_CERT_GEN

WOLFTPM_API int wolfTPM2_CSR_SetCustomExt(WOLFTPM2_DEV* dev, WOLFTPM2_CSR* csr,
    int critical, const char *oid, const byte *der, word32 derSz);

WOLFTPM_API int wolfTPM2_CSR_SetKeyUsage(WOLFTPM2_DEV* dev, WOLFTPM2_CSR* csr,
    const char* keyUsage);

WOLFTPM_API int wolfTPM2_CSR_SetSubject(WOLFTPM2_DEV* dev, WOLFTPM2_CSR* csr,
    const char* subject);

WOLFTPM_API int wolfTPM2_CSR_MakeAndSign_ex(WOLFTPM2_DEV* dev, WOLFTPM2_CSR*
 ↪  csr,
    WOLFTPM2_KEY* key, int outFormat, byte* out, int outSz,
    int sigType, int selfSignCert, int devId);

WOLFTPM_API int wolfTPM2_CSR_MakeAndSign(WOLFTPM2_DEV* dev, WOLFTPM2_CSR* csr,
    WOLFTPM2_KEY* key, int outFormat, byte* out, int outSz);

WOLFTPM_API int wolfTPM2_CSR_Generate_ex(WOLFTPM2_DEV* dev, WOLFTPM2_KEY* key,
    const char* subject, const char* keyUsage, int outFormat,
    byte* out, int outSz, int sigType, int selfSignCert, int devId);

WOLFTPM_API int wolfTPM2_CSR_Generate(WOLFTPM2_DEV* dev, WOLFTPM2_KEY* key,
    const char* subject, const char* keyUsage, int outFormat,
    byte* out, int outSz);

#endif /* WOLFTPM2_CERT_GEN */
```

```
WOLFTPM_API int wolfTPM2_ChangePlatformAuth(WOLFTPM2_DEV* dev,
 ↪  WOLFTPM2_SESSION* session);

WOLFTPM_API int wolfTPM2_ChangeHierarchyAuth(WOLFTPM2_DEV* dev,
 ↪  WOLFTPM2_SESSION* session,
    TPMI_RH_HIERARCHY_AUTH authHandle);

/* moved to tpm.h native code. macros here for backwards compatibility */
#define wolfTPM2_SetupPCRSel  TPM2_SetupPCRSel
#define wolfTPM2_GetAlgName   TPM2_GetAlgName
#define wolfTPM2_GetRCString  TPM2_GetRCString
#define wolfTPM2_GetCurveSize TPM2_GetCurveSize

/* for encrypting secrets (like salt) used in auth sessions and external key
 ↪  import */
WOLFTPM_API int wolfTPM2_EncryptSecret(WOLFTPM2_DEV* dev, const WOLFTPM2_KEY*
 ↪  tpmKey,
    TPM2B_DATA *secret, TPM2B_ENCRYPTED_SECRET *encSecret, const char* label);


#if defined(WOLFTPM_CRYPTOCB) || defined(HAVE_PK_CALLBACKS)
struct TpmCryptoDevCtx;

typedef struct TpmCryptoDevCtx {
    WOLFTPM2_DEV* dev;
#ifndef NO_RSA
    WOLFTPM2_KEY* rsaKey;  /* RSA */
    #ifdef WOLFSSL_KEY_GEN
    WOLFTPM2_KEYBLOB* rsaKeyGen;  /* RSA KeyGen */
    #endif
#endif
#ifdef HAVE_ECC
    WOLFTPM2_KEY* eccKey;       /* ECDSA - public only */
    WOLFTPM2_KEYBLOB* ecdsaKey; /* ECDSA - retain encrypted private portion
 ↪  from keygen */
    WOLFTPM2_KEY* ecdhKey;      /* ECDH */
#endif
    WOLFTPM2_KEY* storageKey;
#ifdef WOLFTPM_USE_SYMMETRIC
    unsigned short useSymmetricOnTPM:1; /* if set indicates desire to use
     ↪  symmetric algorithms on TPM */
#endif
    unsigned short useFIPSMode:1; /* if set requires FIPS mode on TPM and no
     ↪  fallback to software algos */
#ifdef WOLFTPM_MLDSA_SIGN
    WOLFTPM2_KEY* mldsaKey; /* ML-DSA identity key; private key stays in TPM */
#endif
} TpmCryptoDevCtx;

#endif /* WOLFTPM_CRYPTOCB || HAVE_PK_CALLBACKS */

#ifdef WOLFTPM_CRYPTOCB
```

```
/* Key id asking key generation for a transient key at a handle the TPM picks */
#define WOLFTPM2_KEYID_TRANSIENT TRANSIENT_FIRST

WOLFTPM_API int wolfTPM2_CryptoDevCb(int devId, wc_CryptoInfo* info, void*
 ↪  ctx);

WOLFTPM_API int wolfTPM2_SetCryptoDevCb(WOLFTPM2_DEV* dev,
 ↪  CryptoDevCallbackFunc cb,
    TpmCryptoDevCtx* tpmCtx, int* pDevId);

WOLFTPM_API int wolfTPM2_ClearCryptoDevCb(WOLFTPM2_DEV* dev, int devId);

#endif /* WOLFTPM_CRYPTOCB */

#if defined(HAVE_PK_CALLBACKS) && !defined(WOLFTPM2_NO_WRAPPER) && \
    !defined(WOLFCRYPT_ONLY)
#ifndef NO_RSA
WOLFTPM_API int wolfTPM2_PK_RsaSign(WOLFSSL* ssl,
    const unsigned char* in, unsigned int inSz,
    unsigned char* out, word32* outSz,
    const unsigned char* keyDer, unsigned int keySz,
    void* ctx);

WOLFTPM_API int wolfTPM2_PK_RsaSignCheck(WOLFSSL* ssl,
    unsigned char* sig, unsigned int sigSz,
    unsigned char** out,
    const unsigned char* keyDer, unsigned int keySz,
    void* ctx);

#ifdef WC_RSA_PSS
WOLFTPM_API int wolfTPM2_PK_RsaPssSign(WOLFSSL* ssl,
    const unsigned char* in, unsigned int inSz,
    unsigned char* out, unsigned int* outSz,
    int hash, int mgf,
    const unsigned char* keyDer, unsigned int keySz,
    void* ctx);

WOLFTPM_API int wolfTPM2_PK_RsaPssSignCheck(WOLFSSL* ssl,
    unsigned char* sig, unsigned int sigSz, unsigned char** out,
    int hash, int mgf,
    const unsigned char* keyDer, unsigned int keySz,
    void* ctx);
#endif /* WC_RSA_PSS */
#endif /* !NO_RSA */
#ifdef HAVE_ECC
WOLFTPM_API int wolfTPM2_PK_EccSign(WOLFSSL* ssl,
    const unsigned char* in, unsigned int inSz,
    unsigned char* out, word32* outSz,
    const unsigned char* keyDer, unsigned int keySz,
    void* ctx);
#endif

/* Helpers for setting generic PK callbacks */
```

```
WOLFTPM_API int wolfTPM_PK_SetCb(WOLFSSL_CTX* ctx);
WOLFTPM_API int wolfTPM_PK_SetCbCtx(WOLFSSL* ssl, void* userCtx);

#endif /* HAVE_PK_CALLBACKS */

#ifndef WOLFTPM2_NO_HEAP

WOLFTPM_API WOLFTPM2_DEV* wolfTPM2_New(void);

WOLFTPM_API int wolfTPM2_Free(WOLFTPM2_DEV *dev);

WOLFTPM_API WOLFTPM2_KEYBLOB* wolfTPM2_NewKeyBlob(void);

WOLFTPM_API int wolfTPM2_FreeKeyBlob(WOLFTPM2_KEYBLOB* blob);

WOLFTPM_API TPMT_PUBLIC* wolfTPM2_NewPublicTemplate(void);

WOLFTPM_API int wolfTPM2_FreePublicTemplate(TPMT_PUBLIC* PublicTemplate);


WOLFTPM_API WOLFTPM2_KEY* wolfTPM2_NewKey(void);

WOLFTPM_API int wolfTPM2_FreeKey(WOLFTPM2_KEY* key);


WOLFTPM_API WOLFTPM2_SESSION* wolfTPM2_NewSession(void);

WOLFTPM_API int wolfTPM2_FreeSession(WOLFTPM2_SESSION* session);

#ifdef WOLFTPM2_CERT_GEN
WOLFTPM_API WOLFTPM2_CSR* wolfTPM2_NewCSR(void);
WOLFTPM_API int wolfTPM2_FreeCSR(WOLFTPM2_CSR* csr);
#endif
#endif /* !WOLFTPM2_NO_HEAP */

WOLFTPM_API WOLFTPM2_HANDLE* wolfTPM2_GetHandleRefFromKey(WOLFTPM2_KEY* key);

WOLFTPM_API WOLFTPM2_HANDLE*
 ↪  wolfTPM2_GetHandleRefFromKeyBlob(WOLFTPM2_KEYBLOB* keyBlob);

WOLFTPM_API WOLFTPM2_HANDLE*
 ↪  wolfTPM2_GetHandleRefFromSession(WOLFTPM2_SESSION* session);

WOLFTPM_API TPM_HANDLE wolfTPM2_GetHandleValue(WOLFTPM2_HANDLE* handle);

WOLFTPM_API int wolfTPM2_SetKeyAuthPassword(WOLFTPM2_KEY *key, const byte*
 ↪  auth,
    int authSz);

WOLFTPM_API int wolfTPM2_GetKeyBlobAsBuffer(byte *buffer, word32 bufferSz,
    WOLFTPM2_KEYBLOB* key);

WOLFTPM_API int wolfTPM2_GetKeyBlobAsSeparateBuffers(byte* pubBuffer,
    word32* pubBufferSz, byte* privBuffer, word32* privBufferSz,
```

```
    WOLFTPM2_KEYBLOB* key);

WOLFTPM_API int wolfTPM2_SetKeyBlobFromBuffer(WOLFTPM2_KEYBLOB* key,
    byte *buffer, word32 bufferSz);


WOLFTPM_API int wolfTPM2_PolicyRestart(WOLFTPM2_DEV* dev, TPM_HANDLE
 ↪  sessionHandle);

WOLFTPM_API int wolfTPM2_GetPolicyDigest(WOLFTPM2_DEV* dev, TPM_HANDLE
 ↪  sessionHandle,
    byte* policyDigest, word32* policyDigestSz);

WOLFTPM_API int wolfTPM2_PolicyPCR(WOLFTPM2_DEV* dev, TPM_HANDLE sessionHandle,
    TPM_ALG_ID pcrAlg, byte* pcrArray, word32 pcrArraySz);

#ifdef WOLFTPM_SPDM
WOLFTPM_API int wolfTPM2_PolicyTransportSPDM(WOLFTPM2_DEV* dev,
    TPM_HANDLE sessionHandle, const TPM2B_NAME* reqKeyName,
    const TPM2B_NAME* tpmKeyName);
#endif

WOLFTPM_API int wolfTPM2_PolicyAuthorize(WOLFTPM2_DEV* dev, TPM_HANDLE
 ↪  sessionHandle,
    const TPM2B_PUBLIC* pub, const TPMT_TK_VERIFIED* checkTicket,
    const byte* pcrDigest, word32 pcrDigestSz,
    const byte* policyRef, word32 policyRefSz);

WOLFTPM_API int wolfTPM2_PCRGetDigest(WOLFTPM2_DEV* dev, TPM_ALG_ID pcrAlg,
    byte* pcrArray, word32 pcrArraySz, byte* pcrDigest, word32* pcrDigestSz);

WOLFTPM_API int wolfTPM2_PolicyRefMake(TPM_ALG_ID pcrAlg, byte* digest, word32*
 ↪  digestSz,
    const byte* policyRef, word32 policyRefSz);

WOLFTPM_API int wolfTPM2_PolicyPCRMake(TPM_ALG_ID pcrAlg,
    byte* pcrArray, word32 pcrArraySz, const byte* pcrDigest, word32
 ↪  pcrDigestSz,
    byte* digest, word32* digestSz);

#ifdef WOLFTPM_SPDM
WOLFTPM_API int wolfTPM2_PolicyTransportSPDMMake(TPM_ALG_ID hashAlg,
    const TPM2B_NAME* reqKeyName, const TPM2B_NAME* tpmKeyName,
    byte* digest, word32* digestSz);
#endif

WOLFTPM_API int wolfTPM2_PolicyCommandCodeMake(TPM_ALG_ID hashAlg,
    byte* digest, word32* digestSz, TPM_CC cc);

WOLFTPM_API int wolfTPM2_PolicyHash(TPM_ALG_ID hashAlg,
    byte* digest, word32* digestSz, TPM_CC cc,
    const byte* input, word32 inputSz);

WOLFTPM_API int wolfTPM2_PolicyAuthorizeMake(TPM_ALG_ID pcrAlg,
```

```
    const TPM2B_PUBLIC* pub, byte* digest, word32* digestSz,
    const byte* policyRef, word32 policyRefSz);

WOLFTPM_API int wolfTPM2_PolicyPassword(WOLFTPM2_DEV* dev,
    WOLFTPM2_SESSION* tpmSession, const byte* auth, int authSz);

WOLFTPM_API int wolfTPM2_PolicyAuthValue(WOLFTPM2_DEV* dev,
    WOLFTPM2_SESSION* tpmSession, const byte* auth, int authSz);

WOLFTPM_API int wolfTPM2_PolicyCommandCode(WOLFTPM2_DEV* dev,
    WOLFTPM2_SESSION* tpmSession, TPM_CC cc);

WOLFTPM_API int wolfTPM2_PolicyOR(WOLFTPM2_DEV* dev,
    WOLFTPM2_SESSION* tpmSession, const TPML_DIGEST* pHashList);

WOLFTPM_API int wolfTPM2_SetPrimaryPolicy(WOLFTPM2_DEV* dev,
    TPMI_RH_HIERARCHY_AUTH authHandle, TPM_ALG_ID hashAlg,
    const byte* authPolicy, word32 authPolicySz);


/* Pre-provisioned IAK and IDevID key/cert from TPM vendor */
/* Tested with ST33KTPM devices */
/* Default assumes: ECDSA SECP384R1, SHA2-384 */
#ifdef WOLFTPM_MFG_IDENTITY

/* Initial Attestation Key (IAK):
 * Restrictive: Can only sign data generated by the TPM like a TPM2_Quote */
#ifndef TPM2_IAK_KEY_HANDLE
#define TPM2_IAK_KEY_HANDLE     0x81020001
#endif
#ifndef TPM2_IAK_CERT_HANDLE
#define TPM2_IAK_CERT_HANDLE    0x1C90100
#endif
/* Initial Device ID (IDevID):
 * Non-Restrictive: Can sign external data */
#ifndef TPM2_IDEVID_KEY_HANDLE
#define TPM2_IDEVID_KEY_HANDLE  0x81020000
#endif
#ifndef TPM2_IDEVID_CERT_HANDLE
#define TPM2_IDEVID_CERT_HANDLE 0x1C90200
#endif

WOLFTPM_API int wolfTPM2_SetIdentityAuth(WOLFTPM2_DEV* dev, WOLFTPM2_HANDLE*
 ↪  handle,
    uint8_t* masterPassword, uint16_t masterPasswordSz);

#endif /* WOLFTPM_MFG_IDENTITY */


/* Internal API's */
WOLFTPM_LOCAL int GetKeyTemplateRSA(TPMT_PUBLIC* publicTemplate,
    TPM_ALG_ID nameAlg, TPMA_OBJECT objectAttributes, int keyBits, long
 ↪  exponent,
    TPM_ALG_ID sigScheme, TPM_ALG_ID sigHash);
```

```
WOLFTPM_LOCAL int GetKeyTemplateECC(TPMT_PUBLIC* publicTemplate,
    TPM_ALG_ID nameAlg, TPMA_OBJECT objectAttributes, TPM_ECC_CURVE curve,
    TPM_ALG_ID sigScheme, TPM_ALG_ID sigHash);

WOLFTPM_TEST_API int wolfTPM2_EccZToBuffer(byte* out, int* outSz,
    const TPM2B_ECC_PARAMETER* z);


#ifdef WOLFTPM_FIRMWARE_UPGRADE
typedef int (*wolfTPM2FwDataCb)(
    uint8_t* data, uint32_t data_req_sz, uint32_t offset, void* cb_ctx);

WOLFTPM_API int wolfTPM2_FirmwareUpgradeHash(WOLFTPM2_DEV* dev,
    TPM_ALG_ID hashAlg, /* Can use SHA2-384 or SHA2-512 for manifest hash */
    uint8_t* manifest_hash, uint32_t manifest_hash_sz,
    uint8_t* manifest, uint32_t manifest_sz,
    wolfTPM2FwDataCb cb, void* cb_ctx);

WOLFTPM_API int wolfTPM2_FirmwareUpgradeHash_ex(WOLFTPM2_DEV* dev,
    TPM_ALG_ID hashAlg, /* Can use SHA2-384 or SHA2-512 for manifest hash */
    uint8_t* manifest_hash, uint32_t manifest_hash_sz,
    uint8_t* manifest, uint32_t manifest_sz,
    wolfTPM2FwDataCb cb, void* cb_ctx, WOLFTPM2_SESSION* startSession);

#ifndef WOLFTPM2_NO_WOLFCRYPT
WOLFTPM_API int wolfTPM2_FirmwareUpgrade(WOLFTPM2_DEV* dev,
    uint8_t* manifest, uint32_t manifest_sz,
    wolfTPM2FwDataCb cb, void* cb_ctx);

WOLFTPM_API int wolfTPM2_FirmwareUpgrade_ex(WOLFTPM2_DEV* dev,
    uint8_t* manifest, uint32_t manifest_sz,
    wolfTPM2FwDataCb cb, void* cb_ctx, WOLFTPM2_SESSION* startSession);
#endif /* !WOLFTPM2_NO_WOLFCRYPT */

WOLFTPM_API int wolfTPM2_FirmwareUpgradeRecover(WOLFTPM2_DEV* dev,
    uint8_t* manifest, uint32_t manifest_sz,
    wolfTPM2FwDataCb cb, void* cb_ctx);

WOLFTPM_API int wolfTPM2_FirmwareUpgradeRecover_ex(WOLFTPM2_DEV* dev,
    uint8_t* manifest, uint32_t manifest_sz,
    wolfTPM2FwDataCb cb, void* cb_ctx, WOLFTPM2_SESSION* startSession);

WOLFTPM_API int wolfTPM2_FirmwareUpgradeCancel(WOLFTPM2_DEV* dev);

#if defined(WOLFTPM_ST33) || defined(WOLFTPM_AUTODETECT)
WOLFTPM_API int wolfTPM2_ST33_ManifestVersion(const uint8_t* manifest,
    uint32_t manifest_sz, word16* major, word16* minor);

WOLFTPM_API int wolfTPM2_ST33_CmdImplemented(TPM_CC cc, int* isImpl);

WOLFTPM_API int wolfTPM2_ST33_GetFwUpgradeCommands(const WOLFTPM2_CAPS* caps,
    word16 manifestMajor, TPM_CC* ccStart, TPM_CC* ccData, int* fromTpm);
```

```
WOLFTPM_TEST_API int wolfTPM2_ST33_FwUpgradeCommands(word16 fwVerMinor,
    int haveFwVer, word16 manifestMajor, TPM_CC* ccStart, TPM_CC* ccData);
#endif /* WOLFTPM_ST33 || WOLFTPM_AUTODETECT */

#endif /* WOLFTPM_FIRMWARE_UPGRADE */


#ifdef __cplusplus
    }  /* extern "C" */
#endif

#endif /* __TPM2_WRAP_H__ */
```

29.10 hal/tpm_io.h

29.10.1 Functions

	Name
WOLFTPM_API int	**TPM2_IoCb**(TPM2_CTX * ctx, const BYTE * txBuf, BYTE * rxBuf, UINT16 xferSz, void * userCtx)
WOLFTPM_LOCAL int	**TPM2_IoCb_Linux_I2C**(TPM2_CTX * ctx, int isRead, word32 addr, byte * buf, word16 size, void * userCtx)
WOLFTPM_LOCAL int	**TPM2_IoCb_STCubeMX_I2C**(TPM2_CTX * ctx, int isRead, word32 addr, byte * buf, word16 size, void * userCtx)
WOLFTPM_LOCAL int	**TPM2_IoCb_Infineon_I2C**(TPM2_CTX * ctx, int isRead, word32 addr, byte * buf, word16 size, void * userCtx)
WOLFTPM_LOCAL int	**TPM2_IoCb_Espressif_I2C**(TPM2_CTX * ctx, int isRead, word32 addr, byte * buf, word16 size, void * userCtx)
WOLFTPM_LOCAL int	**TPM2_IoCb_MicrochipHarmony_I2C**(TPM2_CTX * ctx, int isRead, word32 addr, byte * buf, word16 size, void * userCtx)
WOLFTPM_LOCAL int	**TPM2_IoCb_Zephyr_I2C**(TPM2_CTX * ctx, int isRead, word32 addr, byte * buf, word16 size, void * userCtx)
WOLFTPM_LOCAL int	**TPM2_IoCb_Wolfhal_I2C**(TPM2_CTX * ctx, int isRead, word32 addr, byte * buf, word16 size, void * userCtx)
WOLFTPM_LOCAL int	**TPM2_IoCb_Atmel_SPI**(TPM2_CTX * ctx, const byte * txBuf, byte * rxBuf, word16 xferSz, void * userCtx)
WOLFTPM_LOCAL int	**TPM2_IoCb_Barebox_SPI**(TPM2_CTX * ctx, const byte * txBuf, byte * rxBuf, word16 xferSz, void * userCtx)
WOLFTPM_LOCAL int	**TPM2_IoCb_Uboot_SPI**(TPM2_CTX * ctx, const byte * txBuf, byte * rxBuf, word16 xferSz, void * userCtx)
WOLFTPM_LOCAL int	**TPM2_IoCb_Linux_SPI**(TPM2_CTX * ctx, const byte * txBuf, byte * rxBuf, word16 xferSz, void * userCtx)

	Name
WOLFTPM_LOCAL int	**TPM2_IoCb_STCubeMX_SPI**(TPM2_CTX * ctx, const byte * txBuf, byte * rxBuf, word16 xferSz, void * userCtx)
WOLFTPM_LOCAL int	**TPM2_IoCb_QNX_SPI**(TPM2_CTX * ctx, const byte * txBuf, byte * rxBuf, word16 xferSz, void * userCtx)
WOLFTPM_LOCAL int	**TPM2_IoCb_Xilinx_SPI**(TPM2_CTX * ctx, const byte * txBuf, byte * rxBuf, word16 xferSz, void * userCtx)
WOLFTPM_LOCAL int	**TPM2_IoCb_Infineon_SPI**(TPM2_CTX * ctx, const byte * txBuf, byte * rxBuf, word16 xferSz, void * userCtx)
WOLFTPM_LOCAL int	**TPM2_IoCb_Infineon_TriCore_SPI**(TPM2_CTX * ctx, const byte * txBuf, byte * rxBuf, word16 xferSz, void * userCtx)
WOLFTPM_LOCAL int	**TPM2_IoCb_Microchip_SPI**(TPM2_CTX * ctx, const byte * txBuf, byte * rxBuf, word16 xferSz, void * userCtx)
WOLFTPM_LOCAL int	**TPM2_IoCb_Wolfhal_SPI**(TPM2_CTX * ctx, const byte * txBuf, byte * rxBuf, word16 xferSz, void * userCtx)
WOLFTPM_LOCAL int	**TPM2_IoCb_Mmio**(TPM2_CTX * ctx, int isRead, word32 addr, byte * buf, word16 size, void * userCtx)
WOLFTPM_LOCAL int	**TPM2_IoCb_FwTPM**(TPM2_CTX * ctx, int isRead, word32 addr, byte * buf, word16 size, void * userCtx)
WOLFTPM_API int	**TPM2_IoCb_Reset**(TPM2_CTX * ctx, void * userCtx)
WOLFTPM_LOCAL int	**TPM2_IoCb_Linux_Reset**(TPM2_CTX * ctx, void * userCtx)

29.10.2 Attributes

Name
C

29.10.3 Functions Documentation

29.10.3.1 function TPM2_IoCb

```
WOLFTPM_API int TPM2_IoCb(
    TPM2_CTX * ctx,
    const BYTE * txBuf,
    BYTE * rxBuf,
    UINT16 xferSz,
    void * userCtx
)
```

29.10.3.2 function TPM2_IoCb_Linux_I2C

```
WOLFTPM_LOCAL int TPM2_IoCb_Linux_I2C(
    TPM2_CTX * ctx,
    int isRead,
    word32 addr,
    byte * buf,
    word16 size,
    void * userCtx
)
```

29.10.3.3 function TPM2_IoCb_STCubeMX_I2C

```
WOLFTPM_LOCAL int TPM2_IoCb_STCubeMX_I2C(
    TPM2_CTX * ctx,
    int isRead,
    word32 addr,
    byte * buf,
    word16 size,
    void * userCtx
)
```

29.10.3.4 function TPM2_IoCb_Infineon_I2C

```
WOLFTPM_LOCAL int TPM2_IoCb_Infineon_I2C(
    TPM2_CTX * ctx,
    int isRead,
    word32 addr,
    byte * buf,
    word16 size,
    void * userCtx
)
```

29.10.3.5 function TPM2_IoCb_Espressif_I2C

```
WOLFTPM_LOCAL int TPM2_IoCb_Espressif_I2C(
    TPM2_CTX * ctx,
    int isRead,
    word32 addr,
    byte * buf,
    word16 size,
    void * userCtx
)
```

29.10.3.6 function TPM2_IoCb_MicrochipHarmony_I2C

```
WOLFTPM_LOCAL int TPM2_IoCb_MicrochipHarmony_I2C(
    TPM2_CTX * ctx,
    int isRead,
    word32 addr,
    byte * buf,
    word16 size,
    void * userCtx
)
```

29.10.3.7 function TPM2_IoCb_Zephyr_I2C

```
WOLFTPM_LOCAL int TPM2_IoCb_Zephyr_I2C(
    TPM2_CTX * ctx,
    int isRead,
    word32 addr,
    byte * buf,
    word16 size,
    void * userCtx
)
```

29.10.3.8 function TPM2_IoCb_Wolfhal_I2C

```
WOLFTPM_LOCAL int TPM2_IoCb_Wolfhal_I2C(
    TPM2_CTX * ctx,
    int isRead,
    word32 addr,
    byte * buf,
    word16 size,
    void * userCtx
)
```

29.10.3.9 function TPM2_IoCb_Atmel_SPI

```
WOLFTPM_LOCAL int TPM2_IoCb_Atmel_SPI(
    TPM2_CTX * ctx,
    const byte * txBuf,
    byte * rxBuf,
    word16 xferSz,
    void * userCtx
)
```

29.10.3.10 function TPM2_IoCb_Barebox_SPI

```
WOLFTPM_LOCAL int TPM2_IoCb_Barebox_SPI(
    TPM2_CTX * ctx,
    const byte * txBuf,
    byte * rxBuf,
    word16 xferSz,
    void * userCtx
)
```

29.10.3.11 function TPM2_IoCb_Uboot_SPI

```
WOLFTPM_LOCAL int TPM2_IoCb_Uboot_SPI(
    TPM2_CTX * ctx,
    const byte * txBuf,
    byte * rxBuf,
    word16 xferSz,
    void * userCtx
)
```

29.10.3.12 function TPM2_IoCb_Linux_SPI

```
WOLFTPM_LOCAL int TPM2_IoCb_Linux_SPI(
    TPM2_CTX * ctx,
    const byte * txBuf,
    byte * rxBuf,
    word16 xferSz,
    void * userCtx
)
```

29.10.3.13 function TPM2_IoCb_STCubeMX_SPI

```
WOLFTPM_LOCAL int TPM2_IoCb_STCubeMX_SPI(
    TPM2_CTX * ctx,
    const byte * txBuf,
    byte * rxBuf,
    word16 xferSz,
    void * userCtx
)
```

29.10.3.14 function TPM2_IoCb_QNX_SPI

```
WOLFTPM_LOCAL int TPM2_IoCb_QNX_SPI(
    TPM2_CTX * ctx,
    const byte * txBuf,
    byte * rxBuf,
    word16 xferSz,
    void * userCtx
)
```

29.10.3.15 function TPM2_IoCb_Xilinx_SPI

```
WOLFTPM_LOCAL int TPM2_IoCb_Xilinx_SPI(
    TPM2_CTX * ctx,
    const byte * txBuf,
    byte * rxBuf,
    word16 xferSz,
    void * userCtx
)
```

29.10.3.16 function TPM2_IoCb_Infineon_SPI

```
WOLFTPM_LOCAL int TPM2_IoCb_Infineon_SPI(
    TPM2_CTX * ctx,
    const byte * txBuf,
    byte * rxBuf,
    word16 xferSz,
    void * userCtx
)
```

29.10.3.17 function TPM2_IoCb_Infineon_TriCore_SPI

```
WOLFTPM_LOCAL int TPM2_IoCb_Infineon_TriCore_SPI(
    TPM2_CTX * ctx,
    const byte * txBuf,
    byte * rxBuf,
```

```
    word16 xferSz,
    void * userCtx
)
```

29.10.3.18 function TPM2_IoCb_Microchip_SPI

```
WOLFTPM_LOCAL int TPM2_IoCb_Microchip_SPI(
    TPM2_CTX * ctx,
    const byte * txBuf,
    byte * rxBuf,
    word16 xferSz,
    void * userCtx
)
```

29.10.3.19 function TPM2_IoCb_Wolfhal_SPI

```
WOLFTPM_LOCAL int TPM2_IoCb_Wolfhal_SPI(
    TPM2_CTX * ctx,
    const byte * txBuf,
    byte * rxBuf,
    word16 xferSz,
    void * userCtx
)
```

29.10.3.20 function TPM2_IoCb_Mmio

```
WOLFTPM_LOCAL int TPM2_IoCb_Mmio(
    TPM2_CTX * ctx,
    int isRead,
    word32 addr,
    byte * buf,
    word16 size,
    void * userCtx
)
```

29.10.3.21 function TPM2_IoCb_FwTPM

```
WOLFTPM_LOCAL int TPM2_IoCb_FwTPM(
    TPM2_CTX * ctx,
    int isRead,
    word32 addr,
    byte * buf,
    word16 size,
    void * userCtx
)
```

29.10.3.22 function TPM2_IoCb_Reset

```
WOLFTPM_API int TPM2_IoCb_Reset(
    TPM2_CTX * ctx,
    void * userCtx
)
```

29.10.3.23 function TPM2_IoCb_Linux_Reset

```
WOLFTPM_LOCAL int TPM2_IoCb_Linux_Reset(
    TPM2_CTX * ctx,
    void * userCtx
)
```

29.10.4 Attributes Documentation

29.10.4.1 variable C

```
C {
#endif




#if defined(WOLFTPM_LINUX_DEV) || defined(WOLFTPM_SWTPM) || \
    defined(WOLFTPM_OPTEE) || defined(WOLFTPM_WINAPI)


#define TPM2_IoCb NULL

#else

#if defined(WOLFTPM_EXAMPLE_HAL) || defined(WOLFTPM_MMIO) || \
    defined(WOLFTPM_FWTPM_HAL)

#ifdef WOLFTPM_ADV_IO
WOLFTPM_API int TPM2_IoCb(TPM2_CTX* ctx, INT32 isRead, UINT32 addr,
    BYTE* buf, UINT16 size, void* userCtx);
```

29.10.5 Source code

```
/* tpm_io.h
 *
 * Copyright (C) 2006-2026 wolfSSL Inc.
 *
 * This file is part of wolfTPM.
 *
 * wolfTPM is free software; you can redistribute it and/or modify
 * it under the terms of the GNU General Public License as published by
 * the Free Software Foundation; either version 3 of the License, or
 * (at your option) any later version.
 *
 * wolfTPM is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
 * GNU General Public License for more details.
 *
 * You should have received a copy of the GNU General Public License
 * along with this program; if not, write to the Free Software
 * Foundation, Inc., 51 Franklin Street, Fifth Floor, Boston, MA 02110-1335, USA
 */
```

```
#ifndef _TPM_IO_H_
#define _TPM_IO_H_

#include <wolftpm/tpm2.h>

#ifdef __cplusplus
    extern "C" {
#endif

/* TPM2 IO Examples */

#if defined(WOLFTPM_LINUX_DEV) || defined(WOLFTPM_SWTPM) || \
    defined(WOLFTPM_OPTEE) || defined(WOLFTPM_WINAPI)

/* HAL not required, so use NULL */
#define TPM2_IoCb NULL

#else

#if defined(WOLFTPM_EXAMPLE_HAL) || defined(WOLFTPM_MMIO) || \
    defined(WOLFTPM_FWTPM_HAL)

#ifdef WOLFTPM_ADV_IO
WOLFTPM_API int TPM2_IoCb(TPM2_CTX* ctx, INT32 isRead, UINT32 addr,
    BYTE* buf, UINT16 size, void* userCtx);
#else
WOLFTPM_API int TPM2_IoCb(TPM2_CTX* ctx, const BYTE* txBuf, BYTE* rxBuf,
    UINT16 xferSz, void* userCtx);
#endif

/* Platform support, in alphabetical order */
#ifdef WOLFTPM_I2C

#if defined(__linux__)
WOLFTPM_LOCAL int TPM2_IoCb_Linux_I2C(TPM2_CTX* ctx, int isRead, word32 addr,
 ↪  byte* buf,
    word16 size, void* userCtx);
#elif defined(WOLFSSL_STM32_CUBEMX)
WOLFTPM_LOCAL int TPM2_IoCb_STCubeMX_I2C(TPM2_CTX* ctx, int isRead, word32
 ↪  addr,
    byte* buf, word16 size, void* userCtx);
#elif defined(CY_USING_HAL)
WOLFTPM_LOCAL int TPM2_IoCb_Infineon_I2C(TPM2_CTX* ctx, int isRead, word32
 ↪  addr,
    byte* buf, word16 size, void* userCtx);
#elif defined(WOLFSSL_ESPIDF)
WOLFTPM_LOCAL int TPM2_IoCb_Espressif_I2C(TPM2_CTX* ctx, int isRead, word32
 ↪  addr,
    byte* buf, word16 size, void* userCtx);
#elif defined(WOLFTPM_MICROCHIP_HARMONY)
WOLFTPM_LOCAL int TPM2_IoCb_MicrochipHarmony_I2C(TPM2_CTX* ctx, int isRead,
 ↪  word32 addr,
    byte* buf, word16 size, void* userCtx);
```

```
#elif defined(WOLFSSL_ZEPHYR)
WOLFTPM_LOCAL int TPM2_IoCb_Zephyr_I2C(TPM2_CTX* ctx, int isRead, word32 addr,
    byte* buf, word16 size, void* userCtx);
#elif defined(WOLFTPM_WOLFHAL)
WOLFTPM_LOCAL int TPM2_IoCb_Wolfhal_I2C(TPM2_CTX* ctx, int isRead, word32 addr,
    byte* buf, word16 size, void* userCtx);
#endif /* __linux__ */

#else /* SPI */

#if defined(WOLFSSL_ATMEL)
WOLFTPM_LOCAL int TPM2_IoCb_Atmel_SPI(TPM2_CTX* ctx, const byte* txBuf, byte*
 ↪  rxBuf,
    word16 xferSz, void* userCtx);
#elif defined(__BAREBOX__)
WOLFTPM_LOCAL int TPM2_IoCb_Barebox_SPI(TPM2_CTX* ctx, const byte* txBuf,
    byte* rxBuf, word16 xferSz, void* userCtx);
#elif defined(__UBOOT__)
WOLFTPM_LOCAL int TPM2_IoCb_Uboot_SPI(TPM2_CTX* ctx, const byte* txBuf,
    byte* rxBuf, word16 xferSz, void* userCtx);
#elif defined(__linux__)
WOLFTPM_LOCAL int TPM2_IoCb_Linux_SPI(TPM2_CTX* ctx, const byte* txBuf, byte*
 ↪  rxBuf,
    word16 xferSz, void* userCtx);
#elif defined(WOLFSSL_STM32_CUBEMX)
WOLFTPM_LOCAL int TPM2_IoCb_STCubeMX_SPI(TPM2_CTX* ctx, const byte* txBuf,
 ↪  byte* rxBuf,
    word16 xferSz, void* userCtx);
#elif defined(__QNX__) || defined(__QNXTO__)
WOLFTPM_LOCAL int TPM2_IoCb_QNX_SPI(TPM2_CTX* ctx, const byte* txBuf,
    byte* rxBuf, word16 xferSz, void* userCtx);
#elif defined(__XILINX__)
WOLFTPM_LOCAL int TPM2_IoCb_Xilinx_SPI(TPM2_CTX* ctx, const byte* txBuf,
    byte* rxBuf, word16 xferSz, void* userCtx);
#elif defined(CY_USING_HAL)
WOLFTPM_LOCAL int TPM2_IoCb_Infineon_SPI(TPM2_CTX* ctx, const byte* txBuf,
    byte* rxBuf, word16 xferSz, void* userCtx);
#elif defined(WOLFTPM_INFINEON_TRICORE)
WOLFTPM_LOCAL int TPM2_IoCb_Infineon_TriCore_SPI(TPM2_CTX* ctx, const byte*
 ↪  txBuf,
    byte* rxBuf, word16 xferSz, void* userCtx);
#elif defined(WOLFTPM_MICROCHIP_HARMONY)
WOLFTPM_LOCAL int TPM2_IoCb_Microchip_SPI(TPM2_CTX* ctx, const byte* txBuf,
 ↪  byte* rxBuf,
    word16 xferSz, void* userCtx);
#elif defined(WOLFTPM_WOLFHAL)
WOLFTPM_LOCAL int TPM2_IoCb_Wolfhal_SPI(TPM2_CTX* ctx, const byte* txBuf,
    byte* rxBuf, word16 xferSz, void* userCtx);
#endif

#endif /* WOLFTPM_I2C */

#if defined(WOLFTPM_MMIO)
/* requires WOLFTPM_ADV_IO */
```

```
WOLFTPM_LOCAL int TPM2_IoCb_Mmio(TPM2_CTX* ctx, int isRead, word32 addr, byte*
 ↪  buf,
    word16 size, void* userCtx);
#endif

#if defined(WOLFTPM_FWTPM_HAL)
/* fwTPM TIS/shared-memory transport (requires WOLFTPM_ADV_IO) */
WOLFTPM_LOCAL int TPM2_IoCb_FwTPM(TPM2_CTX* ctx, int isRead, word32 addr,
    byte* buf, word16 size, void* userCtx);
#endif

#ifdef WOLFTPM_HAL_RESET
/* Optional TPM hardware reset (nRST) control. Pulses the reset line to reset
 * the TPM. Enable with --enable-hal-reset (-DWOLFTPM_HAL_RESET). */
WOLFTPM_API int TPM2_IoCb_Reset(TPM2_CTX* ctx, void* userCtx);
#if defined(__linux__)
WOLFTPM_LOCAL int TPM2_IoCb_Linux_Reset(TPM2_CTX* ctx, void* userCtx);
#endif
#endif

#else

/* No example HAL: caller supplies its own IO callback to wolfTPM2_Init */
#define TPM2_IoCb NULL

#endif /* WOLFTPM_EXAMPLE_HAL || WOLFTPM_MMIO || WOLFTPM_FWTPM_HAL */
#endif /* !(WOLFTPM_LINUX_DEV || WOLFTPM_SWTPM || WOLFTPM_WINAPI) */

#ifdef __cplusplus
    }  /* extern "C" */
#endif

#endif /* _TPM_IO_H_ */
```

30 Testing

This page describes how to run the wolfTPM tests locally and lists the continuous integration workflows that run on the repository.

30.1 Running tests locally

Build and run the main test suite:

```
./configure
make check
```

`make check` runs the unit tests, the fwTPM tests and the post-quantum (PQC) tests. The unit test sources are:

File	Purpose
`tests/unit_tests.c`	wolfTPM library unit tests
`tests/fwtpm_unit_tests.c`	fwTPM command processor unit tests
`tests/fwtpm_hal_unit_tests.c`	fwTPM HAL unit tests

The shell-based tests are:

File	Purpose
`tests/fwtpm_check.sh`	Entry point used by `make check` for the fwTPM tests
`tests/fwtpm_da_retry.sh`	Dictionary attack and retry handling
`tests/pqc_mssim_e2e.sh`	Post-quantum end-to-end test

To run the example programs against a TPM or simulator:

```
./examples/run_examples.sh
```

The script reads these environment variables:

Variable	Purpose
`WOLFSSL_PATH`	Path to the wolfSSL build used by the examples
`WOLFCRYPT_ENABLE`	Set when wolfCrypt support is built in
`NO_FILESYSTEM`	Skip examples that need a filesystem
`ENABLE_DESTRUCTIVE_TESTS`	Also run tests that change TPM state, such as clearing it

!!! warning Destructive tests modify the TPM. Do not enable them on a TPM that holds keys or data you need.

30.2 CI workflows

The workflows live in `.github/workflows/`. The table lists the workflow name from each file.

File	Name
`_resolve-wolfssl.yml`	Resolve wolfSSL versions

File	Name
cmake-build.yml	WolfTPM CMake Build Tests
codeql.yml	CodeQL
codespell.yml	Codespell
coverity-scan-fixes.yml	Coverity Scan master branch
docs-site.yml	Build manual with documentation tooling
freestanding-build.yml	Freestanding Build (WOLFTPM_NO_STD_HEADERS)
fuzz.yml	Fuzz Testing
fwtpm-test.yml	fwTPM Tests
make-test-swtpm.yml	WolfTPM Build Tests
multi-compiler.yml	Multiple Compilers
nightly.yml	Nightly CI
pqc-build-matrix.yml	PQC Build Matrix (v1.85 trimming)
pqc-examples.yml	PQC Examples (v1.85)
publish-ci-image.yml	Publish wolfTPM CI image
publish-docs-image.yml	Publish documentation image
release-checks.yml	Release Checks
rust-test.yml	WolfTPM Rust Wrapper Tests
sanitizer.yml	Sanitizer Tests
sbom.yml	SBOM Test
seal-test.yml	Seal Test Suite
semgrep.yml	Semgrep
smoke-test.yml	Smoke Test
spdm-test.yml	SPDM Test
win-swtpm-test.yml	Windows swtpm Transport Test
win-test.yml	Windows Build Test
wolfhal-build.yml	wolfHAL Build Tests
wolfssl-versions-pqc.yml	wolfSSL Version Matrix
zephyr.yml	Zephyr wolfTPM Tests

30.3 See Also

- Benchmarks
- SBOM and Compliance
- Release Notes

31 Benchmarks

This page shows how fast the supported TPM 2.0 devices run common operations, measured with the `examples/bench/bench` program, plus the first post-quantum numbers from SEALSQ QVault silicon.

31.1 About these numbers

These are representative captures from real hardware, taken on different host boards and bus speeds. Results vary with the TPM firmware version, bus clock, host platform and build options, so treat them as a guide and run `./examples/bench/bench` on your own setup.

31.2 TPM 2.0 benchmarks by device

Average latency per operation, in milliseconds (lower is better). RSA-2048 key generation is a one-off provisioning cost.

Device	Bus	RSA-2048 key gen	RSA-2048 private	ECDSA P-256 sign	ECDSA P-256 verify
Infineon OPTIGA SLB9670	SPI, 43 MHz	2196.2	163.2	68.9	113.5
Infineon OPTIGA SLB9672	SPI, 43 MHz	1567.7	77.0	35.6	24.1
Infineon OPTIGA SLB9673	I2C, 400 kHz	1910.6	168.1	72.1	57.9
STMicro ST33KTPM2XSPI	SPI, 33 MHz	1944.1	90.8	25.3	36.5
STMicro ST33TPHF2XSPI	SPI, 33 MHz	7455.0	247.8	42.3	74.0
Microchip ATTPM20	SPI, 33 MHz	5275.9	117.7	58.7	43.0
Nations Z32H330	SPI, 33 MHz	2183.8	133.2	23.4	36.8
Nations NS350	SPI, 33 MHz	2378.9	51.7	16.8	21.9
Nuvoton NPCT650	not stated	4479.2	540.9	190.1	265.2
Nuvoton NPCT750	SPI, 43 MHz	3408.7	70.3	56.4	39.2
NVIDIA Jetson Orin fTPM (OP-TEE)	`/dev/tpmrm0`	736.4	11.9	45.1	31.7

The README captures for the ST33TPHF2XSPI RSA key generation ran a single operation, so that figure is the least reliable in the table.

Example output from `./examples/bench/bench` on an Infineon OPTIGA SLB9672 at 43 MHz:

```
./examples/bench/bench
TPM2 Benchmark using Wrapper API's
    Use Parameter Encryption: NULL
```

```
Loading SRK: Storage 0x81000200 (282 bytes)
RNG                 24 KB took 1.070 seconds,   22.429 KB/s
Benchmark symmetric AES-128-CBC-enc not supported!
Benchmark symmetric AES-128-CBC-dec not supported!
Benchmark symmetric AES-256-CBC-enc not supported!
Benchmark symmetric AES-256-CBC-dec not supported!
Benchmark symmetric AES-128-CTR-enc not supported!
Benchmark symmetric AES-128-CTR-dec not supported!
Benchmark symmetric AES-256-CTR-enc not supported!
Benchmark symmetric AES-256-CTR-dec not supported!
AES-128-CFB-enc     86 KB took 1.001 seconds,   85.890 KB/s
AES-128-CFB-dec     88 KB took 1.020 seconds,   86.267 KB/s
AES-256-CFB-enc     86 KB took 1.023 seconds,   84.073 KB/s
AES-256-CFB-dec     86 KB took 1.019 seconds,   84.370 KB/s
SHA1                88 KB took 1.021 seconds,   86.155 KB/s
SHA256              86 KB took 1.015 seconds,   84.717 KB/s
SHA384              90 KB took 1.007 seconds,   89.405 KB/s
RSA     2048 key gen       10 ops took 15.677 sec, avg 1567.678 ms, 0.638 ops/
   sec
RSA     2048 Public       110 ops took 1.000 sec, avg 9.095 ms, 109.951 ops/
   sec
RSA     2048 Private       14 ops took 1.078 sec, avg 76.996 ms, 12.988 ops/
   sec
RSA     2048 Pub  OAEP     51 ops took 1.012 sec, avg 19.838 ms, 50.408 ops/
   sec
RSA     2048 Priv OAEP     12 ops took 1.053 sec, avg 87.738 ms, 11.398 ops/
   sec
ECC      256 key gen        8 ops took 1.088 sec, avg 135.956 ms, 7.355 ops/
   sec
ECDSA    256 sign          29 ops took 1.033 sec, avg 35.621 ms, 28.073 ops/
   sec
ECDSA    256 verify        42 ops took 1.013 sec, avg 24.114 ms, 41.470 ops/
   sec
ECDHE    256 agree         16 ops took 1.055 sec, avg 65.948 ms, 15.164 ops/
   sec
```

Devices that do not support a mode print “not supported” for it.

31.3 Post-quantum (SEALSQ QVault)

These numbers were measured with `examples/bench/bench` on a Raspberry Pi 5 driving the SEALSQ QVault TPM over SPI. SEALSQ positions this part as the first post-quantum TPM in silicon.

Operation	Avg latency	Throughput
ML-DSA-65 key gen	2044.7 ms	0.49 ops/s
ML-DSA-65 sign	581.0 ms	1.72 ops/s
ML-DSA-65 verify	163.1 ms	6.13 ops/s
ML-KEM-768 key gen	800.8 ms	1.25 ops/s
ML-KEM-768 encapsulate	211.8 ms	4.72 ops/s
ML-KEM-768 decapsulate	425.5 ms	2.35 ops/s

Key generation is a one-off provisioning cost. The ECDSA figures above come from different TPMs, host boards, buses, and firmware, so they are not a like-for-like comparison with these ML-DSA numbers.

To compare ECDSA and PQC latencies, run `./examples/bench/bench` for both on the same TPM, host, bus, and build.

31.4 See Also

- Testing and CI
- Cited Sources
- Release Notes

COPYRIGHT ©2026 wolfSSL Inc.

32 SBOM and Compliance

wolfTPM can generate a Software Bill of Materials (SBOM) to support compliance with the EU Cyber Resilience Act (CRA). This page covers how to generate it and how the generator is organized.

32.1 SBOM and EU CRA compliance

wolfTPM generates an SBOM in CycloneDX 1.6 and SPDX 2.3 formats. The generator is the wolfssl-compliance-tooling snapshot vendored in `tools/sbom/` and pinned by `tools/sbom/.wolfssl-compliance-tooling-rev`. The SBOM records the configured build options (from `wolftpm/options.h`), hashes the built `libwolftpm` library artifact (shared or static; ELF, Mach-O, or PE), and lists wolfSSL as a dependency so vulnerability scanners can associate wolfSSL advisories with a wolfTPM deployment.

Output is reproducible. Set `SOURCE_DATE_EPOCH` (or build from a git checkout, which uses the last commit time) and repeated runs are byte-identical.

With autotools:

```
make sbom
```

This requires `python3` and `pyspdxtools` (`pip install spdx-tools`). The generator ships in the tree, so `make sbom` does not need a separate wolfSSL checkout. Pass `WOLFSSL_DIR=/path/to/wolfssl` when pkg-config cannot see the wolfSSL that this build linked, so the dependency version is read from `wolfssl/version.h`. `SBOM_WOLFSSL_VERSION` overrides that detection.

The CMake build exposes the same target. `WOLFSSL_DIR` is optional and has the same meaning:

```
cmake -B build .
cmake --build build --target sbom
```

The output files are:

- `wolftpm-<version>.cdx.json`
- `wolftpm-<version>.spdx.json`
- `wolftpm-<version>.spdx`

Optional overrides:

Variable	Purpose
`SBOM_LICENSE_OVERRIDE`	SPDX expression to use instead of the licence parsed from `COPYING` (for example `LicenseRef-wolfSSL-Commercial` for commercial licensees). Defaults to `GPL-3.0-or-later`, the per-file header licence.
`SBOM_LICENSE_TEXT`	Path to the licence text for any `LicenseRef-*` used in `SBOM_LICENSE_OVERRIDE` (required by SPDX 2.3).
`SBOM_WOLFSSL_VERSION`	Version recorded for the wolfSSL dependency. Auto-detected from `WOLFSSL_DIR/wolfssl/version.h` (or the wolfSSL `pkg-config` entry) when unset.

To install the generated files:

```
make install-sbom    # installs to $(datadir)/doc/wolftpm/
make uninstall-sbom
```

For further CRA guidance see wolfssl/doc/CRA.md.

32.2 Generator internals

The share/ set of wolfssl-compliance-tooling is the only vendorable part. The tools/wolfssl-compliance-tooling-sync script copies these files into a product at tools/sbom/, together with the pin files (VERSION and .wolfssl-compliance-tooling-rev). Copy the files, not the share/ folder name.

32.2.1 Contents

File	Role
sbom-driver.py	The product-neutral SBOM engine (Python).
sbom-driver	Thin shell wrapper that runs sbom-driver.py.
validate_sbom.py	Structural validator for CI (--name-prefix).
frontends/compdb_sbom.py	Extractor for any compile_commands.json.
frontends/iar_sbom.py	Extractor for an IAR Embedded Workbench .ewp.
frontends/zephyr_sbom.py	Extractor for a Zephyr module CMakeLists.txt.
build/sbom.mk	Shared plain-Make fragment and wolfssl_compliance_tooling_sbom_rule macro.
build/sbom.cmake	Shared CMake helper: wolfssl_compliance_tooling_add_sbom().
gen-sbom	The vendored SBOM generator.
sbom.am	Shared autotools fragment.

32.2.2 The driver contract

Every front end produces a composition input and a config input and hands them to the driver.

Composition (at least one):

- --srcs-file PATH: the source files compiled into the artifact (tier E).
- --lib PATH: the built library to hash (tier R/L/S).
- --no-artifact-hash: record the artifact as-built and do not re-hash. Use it with --lib for a FIPS canister or a kernel module. Never substitute a source list for a certified artifact.

Config (choose one):

- --cflags="...": raw CFLAGS. The driver expands the -D tokens through the host compiler. Use the = form so a leading-dash value is not read as a flag.
- --options-h PATH: a pre-expanded flat #define header, used verbatim.
- --user-settings PATH: a user_settings.h, which the generator captures.
- --source-only: no build-config macros (for example a Kconfig-driven build).

Dependency (for linkers and bindings): --dep-wolfssl, --dep-openssl, and --dep-version are passed through only when the generator supports them.

The driver captures macros with the host compiler, so the SBOM is reproducible across toolchains. It scrubs absolute host paths from the captured macros unless you pass --no-scrub.

The shared driver is product-neutral and calls the vendored `share/gen-sbom` by default. Pass `--gen-sbom` only when you want to override that copy.

32.2.3 The manifest contract

A product does not copy logic. It describes itself:

- **Make:** set `SBOM_NAME`, `SBOM_SRCS`, `SBOM_CFLAGS`, and a version (`SBOM_VERSION`, or `SBOM_VERSION_FILE` and `SBOM_VERSION_MACRO`), then `include tools/sbom/build/sbom.mk`. For a second target, instantiate `$(eval $(call wolfssl_compliance_tooling_sbom_rule,<targ` If the product configuration lives in a `user_settings.h`, also set `SBOM_SETTINGS_H` (and `SBOM_INCLUDE_DIRS` if that header needs paths the CFLAGS do not already carry). `SBOM_CFLAGS` alone records the literal `-D` set and nothing it derives, which for a gated header means the SBOM describes a configuration nobody built.
- **CMake:** `include(tools/sbom/build/sbom.cmake)` and call `wolfssl_compliance_tooling_add_sbom()` with `NAME`, `VERSION_FILE`, `VERSION_MACRO`, `TARGETS`, `DEFS`, `LICENSE`. `SBOM_GEN` is the canonical generator override. `GEN_SBOM` remains a legacy alias for compatibility.
- **Autotools:** set the `SBOM_*` variables and `include tools/sbom/sbom.am`.

Keep only true product knowledge in the product: the route-through script, the module extractor, and the HAL source selector.

32.3 See Also

- Testing and CI
- Release Notes
- API Reference

33 Release Notes

This page tracks recent wolfTPM releases. It reproduces the Unreleased section and the two most recent releases from the repository ChangeLog. The complete history, back to the first release, is in ChangeLog.md at the top of the repository.

33.1 Unreleased

- Added `wolfTPM2_AllocatePCRBanks` for changing which PCR banks a TPM allocates.
  - Added `examples/pcr/allocate` to report and re-provision the banks.
  - Fixed the fwTPM applying the allocation immediately instead of at the next `Startup(CLEAR)` per TPM 2.0 Part 3 22.5, accepting a selection that would leave it with no PCR banks, ignoring the `pcrSelect` bitmap, and reporting no allocated banks after a restart against an existing NV file.

33.2 wolfTPM Release 4.2.0 (Sep 14, 2026)

Summary

Feature and maintenance release centered on TCG TPM 2.0 v1.85 specification compliance in the firmware TPM (fwTPM), expanded post-quantum support, and new platform backends. Highlights: a broad set of fwTPM v1.85 compliance fixes (command attributes, PolicyAuthorize, context-blob authentication, NV authorization, ticket HMAC ordering), plus SPDM responder corrections; ML-DSA authentication for post-quantum TLS 1.3 and SealSQ QVault post-quantum TPM support; wolfHAL I2C/SPI backends and NVIDIA Jetson Orin OP-TEE fwTPM support; ST33 firmware-update corrections; transport and NV/hash performance improvements; and extensive security hardening (Coverity, static analysis, and negative tests).

Detail

- Firmware TPM (fwTPM) TCG v1.85 specification compliance
  - Command-code masking and vendor-bit return-code fix (PR #556)
  - Corrected auth-entry handling (PR #566)
  - PolicyAuthorize compliance fixes and corrected response codes for the keySign name ticket and approvedPolicy (PRs #567, #572)
  - Authenticate object context blobs on ContextSave/ContextLoad (PR #568)
  - Reject unsupported LoadExternal private types, fix creation-ticket HMAC ordering, and validate ML-DSA / ML-KEM templates in LoadExternal and CreateLoaded (PRs #573, #578)
  - Validate NV space authorization and report the v1.85 revision (PR #575)
  - Fix hierarchy and policy-authorization gaps and SPDM responder version negotiation (PR #577)
  - Correct command-attribute reporting and the verified-ticket HMAC algorithm (PR #579)
  - Additional TCG v1.85 fwTPM and SPDM compliance fixes (PR #584)
- Post-quantum and TLS
  - TPM-backed ML-DSA authentication for post-quantum TLS 1.3, with an example and tests (PR #559)
  - SealSQ QVault post-quantum TPM support (PR #570)
  - ML-KEM credential activation and ML-DSA quotes in the fwTPM (PR #592)
- New platform and HAL support
  - wolfHAL I2C and SPI backends, enabled with `--enable-wolfhal` and an application-supplied `board.h` (PR #562)
  - NVIDIA Jetson Orin (Tegra234) OP-TEE firmware TPM, reached through the Linux TPM kernel driver as `/dev/tpmrm0` (PR #576)
  - Finer per-command-group gating macros in the fwTPM (PR #574)
  - Caller-supplied policy authorization for firmware upgrade (PR #560)

- ST33 firmware update
  - Fix Generation 1 manifest size and refuse oversized commands (PR #583)
  - Select ST33 field-upgrade commands from the TPM command set (PR #586)
- Performance
  - Reuse transport connections and reduce NV-write and hash-cache overhead (PR #563)
- Security hardening (Coverity, static analysis, input validation)
  - Harden the crypto callback, ASN.1 parsing, parameter encryption, and marshalling, with negative tests (PR #551)
  - Bound the TPM2 response decrypt-parameter size, zeroize primary-key auth, and expand marshalling/import test coverage (PR #555)
  - Guard a tainted PCR-select copy in the fwTPM properties path (PR #558)
  - Fix a fwTPM response buffer overflow and an SPDM clear-frame command bypass (PR #561)
  - Harden wolfTPM2 PCR/hash wrapper validation (PR #554)
  - Strengthen TPM input validation and memory handling (PR #565)
  - Fix a wolfCrypt refcount race in P521 primary derivation and a policy-session authorization bypass (PR #571)
  - Harden PCR policy bounds checks (PR #581)
  - Harden wolfTPM validation and data handling (PR #582)
  - Harden fwTPM protocol handling and SPDM authentication (PR #588)
  - Make fwTPM state changes transactional and harden PolicyPCR and private-blob wrapping (PR #593)
  - Additional Coverity fixes across the TPM bounds and configuration paths, including bounded fwTPM child-blob copies, a guarded public-name buffer allocation, and restricted unsealed-output file permissions (PRs #591, #595, #603, #605)
  - Harden fwTPM key derivation and command validation (PR #596)
  - Fix TPM2 core key-import parsing and response handling and improve core zeroization and robustness (PRs #597, #600)
  - Harden error handling and secret zeroization in the examples, SPI transfer, and SPDM version parsing (PRs #598, #599, #604)
  - Reject truncated fwTPM Rewrap input, require bound authorization for policy sessions over empty-policy objects, bind the full keyed-hash secret into the public name, fail public-area parsing on overflow, normalize the TIS locality return, and clear residual request bytes from the shared command buffer (PR #608)
- Build fixes
  - Fix AES_BLOCK_SIZE undeclared under OPENSSL_COEXIST wolfSSL (PR #552)
  - Fix an edge-case build with TIS lock and no wolfCrypt (PR #564)
  - Fix the –enable-pqc build with –disable-wolfcrypt (PR #606)
  - Refresh the expired wolfSSL example CA certificates and add a refresh script (PR #601)
- Documentation and licensing
  - Add contribution guidance (CONTRIBUTING.md) (PR #569)
  - GPLv2 exception to the base GPLv3 license: wolfTPM combined with U-Boot from Cisco Systems, Inc. may be licensed under GPLv2 (PR #557)

33.3 wolfTPM Release 4.1.0 (Jul 10, 2026)

Summary

Feature release centered on TPM locality control and expanded post-quantum support. Highlights: runtime locality selection (`wolfTPM2_SetLocality`) with a corrected fwTPM per-PCR locality enforcement table and an optional GPIO nRST reset HAL; TPM 2.0 v1.85 post-quantum (ML-DSA / ML-KEM) support brought into the firmware TPM with fine-grained build macros; fwTPM Dictionary Attack hardening, transparent `TPM_RC_RETRY` handling, and SPDM secured transport for the fwTPM; FIPS 140-3 capability reporting; freestanding (no-libc) build support; SBOM (CycloneDX / SPDX) generation for EU Cyber Resilience Act (CRA) compliance; and extensive security hardening (Coverity, CodeQL).

Detail

- Runtime TPM locality control (PR #546)
  - New `wolfTPM2_SetLocality(dev, locality)` selects locality 0-4 at runtime, and `examples/pcr/reset` takes a `-loc=n` flag. Works on the built-in TIS/SPI driver (with release-and-retry for non-preempting chips) and the fwTPM over both socket and TIS/SHM; returns `NOT_COMPILED_IN` where locality is not selectable (I2C, Linux kernel driver, Windows TBS)
  - fwTPM per-PCR locality enforcement replaced with one source-of-truth table (TCG PC Client profile): corrects the reset map, adds the missing PCR-extend check, and reports proper `RESET_L*/EXTEND_L*/DRTM_RESET` bitmaps. Behavior change: DRTM PCRs 17-22 can no longer be extended from locality 0 (now `TPM_RC_LOCALITY`)
  - Fixed `TPM_CAP_ALGS/TPM_CAP_COMMANDS` paging to honor the property cursor so clients that follow `moreData` make progress
  - Optional hardware-reset HAL: `--enable-hal-reset[=LINE]` adds `TPM2_IoCb_Reset()`, pulsing the nRST line via the Linux GPIO char device (default ST33 GPIO24, Nuvoton GPIO4)
- TPM 2.0 v1.85 post-quantum (PQC) support in the fwTPM: ML-DSA sign/verify, ML-KEM encap/decap, and seed handling to TCG Phase B, with PQC CI and fuzz coverage (PR #445)
  - Fine-grained build macros to trim the footprint: `WOLFTPM_PQC` (the new lean `--enable-pqc`), per-algorithm `WOLFTPM_MLDSA`/`WOLFTPM_MLKEM`, and per-operation gates, plus `--enable-mldsa[=...]` / `--enable-mlkem[=...]` / `--disable-hash-mldsa` (PRs #527, #533)
  - wolfSSL v5.8.0+ PQC floor and upstream-drift CI; ML-DSA `TPM2_CreateLoaded` primary and PQC parameter encryption in the examples; new `_ex` session/OAEP/PQC-hash wrappers (PRs #501, #509, #531, #539, #520)
- fwTPM Dictionary Attack (DA) hardening to the TCG spec (PR #541): noDA honored on objects, `failedTries` persisted with the non-orderly-shutdown penalty, `recoveryTime`/`lockoutRecovery` self-heal, and DA properties reported via `TPM2_GetCapability`. Adds `wolfTPM2_DictionaryAttackLockRes` an `examples/management/da_check` example, and the `tests/fwtpm_da_retry.sh` harness
- Optional transparent `TPM_RC_RETRY` handling for TPMs that momentarily report busy; opt in via `TPM2_SetCommandRetries` or `-DWOLFTPM_MAX_RETRIES=N`, or compile out with `WOLFTPM_NO_RETRY` (PR #537)
- SPDM secured transport extended to the fwTPM (PR #510), and FIPS 140-3 capability reporting (PR #502)
- fwTPM session, policy, and NV fixes: transient state kept across command-port reconnects, `continueSession` set in the password-auth response, PolicyAuthorize zero-ticket handling, and an append-only NV journal for write-once flash ports (PRs #518, #530, #517, #540)
- New examples and options: crypto-primitive examples (getrandom, hash, AES, ECDH), a `WOLFTPM2_ECC_DEFAULT_CURVE` option (ZD 21780), and native_test ECC P-384 coverage (PRs #532, #519, #492)
- Nations NS350 example-suite fixes: RSA-4096 buffer sizing and SRK algorithm selection from the stored key type (PR #494)
- Freestanding build support: `WOLFTPM_NO_STD_HEADERS` keeps the standard C headers out of `tpm2_types.h` for bare-metal integrators, with a `freestanding-build.yml` CI job (PR #549)
- Software Bill of Materials (SBOM) generation for EU Cyber Resilience Act (CRA) compliance: new `make sbom` / `install-sbom` autotools targets and a CMake `sbom` target emit CycloneDX and SPDX documents for the built library, recording wolfSSL as a dependency (PR #536)
- Security hardening: automated security review (bounds/OOB fixes in the TPM2 packet parsers and marshaling, secret zeroization, policy/ticket bypass fixes), Coverity fixes across the fwTPM PCR/seed/hash/seal paths, CodeQL/Semgrep/Copilot review gates, and a heap out-of-bounds read fix in `TPM2_ASN_RsaUnpadPkcsv15` (PRs #496, #503, #511, #512, #518, #523, #535, #545, #547, #548, #543, #542, #544, #538, #513, #514, #524, #528, #507, #516)
- CI and build improvements: expanded CMake test cases, a GHCR container image, nightly fuzzing, wolfSSL latest-stable auto-resolve, and preflight smoke tests (PRs #495, #534, #522, #525, #508, #526, #521)
- Bug fixes

- Fixed the wolfCrypt crypto callback to propagate ALREADY_E for wolfSSL PR 10604 (PR #546)
- Ensured the wolfCrypt DRBG is used with crypto callbacks and made TPM2_StirRandom a TCG-compliant no-op on HW-RNG-backed TPMs (PRs #498, #493)
- Added a note on using the TPM RNG for the StartAuth session nonce (ZD 21476, PR #478)

33.4 Full History

Older releases are not repeated here. See ChangeLog.md in the repository root for every release.

33.5 See Also

- Testing and CI
- SBOM and Compliance
- API Reference

COPYRIGHT ©2026 wolfSSL Inc.

34 Cited Sources

This page lists the references used while writing this manual: the two works cited in the original introduction, followed by the specifications the manual refers to.

34.1 References

1. Wikipedia contributors. (2018, May 30). Trusted Platform Module. In *Wikipedia, The Free Encyclopedia*. Retrieved 22:46, June 20, 2018.
2. Arthur W., Challener D., Goldman K. (2015). Platform Configuration Registers. In: *A Practical Guide to TPM 2.0*. Apress, Berkeley, CA.

34.2 Specifications

Specification	Body	Where it applies
TPM 2.0 Library Specification, versions 1.38, 1.59, 1.84 and 1.85	Trusted Computing Group (TCG)	The TPM 2.0 command set, structures and behavior that wolfTPM and the fwTPM implement. Version 1.85 adds the post-quantum commands.
FIPS 203, Module-Lattice-Based Key-Encapsulation Mechanism Standard (ML-KEM)	NIST	Post-quantum key encapsulation (ML-KEM-768).
FIPS 204, Module-Lattice-Based Digital Signature Standard (ML-DSA)	NIST	Post-quantum signatures (ML-DSA-65).
DSP0274, Security Protocol and Data Model (SPDM) Specification	DMTF	The SPDM secured transport used with supported TPM modules and the fwTPM.

34.3 See Also

- Benchmarks
- Release Notes
- API Reference

COPYRIGHT ©2026 wolfSSL Inc.