wolfTPM Documentation

2025-04-30

Contents

1 Intro

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.

1.1 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.

According to Wikipedia, a TPM provides the following[1] :

- 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).

In addition, TPM can also be used for various applications such as platform integrity, disk encryption, password protection, and software license protection.

1.2 Hierarchies

Platform: **TPM_RH_PLATFORM**

Owner: **PM_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.

1.3 Platform Configuration Registers (PCRs)

Platform Configuration Registers (PCRs) are one of the essential features of a TPM. Their prime use case is to provide a method to cryptographically record (measure) software state: both the software running on a platform and configuration data used by that software.[2]

wolfTPM contains hash digests for SHA-1 and SHA-256 with an index 0-23. These hash digests can be extended to prove the integrity of a boot sequence (secure boot).

1.4 Terminology

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

1.5 Hardware

Tested with:

- Infineon OPTIGA (TM) Trusted Platform Module 2.0 SLB 9670.
  - LetsTrust: [http://letstrust.de] (https://buyzero.de/collections/andere-platinen/products/letstrust-hardware-tpm-trusted-platform-module). Compact Raspberry Pi TPM 2.0 board based on Infineon SLB 9670.
- ST ST33TP* TPM 2.0 module (SPI and I2C)
- Microchip ATTPM20 module
- Nuvoton NPCT65X or NPCT75x TPM2.0 module

1.5.1 Device Identification

Infineon SLB9670: TIS: 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

ST ST33TP SPI TPM2: Caps 0x1a7e2882, Did 0x0000, Vid 0x104a, Rid 0x4e Mfg STM (2), Vendor , Fw 74.8 (1151341959), FIPS 140-2 1, CC-EAL4 0

ST ST33TP I2C 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. TPM 2.0 module Mfg NTZ (0), Vendor Z32H330, Fw 7.51 (419631892), FIPS 140-2 0, 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

2 Building wolfTPM

To build the wolfTPM library, it's required to first build and install the wolfSSL library. This can be downloaded from the downloads page, or through a "git clone" command, shown below:

```
$ git clone https://github.com/wolfssl/wolfssl
```

Once the wolfSSL library has been downloaded, it needs to be built with the following option being passed to the configure script:

```
$ ./configure --enable-wolftpm
```

Or equivalently, with the following options:

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

Then the wolfSSL library just needs to be built and installed however the user prefers.

The next step is to download and install the wolfTPM library. wolfTPM can similarly be downloaded from the downloads page or be cloned from GitHub. The following commands show how to clone and install wolfTPM:

```
$ git clone https://github.com/wolfssl/wolftpm
$ cd wolftpm
$ ./autogen.sh
$ ./configure
$ make
```

2.0.1 Build options and defines

```
--enable-debug          Add debug code/turns off optimizations (yes|no|verbose
    |io)
                        - DEBUG_WOLFTPM, WOLFTPM_DEBUG_VERBOSE,
                           WOLFTPM_DEBUG_IO
--enable-examples       Enable Examples (default: enabled)
--enable-wrapper        Enable wrapper code (default: enabled) -
    WOLFTPM2_NO_WRAPPER
--enable-wolfcrypt      Enable wolfCrypt hooks for RNG, Auth Sessions and
    Parameter encryption
                        (default: enabled) - WOLFTPM2_NO_WOLFCRYPT
--enable-advio          Enable Advanced IO (default: disabled) -
    WOLFTPM_ADV_IO
--enable-i2c            Enable I2C TPM Support (default: disabled, requires
    advio) - WOLFTPM_I2C
--enable-checkwaitstate Enable TIS / SPI Check Wait State support (default:
    depends on chip)
                        - WOLFTPM_CHECK_WAIT_STATE
--enable-smallstack     Enable options to reduce stack usage
--enable-tislock        Enable Linux Named Semaphore for locking access to SPI
     device for
                        concurrent access between processes - WOLFTPM_TIS_LOCK

--enable-autodetect     Enable Runtime Module Detection (default: enable -
    when no module
                        specified) - WOLFTPM_AUTODETECT
```

```
--enable-infineon       Enable Infineon SLB9670 TPM Support (default: disabled
    )
--enable-st             Enable ST ST33TPM Support (default: disabled) -
    WOLFTPM_ST33
--enable-microchip      Enable Microchip ATTPM20 Support (default: disabled) -
     WOLFTPM_MCHP
--enable-nuvoton        Enable Nuvoton NPCT65x/NPCT75x Support (default:
    disabled)
                        - WOLFTPM_NUVOTON

--enable-devtpm         Enable using Linux kernel driver for /dev/tpmX (
    default: disabled)
                        - WOLFTPM_LINUX_DEV
--enable-swtpm          Enable using SWTPM TCP protocol. For use with
    simulator.
                        (default: disabled) - WOLFTPM_SWTPM
--enable-winapi         Use Windows TBS API. (default: disabled) -
    WOLFTPM_WINAPI

WOLFTPM_USE_SYMMETRIC   Enables symmetric AES/Hashing/HMAC support for TLS
    examples.
WOLFTPM2_USE_SW_ECDHE   Disables use of TPM for ECC ephemeral key generation
    and shared secret
                        for TLS examples.
TLS_BENCH_MODE          Enables TLS benchmarking mode.
NO_TPM_BENCH            Disables the TPM benchmarking example.
```

2.0.2 Building Infineon SLB9670

Build wolfTPM:

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

2.0.3 Building ST ST33TP*

Build wolfTPM:

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

For the I2C support on Raspberry Pi you may need to enable I2C. Here are the steps: 1. Edit `sudo vim /boot/config.txt` 2. Uncomment `dtparam=i2c_arm=on` 3. Reboot `sudo reboot`

2.0.4 Building Microchip ATTPM20

Build wolfTPM:

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

2.0.5 Building Nuvoton

Build wolfTPM:

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

2.0.6 Building for "/dev/tpmX"

This build option allows you to talk to any TPM vendor supported by the Linux TIS kernel driver

Build wolfTPM:

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

Note: When using a TPM device through the Linux kernel driver make sure sufficient permissions are given to the application that uses wolfTPM, because the "/dev/tpmX" typically has read-write permissions only for the "tss" user group. Either run wolfTPM examples and your application using sudo or add your user to the "tss" group like this:

```
sudo adduser yourusername tss
```

2.0.6.1 With QEMU and swtpm This demonstrates using wolfTPM in QEMU to communicate using the linux kernel device "/dev/tpmX". You will need to install or build swtpm. Below are a short method to build. 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 ..
```

You can setup a basic linux installation. Other installation bases can be used. This step will take 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 it's ready to start the qemu 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
```

To build checkout and build wolfTPM, in the QEMU terminal

```
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 the examples such as `sudo ./examples/wrap/wrap` within QEMU. Using sudo maybe required for access to `/dev/tpm0`.

2.0.7 Building for SWTPM

wolfTPM is to be able to interface with SW TPM interfaces defined by section D.3 of TPM-Rev-2.0-Part-4-Supporting-Routines-01.38-code

Using the socket connection for SWTPM is exclusive and not compatible with TIS or devtpm.

Only a subset of functionality is implemented to support testing of wolfTPM. The platform requests are not used by wolfTPM.

Two implementations were used in testing:

- https://sourceforge.net/projects/ibmswtpm2/files/
- https://github.com/stefanberger/swtpm

To enable this functionality, build wolfTPM as shown below:

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

2.0.7.1 SWTPM simulator setup

2.0.7.1.1 **ibmswtpm2** Checkout and Build

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

Running:

```
./tpm_server --rm
```

The rm switch is optional and remove the cache file NVChip. Alternately you can `rm NVChip`

2.0.7.1.2 **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)
```

Note: On Mac OS X had to 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
```

Running 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


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

2.0.8 Building for Windows TBS API

2.0.7.2 Running examples wolfTPM can be built to use Windows native TBS (TPM Base Services)

When using the Windows TBS interface the NV access is blocked by default. TPM NV storage space is very limited and when filled can cause undefined behaviors, such as failures loading key handles. These are 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 the 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 even encryption/decryption.

For primary keys created with `TPM2_CreatePrimary` you get back a handle. There is no encrypted private data returned. That handle will remain 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 anytime using `TPM2_Load`.

2.0.8.1 Limitations wolfTPM has been tested on Windows 10 with TPM 2.0 devices. While Windows does support TPM 1.2, functionality is limited and not supported by wolfTPM.

Presence of 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
```

2.0.8.2 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
```

2.0.8.3 Building on linux Tested using mingw-w32-bin_x86_64-linux_20131221.tar.bz2 source

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 ..
```

2.0.8.4 Running on Windows

To confirm presence and status of TPM on the machine run `tpm.msc`

3 Getting Started

The wolfTPM library has TPM 2.0 wrapper tests, native tests, and a sample benchmark application that come ready-to-use after a successful installation of wolfTPM. Below are some instructions on how to run the sample applications yourself.

To interface with the hardware platform that is running these applications, please see the function `TPM2_IoCb` inside of `tpm_io.c`.

3.1 Examples

These examples demonstrate features of a TPM 2.0 module.

The examples create RSA and ECC keys in NV for testing using handles defined in `./examples/tpm_test.h`.

The PKCS #7 and TLS examples require generating CSR's and signing them using a test script. See CSR and Certificate Signing below.

To enable parameter encryption use `-aes` for AES-CFB mode or `-xor` for XOR mode. Only some TPM commands / responses support parameter encryption. If the TPM2_ API has .flags `CMD_FLAG_ENC2` or `CMD_FLAG_DEC2` set then the command will use parameter encryption / decryption.

There are some vendor specific examples, like the TPM 2.0 extra GPIO examples for ST33 and NPCT75x.

3.1.1 Native API Test

Demonstrates calling native TPM2_* API's.

```
./examples/native/native_test
TPM2 Demo using Native API's
TPM2: Caps 0x30000495, Did 0x0000, Vid 0x104a, Rid 0x4e
TPM2_Startup pass
TPM2_SelfTest pass
TPM2_GetTestResult: Size 12, Rc 0x0
TPM2_IncrementalSelfTest: Rc 0x0, Alg 0x1 (Todo 0)
TPM2_GetCapability: Property FamilyIndicator 0x322e3000
TPM2_GetCapability: Property PCR Count 24
TPM2_GetCapability: Property FIRMWARE_VERSION_1 0x004a0008
TPM2_GetCapability: Property FIRMWARE_VERSION_2 0x44a01587
TPM2_GetRandom: Got 32 bytes
TPM2_StirRandom: success
TPM2_PCR_Read: Index 0, Count 1
TPM2_PCR_Read: Index 0, Digest Sz 32, Update Counter 20
TPM2_PCR_Read: Index 1, Count 1
TPM2_PCR_Read: Index 1, Digest Sz 32, Update Counter 20
TPM2_PCR_Read: Index 2, Count 1
TPM2_PCR_Read: Index 2, Digest Sz 32, Update Counter 20
TPM2_PCR_Read: Index 3, Count 1
TPM2_PCR_Read: Index 3, Digest Sz 32, Update Counter 20
TPM2_PCR_Read: Index 4, Count 1
TPM2_PCR_Read: Index 4, Digest Sz 32, Update Counter 20
TPM2_PCR_Read: Index 5, Count 1
TPM2_PCR_Read: Index 5, Digest Sz 32, Update Counter 20
TPM2_PCR_Read: Index 6, Count 1
TPM2_PCR_Read: Index 6, Digest Sz 32, Update Counter 20
```

```
TPM2_PCR_Read: Index 7, Count 1
TPM2_PCR_Read: Index 7, Digest Sz 32, Update Counter 20
TPM2_PCR_Read: Index 8, Count 1
TPM2_PCR_Read: Index 8, Digest Sz 32, Update Counter 20
TPM2_PCR_Read: Index 9, Count 1
TPM2_PCR_Read: Index 9, Digest Sz 32, Update Counter 20
TPM2_PCR_Read: Index 10, Count 1
TPM2_PCR_Read: Index 10, Digest Sz 32, Update Counter 20
TPM2_PCR_Read: Index 11, Count 1
TPM2_PCR_Read: Index 11, Digest Sz 32, Update Counter 20
TPM2_PCR_Read: Index 12, Count 1
TPM2_PCR_Read: Index 12, Digest Sz 32, Update Counter 20
TPM2_PCR_Read: Index 13, Count 1
TPM2_PCR_Read: Index 13, Digest Sz 32, Update Counter 20
TPM2_PCR_Read: Index 14, Count 1
TPM2_PCR_Read: Index 14, Digest Sz 32, Update Counter 20
TPM2_PCR_Read: Index 15, Count 1
TPM2_PCR_Read: Index 15, Digest Sz 32, Update Counter 20
TPM2_PCR_Read: Index 16, Count 1
TPM2_PCR_Read: Index 16, Digest Sz 32, Update Counter 20
TPM2_PCR_Read: Index 17, Count 1
TPM2_PCR_Read: Index 17, Digest Sz 32, Update Counter 20
TPM2_PCR_Read: Index 18, Count 1
TPM2_PCR_Read: Index 18, Digest Sz 32, Update Counter 20
TPM2_PCR_Read: Index 19, Count 1
TPM2_PCR_Read: Index 19, Digest Sz 32, Update Counter 20
TPM2_PCR_Read: Index 20, Count 1
TPM2_PCR_Read: Index 20, Digest Sz 32, Update Counter 20
TPM2_PCR_Read: Index 21, Count 1
TPM2_PCR_Read: Index 21, Digest Sz 32, Update Counter 20
TPM2_PCR_Read: Index 22, Count 1
TPM2_PCR_Read: Index 22, Digest Sz 32, Update Counter 20
TPM2_PCR_Read: Index 23, Count 1
TPM2_PCR_Read: Index 23, Digest Sz 32, Update Counter 20
TPM2_PCR_Extend success
TPM2_PCR_Read: Index 0, Count 1
TPM2_PCR_Read: Index 0, Digest Sz 32, Update Counter 21
TPM2_StartAuthSession: sessionHandle 0x3000000
TPM2_PolicyGetDigest: size 32
TPM2_PCR_Read: Index 0, Digest Sz 20, Update Counter 21
wc_Hash of PCR[0]: size 32
TPM2_PolicyPCR failed 0x1c4: TPM_RC_AUTHSIZE
TPM2_PolicyRestart: Done
TPM2_HashSequenceStart: sequenceHandle 0x80000000
Hash SHA256 test success
TPM2_CreatePrimary: Endorsement 0x80000000 (314 bytes)
TPM2_CreatePrimary: Storage 0x80000002 (282 bytes)
TPM2_LoadExternal: 0x80000004
TPM2_MakeCredential: credentialBlob 68, secret 256
TPM2_ReadPublic Handle 0x80000004: pub 314, name 34, qualifiedName 34
Create HMAC-SHA256 Key success, public 48, Private 137
TPM2_Load New HMAC Key Handle 0x80000004
TPM2_PolicyCommandCode: success
TPM2_ObjectChangeAuth: private 137
```

```
TPM2_ECC_Parameters: CurveID 3, sz 256, p 32, a 32, b 32, gX 32, gY 32, n 32,
    h 1
TPM2_Create: New ECDSA Key: pub 88, priv 126
TPM2_Load ECDSA Key Handle 0x80000004
TPM2_Sign: ECC S 32, R 32
TPM2_VerifySignature: Tag 32802
TPM2_Create: New ECDH Key: pub 88, priv 126
TPM2_Load ECDH Key Handle 0x80000004
TPM2_ECDH_KeyGen: zPt 68, pubPt 68
TPM2_ECDH_ZGen: zPt 68
TPM2 ECC Shared Secret Pass
TPM2_Create: New RSA Key: pub 278, priv 222
TPM2_Load RSA Key Handle 0x80000004
TPM2_RSA_Encrypt: 256
TPM2_RSA_Decrypt: 68
RSA Encrypt/Decrypt test passed
TPM2_NV_DefineSpace: 0x1bffffff
TPM2_NV_ReadPublic: Sz 14, Idx 0x1bffffff, nameAlg 11, Attr 0x2020002, authPol
    0, dataSz 32, name 34
Create AES128 CFB Key success, public 50, Private 142
TPM2_Load New AES Key Handle 0x80000004
Encrypt/Decrypt test success
```

3.1.2 Wrapper API Test

Demonstrates calling the wolfTPM2_* wrapper API's.

```
./examples/wrap/wrap_test
TPM2 Demo for Wrapper API's
Mfg STM  (2), Vendor , Fw 74.8 (1151341959), FIPS 140-2 1, CC-EAL4 0
RSA Encrypt/Decrypt Test Passed
RSA Encrypt/Decrypt OAEP Test Passed
RSA Key 0x80000000 Exported to wolf RsaKey
wolf RsaKey loaded into TPM: Handle 0x80000000
RSA Private Key Loaded into TPM: Handle 0x80000000
ECC Sign/Verify Passed
ECC DH Test Passed
ECC Verify Test Passed
ECC Key 0x80000000 Exported to wolf ecc_key
wolf ecc_key loaded into TPM: Handle 0x80000000
ECC Private Key Loaded into TPM: Handle 0x80000000
NV Test on index 0x1800200 with 1024 bytes passed
Hash SHA256 test success
HMAC SHA256 test success
Encrypt/Decrypt (known key) test success
Encrypt/Decrypt test success
```

3.1.3 Attestation Use Cases

3.1.3.1 TPM signed timestamp, TPM2.0 GetTime Demonstrates creation of Attestation Identity Keys (AIK) and the generation of TPM signed timestamp that can be later used as protected report of the current system uptime.

This example demonstrates the use of `authSession` (authorization Session) and `policySession` (Policy authorization) to enable the Endorsement Hierarchy necessary for creating AIK. The AIK is used to issue a `TPM2_GetTime` command using the TPM 2.0 native API. This provides a TPM generated and signed timestamp that can be used as a system report of its uptime.

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

3.1.3.2 TPM signed PCR(system) measurement, TPM2.0 Quote Demonstrates the generation of TPM2.0 Quote used for attestation of the system state by putting PCR value(s) in a TPM signed structure.

3.1.3.2.1 List of examples The `./examples/pcr/` folder contains tools for working with Platform Configuration Registers (PCR). It is recommended to build wolfTPM with debug output enabled using `./configure --enable-debug` before `make` to see more logging output. There are example scripts to show using these PCR examples.

Examples:

- `./examples/pcr/reset`: Used to clear the content of a PCR (restrictions apply, see below)
- `./examples/pcr/extend`: Used to modify the content of a PCR (extend is a cryptographic operation, see below)
- `./examples/pcr/quote`: Used to generate a TPM2.0 Quote structure containing the PCR digest and TPM-generated signature

Scripts:

- `./examples/pcr/demo.sh` - script demonstrating the tools above
- `./examples/pcr/demo-quote-zip.sh` - script demonstrating how using the tools above a system file can be measured and a TPM-signed proof with that measurement generated

3.1.3.2.2 Technology introduction Platform Configuration Registers (PCR)

PCRs in TPM2.0 are special registers that allow only one type of write operations to be performed on them. A TPM 2.0 extend operation is the only way to update a PCR.

At power-up, the TPM resets all PCRs to their default state (all zeros or all ones, depending on the PCR). From this state, the TPM can generate the same PCR value only if the PCR is extended with the same hash digest. In case of multiple values(multiple extend operations), the values must be supplied in the correct order, otherwise the final PCR value would differ.

For example, doing a measured boot under Linux would generate the same PCR digest, if the kernel is the same at every boot. However, loading the same (A) Linux kernel, (B) initrd image and (C) configuration file would generate the same PCR digest only when the order of extend operations is consistent (for example, A-B-C). It does not matter which extend operation is first or last as long as the order is kept the same. For example, C-B-A would result in a reproducible digest, but it would differ from the A-B-C digest.

Reset

Not all PCRs are equal. The user can perform `extend` operation on all PCRs, but the user can `reset` only on one of them during normal runtime. This is what makes PCRs so useful.

- PCR0-15 are reset at boot and can be cleared again(reset) only from reboot cycle.
- PCR16 is a PCR for debug purposes. This is the PCR used by all tools above by default. It is safe to test and work with PCR16.
- PCR17-22 are reserved for Dynamic Root of Trust Measurement (DRTM), an advanced topic that is to be covered separately.

Extend

The TPM 2.0 TPM2_Extend API uses a SHA1 or SHA256 cryptographic operation to combine the current value of the PCR and with newly provided hash digest.

Quote

The TPM 2.0 TPM2_Quote API is a standard operation that encapsulates the PCR digest in a TCG defined structure called TPMS_ATTEST together with TPM signature. The signature is produced from a TPM generated key called Attestation Identity Key (AIK) that only the TPM can use. This provides guarantee for the source of the Quote and PCR digest. Together, the Quote and PCR provide the means for system measurement and integrity.

3.1.3.2.3 Example Usage

Reset Example Usage

```
$ ./examples/pcr/reset -?
PCR index is out of range (0-23)
Expected usage:
./examples/pcr/reset [pcr]
* pcr is a PCR index between 0-23 (default 16)
Demo usage without parameters, resets PCR16.
```

Extend Example Usage

```
$ ./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.
```

Quote Example Usage

```
$ ./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.
```

3.1.3.2.4 Typical demo output

All PCR examples can be used without arguments. This is the output of the `./examples/pcr/demo.sh` script:

```
$ ./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 | ................
```

As expected, the PCR16 content is now set back to all zeroes. From this moment on we can generate predictable PCR digests(values) for system measurement. Similar to using PCR7 after boot, because PCR7 is reset at system boot. Using PCR16 allows us to skip system reboots and test safely.

```
$ ./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
```

Based on the old content of the PCR (all zeros) and the provided hash (SHA256 32-byte digest), the PCR gets its new value printed at the end of the `extend` example. This value will always be the same, if `reset` is launched before `extend`. To pass custom hash digest, the `extend` tool accepts PCR index as first argument(recommended to use 16 for PCR16) and user file as second argument.

```
$ ./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......
    6e 2d 9f 9e 5a 86 4f 11 fe 13 84 94 cf 05 b9 d5 | n-..Z.O.........
    eb 5a 34 39 b2 a5 7a 5f 52 c0 f4 e7 2b 70 b7 62 | .Z49..z_R...+p.b
    6a fe 79 4e 2e 46 2e 43 d7 1c ef 2c 14 21 11 14 | j.yN.F.C...,.!..
    95 01 93 a9 85 0d 02 c7 b2 f8 75 1a bd 59 da 56 | ..........u..Y.V
    cc 43 e3 d2 aa 14 49 2a 59 26 09 9e c9 4b 1a 66 | .C....I*Y&...K.f
    cb 77 65 95 79 69 89 bd 46 46 13 3d 2c a9 78 f8 | .we.yi..FF.=,.x.
    2c ab 8a 4a 6b f2 97 67 86 37 f8 f6 9d 85 cd cf | ,..Jk..g.7......
    a4 ae c6 d3 cf c1 63 92 8c 7b 88 79 90 54 0a ba | ......c..{.y.T..
    8d c6 1c 8f 6e 6d 61 bc a9 2f 35 b0 1a 46 74 9a | ....nma../5..Ft.
    e3 7d 39 33 52 1a f5 4b 07 8d 30 53 75 b5 68 40 | .}93R..K..0Su.h@
    04 e7 a1 fc b1 93 5d 1e bc ca f4 a9 fa 75 d3 f6 | ......]......u..
    3d 4a 5b 07 23 0e f0 f4 1f 97 23 76 1a ee 66 93 | =J[.#.....#v..f.
    cd fd 9e 6f 2b d3 95 c5 51 cf f6 81 5b 97 a1 d2 | ...o+...Q...[...
    06 45 c0 30 70 ad bd 36 66 9f 95 af 60 7c d5 a2 | .E.0p..6f...`|..
```

Before producing a TPM-signed structure containing the PCR measurement, the quote example starts by creating an Endorsement Key(EK) that is required for the TPM to operate. It serves essentially as

the primary key for all other keys. Next, a Storage Key(SRK) is generated and under that SRK a special Attestation Identity Key(AIK) is added. Using the AIK the TPM can sign the quote structure.

3.1.3.2.5 Steps for measuring a system file (performing local attestation) A system administrator wants to make sure the zip tool of an user is genuine (legitimate software, correct version and has not been tampered with). To do this, the SysAdmin resets PCR16 and can afterwards generate a PCR digest based on the zip binary that can be used for future references if the file has been modified.

This is the output from `./examples/pcr/demo-quote-zip.sh` script.

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

This is a good known initial state of the PCR. By using the `extend` tool the SysAdmin feeds the `/usr/bin/zip` binary to wolfCrypt for SHA256 hash computation, which then is used by wolfTPM to issue a `TPM2_Extend` operation in 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..
```

Once the extend operation is finished, the SysAdmin wants to create a TPM2.0 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 result of the TPM2.0 Quote operation is saved in the `zip.quote` binary file. The `TPMS_ATTEST` structure of TPM 2.0 Quote contains also useful clock and time information. For more about the TPM time attestation please check the `./examples/timestamp/signed_timestamp` example.

3.1.3.3 Remote Attestation challenge Demonstrates how to create Remote Attestation challenge using the TPM 2.0 and afterwards prepare a response.

3.1.3.3.1 List of examples The `./examples/attestation/` folder contains examples related specifically to remote attestation. However, the demonstration requires the creation of TPM 2.0 keys using the keygen example also included in wolfTPM's source code.

Complete list of the required examples is shown below:

- `./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/keygen/keygen`: Used to create a primary key(PK) and attestation key(AK)

Note: All of these example allow the use of the Endorsement Key and Attestation Key under the Endorsement Hierarchy. This is done by adding the `-eh` option when executing any of the three examples

above. The advantage of using EK/EH is that the private key material of the EK never leaves the TPM. Anything encrypted using the public part of the EK can be encrypted only internally by the TPM owner of the EK, and EK is unique for every TPM chip. Therefore, creating challenges for Remote Attestation using the EK/EH has greater value in some scenarios. One drawback is that by using the EK the identity of the host under attestation is always known, because the EK private-public key pair identifies the TPM and in some scenarios this might rise privacy concerns. Our remote attestation examples support both AK under SRK and AK under EK. It is up to the developer to decide which one to use.

3.1.3.3.2 Technology introduction Remote Attestation is the process of a client providing an evidence to an attestation server that verifies if the client is in a known state.

For this process to take place, the client and server must establish initial trust. This is achieved using the standard TPM 2.0 commands MakeCredential and ActivateCredential.

1. The client transfers the public parts of a TPM 2.0 Primary Attestation Key(PAK) and quote signing Attestation Key(AK).
2. MakeCredential uses the public part of the PAK to encrypt a challenge(secret). Typically, the challenge is a digest of the public part of an Attestation Key(AK).

This way the challenge can only be decrypted by the TPM that can load the private part of the PAK and AK. Because the PAK and AK are bound to the TPM using a fixedTPM key attribute, the only TPM that can load these keys is the TPM where they were originally created.

3. After the challenge is created, it is transferred from the server to the client.
4. ActivateCredential uses the TPM 2.0 loaded PAK and AK to decrypt the challenge and retrieve the secret. Once retrieved, the client can respond to the server challenge.

This way the client confirms to the server it possesses the expected TPM 2.0 System Identity and Attestation Key.

Note:

- The transport protocol to exchange the challenge and response are up to the developer to choose, because this is implementation specific. One approach could be the use of TLS1.3 client-server connection using wolfSSL.

3.1.3.3.3 Example usage Creating TPM 2.0 keys for Remote Attestation

Using the keygen example we can create the necessary TPM 2.0 Attestation Key and TPM 2.0 Primary Storage Key that will be used as a 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
```

Make Credential Example Usage

Using the make_credential example an attestation server can generate remote attestation challenge. The secret is 32 bytes of randomly generated seed that could be used for a symmetric key in some remote 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 attestation server is not part of the make_credential example, because the exchange is implementation specific.

Activate Credential Example Usage

Using the activate_credential example a client can decrypt the remote attestation challenge. The secret will be exposed in plain and can be exchanged with 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
```

The transfer of the challenge response containing the secret in plain (or used as a symmetric key seed) is not part of the activate_credential example, because the exchange is also implementation specific.

3.1.4 Parameter Encryption

3.1.4.1 Key generation with encrypted authorization Detailed information can be found under the section "Key generation"

3.1.4.2 Secure vault for keys with encrypted NVRAM authorization Detailed information can be found in this file under section "Storing keys into the TPM's NVRAM"

3.1.4.3 TPM2.0 Quote with encrypted user data Example for demonstrating how to use parameter encryption to protect the user data between the Host and the TPM.

In this example the qualifying data that can be supplied by the user for a Quote operation is protected. Qualifying data is arbitrary data incorporated into the signed Quote structure. Using parameter encryption, wolfTPM enables the Host to transfer that user data in encrypted form to the TPM and vice versa. Thus, protecting the data from man-in-the-middle attacks.

Only the first parameter of a TPM command can be encrypted and the parameter must be of type TPM2B_DATA. For example, the password auth of a TPM key or the qualifying data of a TPM2.0 Quote.

The encryption of command request and response can be performed together or separate. There can be a communication exchange between the TPM and a client program where only the parameter in the request command is encrypted.

This behavior depends on the sessionAttributes:

- TPMA_SESSION_encrypt for command request
- TPMA_SESSION_decrypt for command response

Either one can be set separately or both can be set in one authorization session. This is up to the user (developer).

./examples/pcr/quote_paramenc

3.1.5 CSR

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

3.1.6 Certificate Signing

External script for generating test certificates based on TPM generated CSR's. Typically 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

3.1.7 PKCS #7

Example signs and verifies data with PKCS #7 using a TPM based key.

- Must first run:

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

The result is displayed to stdout on the console.

3.1.8 TLS Examples

The TLS example uses TPM based ECDHE (ECC Ephemeral key) support. It can be disabled using CFLAGS="-DWOLFTPM2_USE_SW_ECDHE" or #define WOLFTPM2_USE_SW_ECDHE. We are also looking into using the 2-phase TPM2_EC_Ephemeral and TPM2_ZGen_2Phase methods for improved performance and scalability.

To force ECC use with wolfSSL when RSA is enabled define TLS_USE_ECC.

To use symmetric AES/Hashing/HMAC with the TPM define WOLFTPM_USE_SYMMETRIC.

Generation of 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 wolfSSL certs directory.
   a. cp ./certs/ca-ecc-cert.pem ../wolfssl/certs/tpm-ca-ecc-cert.pem
   b. cp ./certs/ca-rsa-cert.pem ../wolfssl/certs/tpm-ca-rsa-cert.pem

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

```
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
```

3.1.8.1 TLS Client Examples show using a TPM key and certificate for TLS mutual authentication (client authentication).

This example client connects to localhost on on port 11111 by default. These can be overridden using TLS_HOST and TLS_PORT.

You can validate using the wolfSSL example server this like: ./examples/server/server -b -p 11111 -g -d -i -V

To validate client certificate use the following wolfSSL example server command: ./examples/server/server -b -p 11111 -g -A ./certs/tpm-ca-rsa-cert.pem -i -V or ./examples/server/server -b -p 11111 -g -A ./certs/tpm-ca-ecc-cert.pem -i -V

Then run the wolfTPM TLS client example: ./examples/tls/tls_client -rsa or ./examples/tls/tls_client -ecc

3.1.8.2 TLS Server This example shows using a TPM key and certificate for a TLS server.

By default it listens on port 11111 and can be overridden at build-time using the TLS_PORT macro.

Run the wolfTPM TLS server example: ./examples/tls/tls_server -rsa or ./examples/tls/tls_server -ecc

Then run the wolfSSL example client this like: ./examples/client/client -h localhost -p 11111 -g -d

To validate server certificate use the following wolfSSL example client comment: ./examples/client/client -h localhost -p 11111 -g -A ./certs/tpm-ca-rsa-cert.pem or ./examples/client/client -h localhost -p 11111 -g -A ./certs/tpm-ca-ecc-cert.pem

Or using your browser: https://localhost:11111

With browsers you will get certificate warnings until you load the test CA's ./certs/ca-rsa-cert.pem and ./certs/ca-ecc-cert.pem into your OS key store. For testing most browsers have a way to continue to the site anyways to bypass the warning.

3.1.9 Clock

Updating the TPM clock

The TPM has internal hardware clock that can be useful to the user. There are two values that the TPM can provide in respect to time.

TPM time is the current uptime, since the last power on sequence. This value can not be changed or modified. There is no mechanism for that. The value is reset at every power sequence.

TPM clock is the total time the TPM has ever been powered. This value can be modified using the TPM2_ClockSet command. The TPM clock can be set only forward.

This way the user can keep track of relative and current time using the TPM clock.

Note: If the new time value makes a change bigger than the TPM clock update interval, then the TPM will first update its volatile register for time and then the non-volatile register for time. This may cause a narrow delay before the commands returns execution to the user. Depending on the TPM manufacturer, the delay can vary from us to few ms.

Note: This example can take an optional argument, the time value in milliseconds used for incrementing the TPM clock. Default value is 50000ms (50 seconds).

```
./examples/timestamp/clock_set
```

3.1.10 Key Generation

Examples for generating a TPM key blob and storing to disk, then loading from disk and loading into 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

./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
```

When filename is not supplied, a default filename “keyblob.bin” is used, therefore `keyload` and `keygen` can be used without additional parameters for quick TPM 2.0 key generation demonstration.

To see the complete list of supported cryptographic algorithms and options by the keygen example, use one of the `--help` switches.

Example for importing a private key as TPM key blob and storing to disk, then loading from disk and loading into temporary TPM handle.

```
$ ./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
```

The `keyload` tool takes only one argument, the filename of the stored key. Because the information what is key scheme (RSA or ECC) is contained within the key blob.

3.1.11 Storing keys into the TPM’s NVRAM

These examples demonstrates how to use the TPM as a secure vault for keys. There are two programs, one to store a TPM key into the TPM’s NVRAM and another to extract the key from the TPM’s NVRAM. Both examples can use parameter encryption to protect from MITM attacks. The Non-volatile memory 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 there is a keyblob.bin generated using the keygen tool. The key can be of any type, RSA, ECC or symmetric. The example will store the private and public part. In case of a symmetric key the public part is meta data from the TPM. How to generate a key you can see above, in the description of the keygen example.

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 will try to load the extracted key, if both the public and private part of the key were stored in NVRAM. The “-aes” switches triggers the use of parameter encryption.

The examples can work with partial key material - private or public. This is achieved by using the “-priv” and “-pub” options.

Typical output of storing only the private key of RSA asymmetric key pair in NVRAM and without parameter encryption enabled.

```
$ ./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 successful key extraction using “read”, the NV Index is destroyed. Therefore, to use “read” again, the “store” example must be run again as well.

3.1.12 Seal / Unseal

TPM 2.0 can protect secrets using a standard Seal/Unseal procedure. 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.

There are two examples available: `seal/seal` and `seal/unseal`.

Demo usage is available, without parameters.

3.1.12.1 Sealing data into a TPM 2.0 Key Using the `seal` example we store securely our data in a newly generated TPM 2.0 key. Only when this key is loaded into the TPM, we could read back our secret data.

Please find 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`.

3.1.13 GPIO Control

Some TPM 2.0 modules have extra I/O functionalities and additional GPIO that the developer could use. This extra GPIO could be used to signal other subsystems about security events or system states.

Currently, the GPIO control examples support ST33 and NPCT75x TPM 2.0 modules.

There are four examples available. Configuration using `gpio/gpio_config`.

Every example has a help option `-h`. Please consult with `gpio_config -h` about the various GPIO modes.

Once configured, a GPIO can be controlled using `gpio/gpio_set` and `gpio/gpio_read`.

Demo usage is available, when no parameters are supplied. Recommended is to use carefully selected options, because GPIO interact with the physical world.

3.1.13.1 GPIO Config ST33 supports 6 modes, information from `gpio/gpio_config` below:

```
$ ./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.
```

Example usage for configuring a GPIO to output can be found below:

```
$ ./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
```

Example usage for configuring a GPIO as input with a pull-up on ST33 can be found below:

```
$ ./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
```

3.1.13.2 GPIO Config (NPCT75xx) NPCT75x supports 3 output modes (no input modes), information from `gpio/gpio_config` below:

```
$ ./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.
```

Please note that NPCT75x GPIO numbering starts from GPIO3, while ST33 starts from GPIO0.

```
$ ./examples/gpio/gpio_nuvoton 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
```

3.1.13.3 GPIO Usage Switching a GPIO configuration is seamless. * For ST33 `gpio/gpio_config` takes care of deleting existing NV Index, so a new GPIO configuration can be chosen. * For NPCT75xx `gpio/gpio_config` can reconfigure any GPIO without deleting the created NV index.

```
$ ./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
```

3.2 Benchmarks

The wolfTPM benchmark application requires the same setup as the example applications.

Note: Key Generation is using existing template from hierarchy seed.

Run on Infineon OPTIGA SLB9670 at 43MHz:

```
./examples/bench/bench
TPM2 Benchmark using Wrapper API's
RNG                 16 KB took 1.140 seconds,   14.033 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!
Benchmark symmetric AES-256-CFB-enc not supported!
Benchmark symmetric AES-256-CFB-dec not supported!
SHA1               138 KB took 1.009 seconds,  136.783 KB/s
SHA256             138 KB took 1.009 seconds,  136.763 KB/s
RSA     2048 key gen        5 ops took 10.981 sec, avg 2196.230 ms, 0.455 ops/
   sec
RSA     2048 Public       113 ops took 1.005 sec, avg 8.893 ms,   112.449 ops/
   sec
RSA     2048 Private        7 ops took 1.142 sec, avg 163.207 ms,   6.127 ops/
   sec
RSA     2048 Pub  OAEP     73 ops took 1.011 sec, avg 13.848 ms,   72.211 ops/
   sec
RSA     2048 Priv OAEP      6 ops took 1.004 sec, avg 167.399 ms,   5.974 ops/
   sec
ECC      256 key gen        5 ops took 1.157 sec, avg 231.350 ms,   4.322 ops/
   sec
ECDSA    256 sign          15 ops took 1.033 sec, avg 68.865 ms,   14.521 ops/
   sec
ECDSA    256 verify         9 ops took 1.022 sec, avg 113.539 ms,   8.808 ops/
   sec
ECDHE    256 agree          5 ops took 1.161 sec, avg 232.144 ms,   4.308 ops/
   sec
```

Run on ST ST33TP SPI at 33MHz:

```
./examples/bench/bench
TPM2 Benchmark using Wrapper API's
RNG                 14 KB took 1.017 seconds,   13.763 KB/s
AES-128-CBC-enc     40 KB took 1.008 seconds,   39.666 KB/s
AES-128-CBC-dec     42 KB took 1.032 seconds,   40.711 KB/s
AES-256-CBC-enc     40 KB took 1.013 seconds,   39.496 KB/s
AES-256-CBC-dec     40 KB took 1.011 seconds,   39.563 KB/s
AES-128-CTR-enc     26 KB took 1.055 seconds,   24.646 KB/s
AES-128-CTR-dec     26 KB took 1.035 seconds,   25.117 KB/s
AES-256-CTR-enc     26 KB took 1.028 seconds,   25.302 KB/s
AES-256-CTR-dec     26 KB took 1.030 seconds,   25.252 KB/s
AES-128-CFB-enc     42 KB took 1.045 seconds,   40.201 KB/s
AES-128-CFB-dec     40 KB took 1.008 seconds,   39.699 KB/s
AES-256-CFB-enc     40 KB took 1.022 seconds,   39.151 KB/s
AES-256-CFB-dec     42 KB took 1.041 seconds,   40.362 KB/s
SHA1                86 KB took 1.005 seconds,   85.559 KB/s
SHA256              84 KB took 1.019 seconds,   82.467 KB/s
RSA     2048 key gen        1 ops took 7.455 sec, avg 7455.036 ms, 0.134 ops/
   sec
RSA     2048 Public       110 ops took 1.003 sec, avg 9.122 ms,  109.624 ops/
   sec
RSA     2048 Private        5 ops took 1.239 sec, avg 247.752 ms,  4.036 ops/
   sec
RSA     2048 Pub  OAEP     81 ops took 1.001 sec, avg 12.364 ms,  80.880 ops/
```

```
   sec
RSA     2048 Priv OAEP      4 ops took 1.007 sec, avg 251.780 ms,  3.972 ops/
   sec
ECC      256 key gen        5 ops took 1.099 sec, avg 219.770 ms,  4.550 ops/
   sec
ECDSA    256 sign          24 ops took 1.016 sec, avg 42.338 ms,  23.619 ops/
   sec
ECDSA    256 verify        14 ops took 1.036 sec, avg 74.026 ms,  13.509 ops/
   sec
ECDHE    256 agree          5 ops took 1.235 sec, avg 247.085 ms,  4.047 ops/
   sec
```

Run on Microchip ATTPM20 at 33MHz:

```
./examples/bench/bench
TPM2 Benchmark using Wrapper API's
RNG                  2 KB took 1.867 seconds,    1.071 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     16 KB took 1.112 seconds,   14.383 KB/s
AES-128-CFB-dec     16 KB took 1.129 seconds,   14.166 KB/s
AES-256-CFB-enc     12 KB took 1.013 seconds,   11.845 KB/s
AES-256-CFB-dec     12 KB took 1.008 seconds,   11.909 KB/s
SHA1                22 KB took 1.009 seconds,   21.797 KB/s
SHA256              22 KB took 1.034 seconds,   21.270 KB/s
RSA     2048 key gen        3 ops took 15.828 sec, avg 5275.861 ms, 0.190 ops/
   sec
RSA     2048 Public        22 ops took 1.034 sec, avg 47.021 ms, 21.267 ops/
   sec
RSA     2048 Private        9 ops took 1.059 sec, avg 117.677 ms, 8.498 ops/
   sec
RSA     2048 Pub  OAEP     21 ops took 1.007 sec, avg 47.959 ms, 20.851 ops/
   sec
RSA     2048 Priv OAEP      9 ops took 1.066 sec, avg 118.423 ms, 8.444 ops/
   sec
ECC      256 key gen        7 ops took 1.072 sec, avg 153.140 ms, 6.530 ops/
   sec
ECDSA    256 sign          18 ops took 1.056 sec, avg 58.674 ms, 17.043 ops/
   sec
ECDSA    256 verify        24 ops took 1.031 sec, avg 42.970 ms, 23.272 ops/
   sec
ECDHE    256 agree         16 ops took 1.023 sec, avg 63.934 ms, 15.641 ops/
   sec
```

Run on Nations Technologies Inc. TPM 2.0 module at 33MHz:

```
./examples/bench/bench
TPM2 Benchmark using Wrapper API's
RNG                 12 KB took 1.065 seconds,   11.270 KB/s
```

```
AES-128-CBC-enc     48 KB took 1.026 seconds,   46.780 KB/s
AES-128-CBC-dec     48 KB took 1.039 seconds,   46.212 KB/s
AES-256-CBC-enc     48 KB took 1.035 seconds,   46.370 KB/s
AES-256-CBC-dec     48 KB took 1.025 seconds,   46.852 KB/s
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     50 KB took 1.029 seconds,   48.591 KB/s
AES-128-CFB-dec     50 KB took 1.035 seconds,   48.294 KB/s
AES-256-CFB-enc     48 KB took 1.000 seconds,   47.982 KB/s
AES-256-CFB-dec     48 KB took 1.003 seconds,   47.855 KB/s
SHA1                80 KB took 1.009 seconds,   79.248 KB/s
SHA256              80 KB took 1.004 seconds,   79.702 KB/s
SHA384              78 KB took 1.018 seconds,   76.639 KB/s
RSA     2048 key gen        8 ops took 17.471 sec, avg 2183.823 ms, 0.458 ops/
   sec
RSA     2048 Public        52 ops took 1.004 sec, avg 19.303 ms, 51.805 ops/
   sec
RSA     2048 Private        8 ops took 1.066 sec, avg 133.243 ms, 7.505 ops/
   sec
RSA     2048 Pub  OAEP     51 ops took 1.001 sec, avg 19.621 ms, 50.966 ops/
   sec
RSA     2048 Priv OAEP      8 ops took 1.073 sec, avg 134.182 ms, 7.453 ops/
   sec
ECC      256 key gen       20 ops took 1.037 sec, avg 51.871 ms, 19.279 ops/
   sec
ECDSA    256 sign          43 ops took 1.006 sec, avg 23.399 ms, 42.736 ops/
   sec
ECDSA    256 verify        28 ops took 1.030 sec, avg 36.785 ms, 27.185 ops/
   sec
ECDHE    256 agree         26 ops took 1.010 sec, avg 38.847 ms, 25.742 ops/
   sec
```

Run on Nuvoton NPCT650:

```
./examples/bench/bench
TPM2 Benchmark using Wrapper API's
RNG                  8 KB took 1.291 seconds,    6.197 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-256-CTR-enc not supported!
Benchmark symmetric AES-256-CTR-dec not supported!
Benchmark symmetric AES-256-CFB-enc not supported!
Benchmark symmetric AES-256-CFB-dec not supported!
SHA1                90 KB took 1.005 seconds,   89.530 KB/s
SHA256              90 KB took 1.010 seconds,   89.139 KB/s
RSA     2048 key gen        8 ops took 35.833 sec, avg 4479.152 ms, 0.223 ops/
   sec
RSA     2048 Public        77 ops took 1.007 sec, avg 13.078 ms, 76.463 ops/
   sec
RSA     2048 Private        2 ops took 1.082 sec, avg 540.926 ms, 1.849 ops/
   sec
```

```
RSA     2048 Pub  OAEP     53 ops took 1.005 sec, avg 18.961 ms, 52.739 ops/
   sec
RSA     2048 Priv OAEP      2 ops took 1.088 sec, avg 544.075 ms, 1.838 ops/
   sec
ECC      256 key gen        7 ops took 1.033 sec, avg 147.608 ms, 6.775 ops/
   sec
ECDSA    256 sign           6 ops took 1.141 sec, avg 190.149 ms, 5.259 ops/
   sec
ECDSA    256 verify         4 ops took 1.061 sec, avg 265.216 ms, 3.771 ops/
   sec
ECDHE    256 agree          6 ops took 1.055 sec, avg 175.915 ms, 5.685 ops/
   sec
```

Run on Nuvoton NPCT750 at 43MHz:

```
RNG                 16 KB took 1.114 seconds,   14.368 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!
SHA1               120 KB took 1.012 seconds,  118.618 KB/s
SHA256             122 KB took 1.012 seconds,  120.551 KB/s
SHA384             120 KB took 1.003 seconds,  119.608 KB/s
RSA     2048 key gen        5 ops took 17.043 sec, avg 3408.678 ms, 0.293 ops/
   sec
RSA     2048 Public       134 ops took 1.004 sec, avg 7.490 ms, 133.517 ops/
   sec
RSA     2048 Private       15 ops took 1.054 sec, avg 70.261 ms, 14.233 ops/
   sec
RSA     2048 Pub  OAEP    116 ops took 1.002 sec, avg 8.636 ms, 115.797 ops/
   sec
RSA     2048 Priv OAEP     15 ops took 1.061 sec, avg 70.716 ms, 14.141 ops/
   sec
ECC      256 key gen       12 ops took 1.008 sec, avg 84.020 ms, 11.902 ops/
   sec
ECDSA    256 sign          18 ops took 1.015 sec, avg 56.399 ms, 17.731 ops/
   sec
ECDSA    256 verify        26 ops took 1.018 sec, avg 39.164 ms, 25.533 ops/
   sec
ECDHE    256 agree         35 ops took 1.029 sec, avg 29.402 ms, 34.011 ops/
   sec
```

4 wolfTPM Library Design

4.1 Library Headers

wolfTPM header files are located in the following locations:

wolfTPM : wolftpm/

wolfSSL : wolfssl/

wolfCrypt : wolfssl/wolfcrypt

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

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

4.2 Example Design

Every example application that is included with wolfTPM includes the tpm_io.h header file, located in wolfTPM/examples. 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.

5 API Reference

5.1 TPM2 Proprietary

More...

5.1.1 Functions

	Name
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_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_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 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 int	**TPM2_GetCurveSize curveID)Determine the size in bytes of any 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.
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.

5.1.2 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.

5.1.3 Functions Documentation

```
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_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

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 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 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 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

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_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};
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_PrintAuthCmd(&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);
```

5.2 wolftpm/tpm2.h

5.1.3.30 function TPM2_PrintPublicArea

5.2.1 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**

	Name
struct	**TPM2B_MAX_NV_BUFFER**
struct	**TPM2B_IV**
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	**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	**TPMS_ACT_DATA**
struct	**TPML_ACT_DATA**
union	**TPMU_CAPABILITIES**
struct	**TPMS_CAPABILITY_DATA**
struct	**TPMS_CLOCK_INFO**
struct	**TPMS_TIME_INFO**
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**
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**

	Name
struct	**TPMS_SENSITIVE_CREATE**
struct	**TPM2B_SENSITIVE_CREATE**
struct	**TPMS_SCHEME_HASH**
struct	**TPMS_SCHEME_ECDAA**
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**
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**
struct	**TPMS_RSA_PARMS**
struct	**TPMS_ECC_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**

	Name
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	**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**
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**

	Name
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**
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	**SetCommandCodeAuditStatus_In**
struct	**PCR_Event_In**
struct	**PCR_Event_Out**
struct	**PCR_Allocate_In**

	Name
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**
struct	**PolicyOR_In**
struct	**PolicyPCR_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**

	Name
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**
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**

5.2.2 Types

	Name
enum	**TPM_ALG_ID_T** { TPM_ALG_ERROR = 0x0000, TPM_ALG_RSA = 0x0001, 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_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_CTR = 0x0040, TPM_ALG_OFB = 0x0041, TPM_ALG_CBC = 0x0042, TPM_ALG_CFB = 0x0043, TPM_ALG_ECB = 0x0044}
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}

	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, 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,

	Name
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, 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

	Name
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}
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_CREATION = 0x8021, TPM_ST_VERIFIED = 0x8022, TPM_ST_AUTH_SECRET = 0x8023, TPM_ST_HASHCHECK = 0x8024, TPM_ST_AUTH_SIGNED = 0x8025, 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_LAST = TPM_CAP_ACT, TPM_CAP_VENDOR_PROPERTY = 0x00000100}

	Name
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, 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, 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

	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 = 0x00000010, TPMA_ALGORITHM_encrypting = 0x00000020, TPMA_ALGORITHM_method = 0x00000040}
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_derivedDataOrigin = 0x00000200, TPMA_OBJECT_noDA = 0x00000400, TPMA_OBJECT_encryptedDuplication = 0x00000800, TPMA_OBJECT_restricted = 0x00010000, TPMA_OBJECT_decrypt = 0x00020000, TPMA_OBJECT_sign = 0x00040000}
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}
enum	**TPMA_LOCALITY_mask** { TPM_LOC_ZERO = 0x01, TPM_LOC_ONE = 0x02, TPM_LOC_TWO = 0x04, TPM_LOC_THREE = 0x08, TPM_LOC_FOUR = 0x10}

	Name
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_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}
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}

	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}

	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}
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 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**
typedef UINT32	**TPM_CAP**
typedef UINT32	**TPM_PT**
typedef UINT32	**TPM_PT_PCR**

	Name
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 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_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**	
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**	

Name
typedef struct TPM2B_EVENT**
typedef struct TPM2B_MAX_BUFFER**
typedef struct TPM2B_MAX_NV_BUFFER**
typedef TPM2B_DIGEST**
typedef struct TPM2B_IV**
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 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 UINT32 **TPMA_ACT**
typedef struct TPMS_ACT_DATA**
typedef struct TPML_ACT_DATA**
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 TPM_ST**
typedef union TPMU_ATTEST**
typedef struct TPMS_ATTEST**
typedef struct TPM2B_ATTEST**
typedef TPM_KEY_BITS**
typedef union TPMU_SYM_KEY_BITS**

Name
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 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**
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**

Name
typedef TPMS_SIGNATURE_RSA**
typedef TPMS_SIGNATURE_RSA**
typedef struct TPMS_SIGNATURE_ECC**
typedef TPMS_SIGNATURE_ECC**
typedef TPMS_SIGNATURE_ECC**
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 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**
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**

Name
typedef GetRandom_In**
typedef UINT32 **TPMI_GPIO_NAME**
typedef UINT32 **TPMI_GPIO_MODE**
typedef struct TPMS_GPIO_CONFIG**
typedef struct TPML_GPIO_CONFIG**

5.2.3 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**(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)
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)

Name
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)
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**(SetCommandCodeAuditStatus_In * in)
WOLFTPM_API TPM_RC**(PCR_Event_In * out)
WOLFTPM_API TPM_RC**(PCR_Allocate_In * out)
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)

Name
WOLFTPM_API TPM_RC**(PolicyOR_In * in)
WOLFTPM_API TPM_RC**(PolicyPCR_In * in)
WOLFTPM_API TPM_RC**(PolicyLocality_In * in)
WOLFTPM_API TPM_RC**(PolicyNV_In * in)
WOLFTPM_API TPM_RC**(PolicyCounterTimer_In * in)
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)

	Name
WOLFTPM_API TPM_RC**(FieldUpgradeStart_In * in)	
WOLFTPM_API TPM_RC**(FieldUpgradeData_In * out)	
WOLFTPM_API TPM_RC**(FirmwareRead_In * out)	
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_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 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_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.
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.

	Name
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 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 int	**TPM2_GetCurveSize curveID)Determine the size in bytes of any 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.
WOLFTPM_LOCAL void	**TPM2_ForceZero**(void * mem, word32 len)
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.

5.2.4 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**

	Name
const BYTE[]	**TPM_20_IAK_POLICY_SHA384**
const BYTE[]	**TPM_20_IDEVID_POLICY_SHA512**
const BYTE[]	**TPM_20_IAK_POLICY_SHA512**

5.2.5 Types Documentation

5.2.5.1 enum TPM_ALG_ID_T

Enumerator	Value	Description
TPM_ALG_ERROR	0x0000	
TPM_ALG_RSA	0x0001	
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_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_CTR	0x0040	
TPM_ALG_OFB	0x0041	
TPM_ALG_CBC	0x0042	
TPM_ALG_CFB	0x0043	
TPM_ALG_ECB	0x0044	

5.2.5.2 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	

5.2.5.3 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	
TPM_CC_FieldUpgradeData	0x00000141	
TPM_CC_IncrementalSelfTest	0x00000142	
TPM_CC_SelfTest	0x00000143	

Enumerator	Value	Description
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	

Enumerator	Value	Description
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_LAST	TPM_CC_EncryptDecrypt2	
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_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	

5.2.5.4 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	

Enumerator	Value	Description
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	

Enumerator	Value	Description
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	
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	

Enumerator	Value	Description
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	

5.2.5.5 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	
TPM_CLOCK_FINE_FASTER	1	
TPM_CLOCK_MEDIUM_FASTER	2	
TPM_CLOCK_COARSE_FASTER	3	

5.2.5.6 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	

5.2.5.7 enum TPM_ST_T

Enumerator	Value	Description
TPM_ST_RSP_COMMAND	0x00C4	
TPM_ST_NULL	0X8000	

Enumerator	Value	Description
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_CREATION	0x8021	
TPM_ST_VERIFIED	0x8022	
TPM_ST_AUTH_SECRET	0x8023	
TPM_ST_HASHCHECK	0x8024	
TPM_ST_AUTH_SIGNED	0x8025	
TPM_ST_FU_MANIFEST	0x8029	

5.2.5.8 enum TPM_SE_T

Enumerator	Value	Description
TPM_SE_HMAC	0x00	
TPM_SE_POLICY	0x01	
TPM_SE_TRIAL	0x03	

5.2.5.9 enum TPM_SU_T

Enumerator	Value	Description
TPM_SU_CLEAR	0x0000	
TPM_SU_STATE	0x0001	

5.2.5.10 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_LAST	TPM_CAP_ACT	
TPM_CAP_VENDOR_PROPERTY	0x00000100	

5.2.5.11 enum TPM_PT_T

Enumerator	Value	Description
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	
PT_VAR	PT_GROUP * 2	

Enumerator	Value	Description
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	

5.2.5.12 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	

5.2.5.13 enum TPM_PS_T

Enumerator	Value	Description
TPM_PS_MAIN	0x00000000	
TPM_PS_PC	0x00000001	

Enumerator	Value	Description
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	

5.2.5.14 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	

5.2.5.15 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	

Enumerator	Value	Description
TPM_RH_LAST	TPM_RH_AUTH_FF	

5.2.5.16 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	0x00000010	
TPMA_ALGORITHM_encrypting	0x00000020	
TPMA_ALGORITHM_method	0x00000040	

5.2.5.17 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_derivedDataOrigin	0x00000200	
TPMA_OBJECT_noDA	0x00000400	
TPMA_OBJECT_encryptedDuplication	0x00000800	
TPMA_OBJECT_restricted	0x00010000	
TPMA_OBJECT_decrypt	0x00020000	
TPMA_OBJECT_sign	0x00040000	

5.2.5.18 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	

5.2.5.19 enum TPMA_LOCALITY_mask

Enumerator	Value	Description
TPM_LOC_ZERO	0x01	
TPM_LOC_ONE	0x02	
TPM_LOC_TWO	0x04	

Enumerator	Value	Description
TPM_LOC_THREE	0x08	
TPM_LOC_FOUR	0x10	

5.2.5.20 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	

5.2.5.21 enum TPMA_MEMORY_mask

Enumerator	Value	Description
TPMA_MEMORY_sharedRAM	0x00000001	
TPMA_MEMORY_sharedNV	0x00000002	
TPMA_MEMORY_objectCopiedToRam	0x00000004	

5.2.5.22 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	

5.2.5.23 enum TPMA_ACT_T

Enumerator	Value	Description
TPMA_ACT_signaled	0x00000001	
TPMA_ACT_preserveSignaled	0x00000002	

5.2.5.24 enum TPM_NT

Enumerator	Value	Description
TPM_NT_ORDINARY	0x0	
TPM_NT_COUNTER	0x1	
TPM_NT_BITS	0x2	

Enumerator	Value	Description
TPM_NT_EXTEND	0x4	
TPM_NT_PIN_FAIL	0x8	
TPM_NT_PIN_PASS	0x9	

5.2.5.25 enum TPM_MODE_Vendor_Mask

Enumerator	Value	Description
TPMLib_2	0x01	
TPMFips	0x02	
TPMLowPowerOff	0x00	
TPMLowPowerByRegister	0x04	
TPMLowPowerByGpio	0x08	
TPMLowPowerAuto	0x0C	

5.2.5.26 enum TPMI_GPIO_NAME_T

Enumerator	Value	Description
TPM_GPIO_PP	0x00000000	
TPM_GPIO_LP	0x00000001	
TPM_GPIO_C	0x00000002	
TPM_GPIO_D	0x00000003	

5.2.5.27 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	

5.2.5.28 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	

5.2.5.29 enum TPM_Vendor_t

Enumerator	Value	Description
TPM_VENDOR_UNKNOWN	0	
TPM_VENDOR_INFINEON	0x15d1	
TPM_VENDOR_STM	0x104a	
TPM_VENDOR_MCHP	0x1114	
TPM_VENDOR_NUVOTON	0x1050	
TPM_VENDOR_NATIONTECH	0x1B4E	

5.2.5.30 typedef TPM_MODIFIER_INDICATOR

```
typedef UINT32 TPM_MODIFIER_INDICATOR;
```

5.2.5.31 typedef TPM_AUTHORIZATION_SIZE

```
typedef UINT32 TPM_AUTHORIZATION_SIZE;
```

5.2.5.32 typedef TPM_PARAMETER_SIZE

```
typedef UINT32 TPM_PARAMETER_SIZE;
```

5.2.5.33 typedef TPM_KEY_SIZE

```
typedef UINT16 TPM_KEY_SIZE;
```

5.2.5.34 typedef TPM_KEY_BITS

```
typedef UINT16 TPM_KEY_BITS;
```

5.2.5.35 typedef TPM_GENERATED

```
typedef UINT32 TPM_GENERATED;
```

5.2.5.36 typedef TPM_ALG_ID

```
typedef UINT16 TPM_ALG_ID;
```

5.2.5.37 typedef TPM_ECC_CURVE

```
typedef UINT16 TPM_ECC_CURVE;
```

5.2.5.38 typedef TPM_CC

```
typedef UINT32 TPM_CC;
```

5.2.5.39 typedef TPM_RC

```
typedef INT32 TPM_RC;
```

5.2.5.40 typedef TPM_CLOCK_ADJUST

```
typedef UINT8 TPM_CLOCK_ADJUST;
```

5.2.5.41 typedef TPM_EO

```
typedef UINT16 TPM_EO;
```

5.2.5.42 typedef TPM_ST

```
typedef UINT16 TPM_ST;
```

5.2.5.43 typedef TPM_SE

```
typedef UINT8 TPM_SE;
```

5.2.5.44 typedef TPM_SU

```
typedef UINT16 TPM_SU;
```

5.2.5.45 typedef TPM_CAP

```
typedef UINT32 TPM_CAP;
```

5.2.5.46 typedef TPM_PT

```
typedef UINT32 TPM_PT;
```

5.2.5.47 typedef TPM_PT_PCR

```
typedef UINT32 TPM_PT_PCR;
```

5.2.5.48 typedef TPM_PS

```
typedef UINT32 TPM_PS;
```

5.2.5.49 typedef TPM_HANDLE

```
typedef UINT32 TPM_HANDLE;
```

5.2.5.50 typedef TPM_HT

```
typedef UINT8 TPM_HT;
```

5.2.5.51 typedef TPM_RH

```
typedef UINT32 TPM_RH;
```

5.2.5.52 typedef TPM_HC

```
typedef UINT32 TPM_HC;
```

5.2.5.53 typedef TPMA_ALGORITHM

```
typedef UINT32 TPMA_ALGORITHM;
```

5.2.5.54 typedef TPMA_OBJECT

```
typedef UINT32 TPMA_OBJECT;
```

5.2.5.55 typedef TPMA_SESSION

```
typedef BYTE TPMA_SESSION;
```

5.2.5.56 typedef TPMA_LOCALITY

```
typedef BYTE TPMA_LOCALITY;
```

5.2.5.57 typedef TPMA_PERMANENT

```
typedef UINT32 TPMA_PERMANENT;
```

5.2.5.58 typedef TPMA_STARTUP_CLEAR

```
typedef UINT32 TPMA_STARTUP_CLEAR;
```

5.2.5.59 typedef TPMA_MEMORY

```
typedef UINT32 TPMA_MEMORY;
```

5.2.5.60 typedef TPMA_CC

```
typedef UINT32 TPMA_CC;
```

5.2.5.61 typedef TPMI_YES_NO

```
typedef BYTE TPMI_YES_NO;
```

5.2.5.62 typedef TPMI_DH_OBJECT

```
typedef TPM_HANDLE TPMI_DH_OBJECT;
```

5.2.5.63 typedef TPMI_DH_PARENT

```
typedef TPM_HANDLE TPMI_DH_PARENT;
```

5.2.5.64 typedef TPMI_DH_PERSISTENT

```
typedef TPM_HANDLE TPMI_DH_PERSISTENT;
```

5.2.5.65 typedef TPMI_DH_ENTITY

```
typedef TPM_HANDLE TPMI_DH_ENTITY;
```

5.2.5.66 typedef TPMI_DH_PCR

```
typedef TPM_HANDLE TPMI_DH_PCR;
```

5.2.5.67 typedef TPMI_SH_AUTH_SESSION

```
typedef TPM_HANDLE TPMI_SH_AUTH_SESSION;
```

5.2.5.68 typedef TPMI_SH_HMAC

```
typedef TPM_HANDLE TPMI_SH_HMAC;
```

5.2.5.69 typedef TPMI_SH_POLICY

```
typedef TPM_HANDLE TPMI_SH_POLICY;
```

5.2.5.70 typedef TPMI_DH_CONTEXT

```
typedef TPM_HANDLE TPMI_DH_CONTEXT;
```

5.2.5.71 typedef TPMI_RH_HIERARCHY

```
typedef TPM_HANDLE TPMI_RH_HIERARCHY;
```

5.2.5.72 typedef TPMI_RH_ENABLES

```
typedef TPM_HANDLE TPMI_RH_ENABLES;
```

5.2.5.73 typedef TPMI_RH_HIERARCHY_AUTH

```
typedef TPM_HANDLE TPMI_RH_HIERARCHY_AUTH;
```

5.2.5.74 typedef TPMI_RH_PLATFORM

```
typedef TPM_HANDLE TPMI_RH_PLATFORM;
```

5.2.5.75 typedef TPMI_RH_OWNER

```
typedef TPM_HANDLE TPMI_RH_OWNER;
```

5.2.5.76 typedef TPMI_RH_ENDORSEMENT

```
typedef TPM_HANDLE TPMI_RH_ENDORSEMENT;
```

5.2.5.77 typedef TPMI_RH_PROVISION

```
typedef TPM_HANDLE TPMI_RH_PROVISION;
```

5.2.5.78 typedef TPMI_RH_CLEAR

```
typedef TPM_HANDLE TPMI_RH_CLEAR;
```

5.2.5.79 typedef TPMI_RH_NV_AUTH

```
typedef TPM_HANDLE TPMI_RH_NV_AUTH;
```

5.2.5.80 typedef TPMI_RH_LOCKOUT

```
typedef TPM_HANDLE TPMI_RH_LOCKOUT;
```

5.2.5.81 typedef TPMI_RH_NV_INDEX

```
typedef TPM_HANDLE TPMI_RH_NV_INDEX;
```

5.2.5.82 typedef TPMI_ALG_HASH

```
typedef TPM_ALG_ID TPMI_ALG_HASH;
```

5.2.5.83 typedef TPMI_ALG_ASYM

```
typedef TPM_ALG_ID TPMI_ALG_ASYM;
```

5.2.5.84 typedef TPMI_ALG_SYM

```
typedef TPM_ALG_ID TPMI_ALG_SYM;
```

5.2.5.85 typedef TPMI_ALG_SYM_OBJECT

```
typedef TPM_ALG_ID TPMI_ALG_SYM_OBJECT;
```

5.2.5.86 typedef TPMI_ALG_SYM_MODE

```
typedef TPM_ALG_ID TPMI_ALG_SYM_MODE;
```

5.2.5.87 typedef TPMI_ALG_KDF

```
typedef TPM_ALG_ID TPMI_ALG_KDF;
```

5.2.5.88 typedef TPMI_ALG_SIG_SCHEME

```
typedef TPM_ALG_ID TPMI_ALG_SIG_SCHEME;
```

5.2.5.89 typedef TPMI_ECC_KEY_EXCHANGE

```
typedef TPM_ALG_ID TPMI_ECC_KEY_EXCHANGE;
```

5.2.5.90 typedef TPMI_ST_COMMAND_TAG

```
typedef TPM_ST TPMI_ST_COMMAND_TAG;
```

5.2.5.91 typedef TPMS_ALGORITHM_DESCRIPTION

```
typedef struct TPMS_ALGORITHM_DESCRIPTION TPMS_ALGORITHM_DESCRIPTION;
```

5.2.5.92 typedef TPMU_HA

```
typedef union TPMU_HA TPMU_HA;
```

5.2.5.93 typedef TPMT_HA

```
typedef struct TPMT_HA TPMT_HA;
```

5.2.5.94 typedef TPM2B_DIGEST

```
typedef struct TPM2B_DIGEST TPM2B_DIGEST;
```

5.2.5.95 typedef TPM2B_DATA

```
typedef struct TPM2B_DATA TPM2B_DATA;
```

5.2.5.96 typedef TPM2B_NONCE

```
typedef TPM2B_DIGEST TPM2B_NONCE;
```

5.2.5.97 typedef TPM2B_AUTH

```
typedef TPM2B_DIGEST TPM2B_AUTH;
```

5.2.5.98 typedef TPM2B_OPERAND

```
typedef TPM2B_DIGEST TPM2B_OPERAND;
```

5.2.5.99 typedef TPM2B_EVENT

```
typedef struct TPM2B_EVENT TPM2B_EVENT;
```

5.2.5.100 typedef TPM2B_MAX_BUFFER

```
typedef struct TPM2B_MAX_BUFFER TPM2B_MAX_BUFFER;
```

5.2.5.101 typedef TPM2B_MAX_NV_BUFFER

```
typedef struct TPM2B_MAX_NV_BUFFER TPM2B_MAX_NV_BUFFER;
```

5.2.5.102 typedef TPM2B_TIMEOUT

```
typedef TPM2B_DIGEST TPM2B_TIMEOUT;
```

5.2.5.103 typedef TPM2B_IV

```
typedef struct TPM2B_IV TPM2B_IV;
```

5.2.5.104 typedef TPMU_NAME

```
typedef union TPMU_NAME TPMU_NAME;
```

5.2.5.105 typedef TPM2B_NAME

```
typedef struct TPM2B_NAME TPM2B_NAME;
```

5.2.5.106 typedef TPMS_PCR_SELECT

```
typedef struct TPMS_PCR_SELECT TPMS_PCR_SELECT;
```

5.2.5.107 typedef TPMS_PCR_SELECTION

```
typedef struct TPMS_PCR_SELECTION TPMS_PCR_SELECTION;
```

5.2.5.108 typedef TPMT_TK_CREATION

```
typedef struct TPMT_TK_CREATION TPMT_TK_CREATION;
```

5.2.5.109 typedef TPMT_TK_VERIFIED

```
typedef struct TPMT_TK_VERIFIED TPMT_TK_VERIFIED;
```

5.2.5.110 typedef TPMT_TK_AUTH

```
typedef struct TPMT_TK_AUTH TPMT_TK_AUTH;
```

5.2.5.111 typedef TPMT_TK_HASHCHECK

```
typedef struct TPMT_TK_HASHCHECK TPMT_TK_HASHCHECK;
```

5.2.5.112 typedef TPMS_ALG_PROPERTY

```
typedef struct TPMS_ALG_PROPERTY TPMS_ALG_PROPERTY;
```

5.2.5.113 typedef TPMS_TAGGED_PROPERTY

```
typedef struct TPMS_TAGGED_PROPERTY TPMS_TAGGED_PROPERTY;
```

5.2.5.114 typedef TPMS_TAGGED_PCR_SELECT

```
typedef struct TPMS_TAGGED_PCR_SELECT TPMS_TAGGED_PCR_SELECT;
```

5.2.5.115 typedef TPMS_TAGGED_POLICY

```
typedef struct TPMS_TAGGED_POLICY TPMS_TAGGED_POLICY;
```

5.2.5.116 typedef TPML_CC

```
typedef struct TPML_CC TPML_CC;
```

5.2.5.117 typedef TPML_CCA

```
typedef struct TPML_CCA TPML_CCA;
```

5.2.5.118 typedef TPML_ALG

```
typedef struct TPML_ALG TPML_ALG;
```

5.2.5.119 typedef TPML_HANDLE

```
typedef struct TPML_HANDLE TPML_HANDLE;
```

5.2.5.120 typedef TPML_DIGEST

```
typedef struct TPML_DIGEST TPML_DIGEST;
```

5.2.5.121 typedef TPML_DIGEST_VALUES

```
typedef struct TPML_DIGEST_VALUES TPML_DIGEST_VALUES;
```

5.2.5.122 typedef TPML_PCR_SELECTION

```
typedef struct TPML_PCR_SELECTION TPML_PCR_SELECTION;
```

5.2.5.123 typedef TPML_ALG_PROPERTY

```
typedef struct TPML_ALG_PROPERTY TPML_ALG_PROPERTY;
```

5.2.5.124 typedef TPML_TAGGED_TPM_PROPERTY

```
typedef struct TPML_TAGGED_TPM_PROPERTY TPML_TAGGED_TPM_PROPERTY;
```

5.2.5.125 typedef TPML_TAGGED_PCR_PROPERTY

```
typedef struct TPML_TAGGED_PCR_PROPERTY TPML_TAGGED_PCR_PROPERTY;
```

5.2.5.126 typedef TPML_ECC_CURVE

```
typedef struct TPML_ECC_CURVE TPML_ECC_CURVE;
```

5.2.5.127 typedef TPML_TAGGED_POLICY

```
typedef struct TPML_TAGGED_POLICY TPML_TAGGED_POLICY;
```

5.2.5.128 typedef TPMA_ACT

```
typedef UINT32 TPMA_ACT;
```

5.2.5.129 typedef TPMS_ACT_DATA

```
typedef struct TPMS_ACT_DATA TPMS_ACT_DATA;
```

5.2.5.130 typedef TPML_ACT_DATA

```
typedef struct TPML_ACT_DATA TPML_ACT_DATA;
```

5.2.5.131 typedef TPMU_CAPABILITIES

```
typedef union TPMU_CAPABILITIES TPMU_CAPABILITIES;
```

5.2.5.132 typedef TPMS_CAPABILITY_DATA

```
typedef struct TPMS_CAPABILITY_DATA TPMS_CAPABILITY_DATA;
```

5.2.5.133 typedef TPMS_CLOCK_INFO

```
typedef struct TPMS_CLOCK_INFO TPMS_CLOCK_INFO;
```

5.2.5.134 typedef TPMS_TIME_INFO

```
typedef struct TPMS_TIME_INFO TPMS_TIME_INFO;
```

5.2.5.135 typedef TPMS_TIME_ATTEST_INFO

```
typedef struct TPMS_TIME_ATTEST_INFO TPMS_TIME_ATTEST_INFO;
```

5.2.5.136 typedef TPMS_CERTIFY_INFO

```
typedef struct TPMS_CERTIFY_INFO TPMS_CERTIFY_INFO;
```

5.2.5.137 typedef TPMS_QUOTE_INFO

```
typedef struct TPMS_QUOTE_INFO TPMS_QUOTE_INFO;
```

5.2.5.138 typedef TPMS_COMMAND_AUDIT_INFO

```
typedef struct TPMS_COMMAND_AUDIT_INFO TPMS_COMMAND_AUDIT_INFO;
```

5.2.5.139 typedef TPMS_SESSION_AUDIT_INFO

```
typedef struct TPMS_SESSION_AUDIT_INFO TPMS_SESSION_AUDIT_INFO;
```

5.2.5.140 typedef TPMS_CREATION_INFO

```
typedef struct TPMS_CREATION_INFO TPMS_CREATION_INFO;
```

5.2.5.141 typedef TPMS_NV_CERTIFY_INFO

```
typedef struct TPMS_NV_CERTIFY_INFO TPMS_NV_CERTIFY_INFO;
```

5.2.5.142 typedef TPMI_ST_ATTEST

```
typedef TPM_ST TPMI_ST_ATTEST;
```

5.2.5.143 typedef TPMU_ATTEST

```
typedef union TPMU_ATTEST TPMU_ATTEST;
```

5.2.5.144 typedef TPMS_ATTEST

```
typedef struct TPMS_ATTEST TPMS_ATTEST;
```

5.2.5.145 typedef TPM2B_ATTEST

```
typedef struct TPM2B_ATTEST TPM2B_ATTEST;
```

5.2.5.146 typedef TPMI_AES_KEY_BITS

```
typedef TPM_KEY_BITS TPMI_AES_KEY_BITS;
```

5.2.5.147 typedef TPMU_SYM_KEY_BITS

```
typedef union TPMU_SYM_KEY_BITS TPMU_SYM_KEY_BITS;
```

5.2.5.148 typedef TPMU_SYM_MODE

```
typedef union TPMU_SYM_MODE TPMU_SYM_MODE;
```

5.2.5.149 typedef TPMT_SYM_DEF

```
typedef struct TPMT_SYM_DEF TPMT_SYM_DEF;
```

5.2.5.150 typedef TPMT_SYM_DEF_OBJECT

```
typedef TPMT_SYM_DEF TPMT_SYM_DEF_OBJECT;
```

5.2.5.151 typedef TPM2B_SYM_KEY

```
typedef struct TPM2B_SYM_KEY TPM2B_SYM_KEY;
```

5.2.5.152 typedef TPMS_SYMCIPHER_PARMS

```
typedef struct TPMS_SYMCIPHER_PARMS TPMS_SYMCIPHER_PARMS;
```

5.2.5.153 typedef TPM2B_LABEL

```
typedef struct TPM2B_LABEL TPM2B_LABEL;
```

5.2.5.154 typedef TPMS_DERIVE

```
typedef struct TPMS_DERIVE TPMS_DERIVE;
```

5.2.5.155 typedef TPM2B_DERIVE

```
typedef struct TPM2B_DERIVE TPM2B_DERIVE;
```

5.2.5.156 typedef TPMU_SENSITIVE_CREATE

```
typedef union TPMU_SENSITIVE_CREATE TPMU_SENSITIVE_CREATE;
```

5.2.5.157 typedef TPM2B_SENSITIVE_DATA

```
typedef struct TPM2B_SENSITIVE_DATA TPM2B_SENSITIVE_DATA;
```

5.2.5.158 typedef TPMS_SENSITIVE_CREATE

```
typedef struct TPMS_SENSITIVE_CREATE TPMS_SENSITIVE_CREATE;
```

5.2.5.159 typedef TPM2B_SENSITIVE_CREATE

```
typedef struct TPM2B_SENSITIVE_CREATE TPM2B_SENSITIVE_CREATE;
```

5.2.5.160 typedef TPMS_SCHEME_HASH

```
typedef struct TPMS_SCHEME_HASH TPMS_SCHEME_HASH;
```

5.2.5.161 typedef TPMS_SCHEME_ECDAA

```
typedef struct TPMS_SCHEME_ECDAA TPMS_SCHEME_ECDAA;
```

5.2.5.162 typedef TPMI_ALG_KEYEDHASH_SCHEME

```
typedef TPM_ALG_ID TPMI_ALG_KEYEDHASH_SCHEME;
```

5.2.5.163 typedef TPMS_SCHEME_HMAC

```
typedef TPMS_SCHEME_HASH TPMS_SCHEME_HMAC;
```

5.2.5.164 typedef TPMU_SCHEME_KEYEDHASH

```
typedef union TPMU_SCHEME_KEYEDHASH TPMU_SCHEME_KEYEDHASH;
```

5.2.5.165 typedef TPMT_KEYEDHASH_SCHEME

```
typedef struct TPMT_KEYEDHASH_SCHEME TPMT_KEYEDHASH_SCHEME;
```

5.2.5.166 typedef TPMS_SIG_SCHEME_RSASSA

```
typedef TPMS_SCHEME_HASH TPMS_SIG_SCHEME_RSASSA;
```

5.2.5.167 typedef TPMS_SIG_SCHEME_RSAPSS

```
typedef TPMS_SCHEME_HASH TPMS_SIG_SCHEME_RSAPSS;
```

5.2.5.168 typedef TPMS_SIG_SCHEME_ECDSA

```
typedef TPMS_SCHEME_HASH TPMS_SIG_SCHEME_ECDSA;
```

5.2.5.169 typedef TPMS_SIG_SCHEME_ECDAA

```
typedef TPMS_SCHEME_ECDAA TPMS_SIG_SCHEME_ECDAA;
```

5.2.5.170 typedef TPMU_SIG_SCHEME

```
typedef union TPMU_SIG_SCHEME TPMU_SIG_SCHEME;
```

5.2.5.171 typedef TPMT_SIG_SCHEME

```
typedef struct TPMT_SIG_SCHEME TPMT_SIG_SCHEME;
```

5.2.5.172 typedef TPMS_ENC_SCHEME_OAEP

```
typedef TPMS_SCHEME_HASH TPMS_ENC_SCHEME_OAEP;
```

5.2.5.173 typedef TPMS_KEY_SCHEME_ECDH

```
typedef TPMS_SCHEME_HASH TPMS_KEY_SCHEME_ECDH;
```

5.2.5.174 typedef TPMS_KEY_SCHEME_ECMQV

```
typedef TPMS_SCHEME_HASH TPMS_KEY_SCHEME_ECMQV;
```

5.2.5.175 typedef TPMS_SCHEME_MGF1

```
typedef TPMS_SCHEME_HASH TPMS_SCHEME_MGF1;
```

5.2.5.176 typedef TPMS_SCHEME_KDF1_SP800_56A

```
typedef TPMS_SCHEME_HASH TPMS_SCHEME_KDF1_SP800_56A;
```

5.2.5.177 typedef TPMS_SCHEME_KDF2

```
typedef TPMS_SCHEME_HASH TPMS_SCHEME_KDF2;
```

5.2.5.178 typedef TPMS_SCHEME_KDF1_SP800_108

```
typedef TPMS_SCHEME_HASH TPMS_SCHEME_KDF1_SP800_108;
```

5.2.5.179 typedef TPMU_KDF_SCHEME

```
typedef union TPMU_KDF_SCHEME TPMU_KDF_SCHEME;
```

5.2.5.180 typedef TPMT_KDF_SCHEME

```
typedef struct TPMT_KDF_SCHEME TPMT_KDF_SCHEME;
```

5.2.5.181 typedef TPMI_ALG_ASYM_SCHEME

```
typedef TPM_ALG_ID TPMI_ALG_ASYM_SCHEME;
```

5.2.5.182 typedef TPMU_ASYM_SCHEME

```
typedef union TPMU_ASYM_SCHEME TPMU_ASYM_SCHEME;
```

5.2.5.183 typedef TPMT_ASYM_SCHEME

```
typedef struct TPMT_ASYM_SCHEME TPMT_ASYM_SCHEME;
```

5.2.5.184 typedef TPMI_ALG_RSA_SCHEME

```
typedef TPM_ALG_ID TPMI_ALG_RSA_SCHEME;
```

5.2.5.185 typedef TPMT_RSA_SCHEME

```
typedef struct TPMT_RSA_SCHEME TPMT_RSA_SCHEME;
```

5.2.5.186 typedef TPMI_ALG_RSA_DECRYPT

```
typedef TPM_ALG_ID TPMI_ALG_RSA_DECRYPT;
```

5.2.5.187 typedef TPMT_RSA_DECRYPT

```
typedef struct TPMT_RSA_DECRYPT TPMT_RSA_DECRYPT;
```

5.2.5.188 typedef TPM2B_PUBLIC_KEY_RSA

```
typedef struct TPM2B_PUBLIC_KEY_RSA TPM2B_PUBLIC_KEY_RSA;
```

5.2.5.189 typedef TPMI_RSA_KEY_BITS

```
typedef TPM_KEY_BITS TPMI_RSA_KEY_BITS;
```

5.2.5.190 typedef TPM2B_PRIVATE_KEY_RSA

```
typedef struct TPM2B_PRIVATE_KEY_RSA TPM2B_PRIVATE_KEY_RSA;
```

5.2.5.191 typedef TPM2B_ECC_PARAMETER

```
typedef struct TPM2B_ECC_PARAMETER TPM2B_ECC_PARAMETER;
```

5.2.5.192 typedef TPMS_ECC_POINT

```
typedef struct TPMS_ECC_POINT TPMS_ECC_POINT;
```

5.2.5.193 typedef TPM2B_ECC_POINT

```
typedef struct TPM2B_ECC_POINT TPM2B_ECC_POINT;
```

5.2.5.194 typedef TPMI_ALG_ECC_SCHEME

```
typedef TPM_ALG_ID TPMI_ALG_ECC_SCHEME;
```

5.2.5.195 typedef TPMI_ECC_CURVE

```
typedef TPM_ECC_CURVE TPMI_ECC_CURVE;
```

5.2.5.196 typedef TPMT_ECC_SCHEME

```
typedef TPMT_SIG_SCHEME TPMT_ECC_SCHEME;
```

5.2.5.197 typedef TPMS_ALGORITHM_DETAIL_ECC

```
typedef struct TPMS_ALGORITHM_DETAIL_ECC TPMS_ALGORITHM_DETAIL_ECC;
```

5.2.5.198 typedef TPMS_SIGNATURE_RSA

```
typedef struct TPMS_SIGNATURE_RSA TPMS_SIGNATURE_RSA;
```

5.2.5.199 typedef TPMS_SIGNATURE_RSASSA

```
typedef TPMS_SIGNATURE_RSA TPMS_SIGNATURE_RSASSA;
```

5.2.5.200 typedef TPMS_SIGNATURE_RSAPSS

```
typedef TPMS_SIGNATURE_RSA TPMS_SIGNATURE_RSAPSS;
```

5.2.5.201 typedef TPMS_SIGNATURE_ECC

```
typedef struct TPMS_SIGNATURE_ECC TPMS_SIGNATURE_ECC;
```

5.2.5.202 typedef TPMS_SIGNATURE_ECDSA

```
typedef TPMS_SIGNATURE_ECC TPMS_SIGNATURE_ECDSA;
```

5.2.5.203 typedef TPMS_SIGNATURE_ECDAA

```
typedef TPMS_SIGNATURE_ECC TPMS_SIGNATURE_ECDAA;
```

5.2.5.204 typedef TPMU_SIGNATURE

```
typedef union TPMU_SIGNATURE TPMU_SIGNATURE;
```

5.2.5.205 typedef TPMT_SIGNATURE

```
typedef struct TPMT_SIGNATURE TPMT_SIGNATURE;
```

5.2.5.206 typedef TPMU_ENCRYPTED_SECRET

```
typedef union TPMU_ENCRYPTED_SECRET TPMU_ENCRYPTED_SECRET;
```

5.2.5.207 typedef TPM2B_ENCRYPTED_SECRET

```
typedef struct TPM2B_ENCRYPTED_SECRET TPM2B_ENCRYPTED_SECRET;
```

5.2.5.208 typedef TPMI_ALG_PUBLIC

```
typedef TPM_ALG_ID TPMI_ALG_PUBLIC;
```

5.2.5.209 typedef TPMU_PUBLIC_ID

```
typedef union TPMU_PUBLIC_ID TPMU_PUBLIC_ID;
```

5.2.5.210 typedef TPMS_KEYEDHASH_PARMS

```
typedef struct TPMS_KEYEDHASH_PARMS TPMS_KEYEDHASH_PARMS;
```

5.2.5.211 typedef TPMS_ASYM_PARMS

```
typedef struct TPMS_ASYM_PARMS TPMS_ASYM_PARMS;
```

5.2.5.212 typedef TPMS_RSA_PARMS

```
typedef struct TPMS_RSA_PARMS TPMS_RSA_PARMS;
```

5.2.5.213 typedef TPMS_ECC_PARMS

```
typedef struct TPMS_ECC_PARMS TPMS_ECC_PARMS;
```

5.2.5.214 typedef TPMU_PUBLIC_PARMS

```
typedef union TPMU_PUBLIC_PARMS TPMU_PUBLIC_PARMS;
```

5.2.5.215 typedef TPMT_PUBLIC_PARMS

```
typedef struct TPMT_PUBLIC_PARMS TPMT_PUBLIC_PARMS;
```

5.2.5.216 typedef TPMT_PUBLIC

```
typedef struct TPMT_PUBLIC TPMT_PUBLIC;
```

5.2.5.217 typedef TPM2B_PUBLIC

```
typedef struct TPM2B_PUBLIC TPM2B_PUBLIC;
```

5.2.5.218 typedef TPM2B_TEMPLATE

```
typedef struct TPM2B_TEMPLATE TPM2B_TEMPLATE;
```

5.2.5.219 typedef TPM2B_PRIVATE_VENDOR_SPECIFIC

```
typedef struct TPM2B_PRIVATE_VENDOR_SPECIFIC TPM2B_PRIVATE_VENDOR_SPECIFIC;
```

5.2.5.220 typedef TPMU_SENSITIVE_COMPOSITE

```
typedef union TPMU_SENSITIVE_COMPOSITE TPMU_SENSITIVE_COMPOSITE;
```

5.2.5.221 typedef TPMT_SENSITIVE

```
typedef struct TPMT_SENSITIVE TPMT_SENSITIVE;
```

5.2.5.222 typedef TPM2B_SENSITIVE

```
typedef struct TPM2B_SENSITIVE TPM2B_SENSITIVE;
```

5.2.5.223 typedef TPMT_PRIVATE

```
typedef struct TPMT_PRIVATE TPMT_PRIVATE;
```

5.2.5.224 typedef TPM2B_PRIVATE

```
typedef struct TPM2B_PRIVATE TPM2B_PRIVATE;
```

5.2.5.225 typedef TPMS_ID_OBJECT

```
typedef struct TPMS_ID_OBJECT TPMS_ID_OBJECT;
```

5.2.5.226 typedef TPM2B_ID_OBJECT

```
typedef struct TPM2B_ID_OBJECT TPM2B_ID_OBJECT;
```

5.2.5.227 typedef TPM_NV_INDEX

```
typedef UINT32 TPM_NV_INDEX;
```

5.2.5.228 typedef TPM_NT

```
typedef enum TPM_NT TPM_NT;
```

5.2.5.229 typedef TPMS_NV_PIN_COUNTER_PARAMETERS

```
typedef struct TPMS_NV_PIN_COUNTER_PARAMETERS TPMS_NV_PIN_COUNTER_PARAMETERS;
```

5.2.5.230 typedef TPMA_NV

```
typedef UINT32 TPMA_NV;
```

5.2.5.231 typedef TPMS_NV_PUBLIC

```
typedef struct TPMS_NV_PUBLIC TPMS_NV_PUBLIC;
```

5.2.5.232 typedef TPM2B_NV_PUBLIC

```
typedef struct TPM2B_NV_PUBLIC TPM2B_NV_PUBLIC;
```

5.2.5.233 typedef TPM2B_CONTEXT_SENSITIVE

```
typedef struct TPM2B_CONTEXT_SENSITIVE TPM2B_CONTEXT_SENSITIVE;
```

5.2.5.234 typedef TPMS_CONTEXT_DATA

```
typedef struct TPMS_CONTEXT_DATA TPMS_CONTEXT_DATA;
```

5.2.5.235 typedef TPM2B_CONTEXT_DATA

```
typedef struct TPM2B_CONTEXT_DATA TPM2B_CONTEXT_DATA;
```

5.2.5.236 typedef TPMS_CONTEXT

```
typedef struct TPMS_CONTEXT TPMS_CONTEXT;
```

5.2.5.237 typedef TPMS_CREATION_DATA

```
typedef struct TPMS_CREATION_DATA TPMS_CREATION_DATA;
```

5.2.5.238 typedef TPM2B_CREATION_DATA

```
typedef struct TPM2B_CREATION_DATA TPM2B_CREATION_DATA;
```

5.2.5.239 typedef TPMS_AUTH_COMMAND

```
typedef struct TPMS_AUTH_COMMAND TPMS_AUTH_COMMAND;
```

5.2.5.240 typedef TPMS_AUTH_RESPONSE

```
typedef struct TPMS_AUTH_RESPONSE TPMS_AUTH_RESPONSE;
```

5.2.5.241 typedef TPM2_AUTH_SESSION

```
typedef struct TPM2_AUTH_SESSION TPM2_AUTH_SESSION;
```

5.2.5.242 typedef TPM2HalIoCb

```
typedef int(* TPM2HalIoCb)(struct TPM2_CTX *, const BYTE *txBuf, BYTE *rxBuf,
 ↪  UINT16 xferSz, void *userCtx);
```

5.2.5.243 typedef TPM2_CTX

```
typedef struct TPM2_CTX TPM2_CTX;
```

5.2.5.244 typedef ChangePPS_In

```
typedef ChangeSeed_In ChangePPS_In;
```

5.2.5.245 typedef ChangeEPS_In

```
typedef ChangeSeed_In ChangeEPS_In;
```

5.2.5.246 typedef TPM_MODE_SET

```
typedef struct TPM_MODE_SET TPM_MODE_SET;
```

5.2.5.247 typedef GetRandom2_In

```
typedef GetRandom_In GetRandom2_In;
```

5.2.5.248 typedef TPMI_GPIO_NAME

```
typedef UINT32 TPMI_GPIO_NAME;
```

5.2.5.249 typedef TPMI_GPIO_MODE

```
typedef UINT32 TPMI_GPIO_MODE;
```

5.2.5.250 typedef TPMS_GPIO_CONFIG

```
typedef struct TPMS_GPIO_CONFIG TPMS_GPIO_CONFIG;
```

5.2.5.251 typedef TPML_GPIO_CONFIG

```
typedef struct TPML_GPIO_CONFIG TPML_GPIO_CONFIG;
```

5.2.6 Functions Documentation

5.2.6.1 function TPM2_Startup

```
WOLFTPM_API TPM_RC TPM2_Startup(
    Startup_In * in
)
```

5.2.6.2 function TPM2_Shutdown

```
WOLFTPM_API TPM_RC TPM2_Shutdown(
    Shutdown_In * in
)
```

5.2.6.3 function TPM2_GetCapability

```
WOLFTPM_API TPM_RC TPM2_GetCapability(
    GetCapability_In * in,
    GetCapability_Out * out
)
```

5.2.6.4 function TPM2_SelfTest

```
WOLFTPM_API TPM_RC TPM2_SelfTest(
    SelfTest_In * in
)
```

5.2.6.5 function TPM2_IncrementalSelfTest

```
WOLFTPM_API TPM_RC TPM2_IncrementalSelfTest(
    IncrementalSelfTest_In * in,
    IncrementalSelfTest_Out * out
)
```

5.2.6.6 function TPM2_GetTestResult

```
WOLFTPM_API TPM_RC TPM2_GetTestResult(
    GetTestResult_Out * out
)
```

5.2.6.7 function TPM2_GetRandom

```
WOLFTPM_API TPM_RC TPM2_GetRandom(
    GetRandom_In * in,
    GetRandom_Out * out
)
```

5.2.6.8 function TPM2_StirRandom

```
WOLFTPM_API TPM_RC TPM2_StirRandom(
    StirRandom_In * in
)
```

5.2.6.9 function TPM2_PCR_Read

```
WOLFTPM_API TPM_RC TPM2_PCR_Read(
    PCR_Read_In * in,
    PCR_Read_Out * out
)
```

5.2.6.10 function TPM2_PCR_Extend

```
WOLFTPM_API TPM_RC TPM2_PCR_Extend(
    PCR_Extend_In * in
)
```

5.2.6.11 function TPM2_Create

```
WOLFTPM_API TPM_RC TPM2_Create(
    Create_In * in,
    Create_Out * out
)
```

5.2.6.12 function TPM2_CreateLoaded

```
WOLFTPM_API TPM_RC TPM2_CreateLoaded(
    CreateLoaded_In * in,
    CreateLoaded_Out * out
)
```

5.2.6.13 function TPM2_CreatePrimary

```
WOLFTPM_API TPM_RC TPM2_CreatePrimary(
    CreatePrimary_In * in,
    CreatePrimary_Out * out
)
```

5.2.6.14 function TPM2_Load

```
WOLFTPM_API TPM_RC TPM2_Load(
    Load_In * in,
    Load_Out * out
)
```

5.2.6.15 function TPM2_FlushContext

```
WOLFTPM_API TPM_RC TPM2_FlushContext(
    FlushContext_In * in
)
```

5.2.6.16 function TPM2_Unseal

```
WOLFTPM_API TPM_RC TPM2_Unseal(
    Unseal_In * in,
    Unseal_Out * out
)
```

5.2.6.17 function TPM2_StartAuthSession

```
WOLFTPM_API TPM_RC TPM2_StartAuthSession(
    StartAuthSession_In * in,
    StartAuthSession_Out * out
)
```

5.2.6.18 function TPM2_PolicyRestart

```
WOLFTPM_API TPM_RC TPM2_PolicyRestart(
    PolicyRestart_In * in
)
```

5.2.6.19 function TPM2_LoadExternal

```
WOLFTPM_API TPM_RC TPM2_LoadExternal(
    LoadExternal_In * in,
    LoadExternal_Out * out
)
```

5.2.6.20 function TPM2_ReadPublic

```
WOLFTPM_API TPM_RC TPM2_ReadPublic(
    ReadPublic_In * in,
    ReadPublic_Out * out
)
```

5.2.6.21 function TPM2_ActivateCredential

```
WOLFTPM_API TPM_RC TPM2_ActivateCredential(
    ActivateCredential_In * in,
    ActivateCredential_Out * out
)
```

5.2.6.22 function TPM2_MakeCredential

```
WOLFTPM_API TPM_RC TPM2_MakeCredential(
    MakeCredential_In * in,
    MakeCredential_Out * out
)
```

5.2.6.23 function TPM2_ObjectChangeAuth

```
WOLFTPM_API TPM_RC TPM2_ObjectChangeAuth(
    ObjectChangeAuth_In * in,
    ObjectChangeAuth_Out * out
)
```

5.2.6.24 function TPM2_Duplicate

```
WOLFTPM_API TPM_RC TPM2_Duplicate(
    Duplicate_In * in,
    Duplicate_Out * out
)
```

5.2.6.25 function TPM2_Rewrap

```
WOLFTPM_API TPM_RC TPM2_Rewrap(
    Rewrap_In * in,
    Rewrap_Out * out
)
```

5.2.6.26 function TPM2_Import

```
WOLFTPM_API TPM_RC TPM2_Import(
    Import_In * in,
    Import_Out * out
)
```

5.2.6.27 function TPM2_RSA_Encrypt

```
WOLFTPM_API TPM_RC TPM2_RSA_Encrypt(
    RSA_Encrypt_In * in,
    RSA_Encrypt_Out * out
)
```

5.2.6.28 function TPM2_RSA_Decrypt

```
WOLFTPM_API TPM_RC TPM2_RSA_Decrypt(
    RSA_Decrypt_In * in,
    RSA_Decrypt_Out * out
)
```

5.2.6.29 function TPM2_ECDH_KeyGen

```
WOLFTPM_API TPM_RC TPM2_ECDH_KeyGen(
    ECDH_KeyGen_In * in,
    ECDH_KeyGen_Out * out
)
```

5.2.6.30 function TPM2_ECDH_ZGen

```
WOLFTPM_API TPM_RC TPM2_ECDH_ZGen(
    ECDH_ZGen_In * in,
    ECDH_ZGen_Out * out
)
```

5.2.6.31 function TPM2_ECC_Parameters

```
WOLFTPM_API TPM_RC TPM2_ECC_Parameters(
    ECC_Parameters_In * in,
    ECC_Parameters_Out * out
)
```

5.2.6.32 function TPM2_ZGen_2Phase

```
WOLFTPM_API TPM_RC TPM2_ZGen_2Phase(
    ZGen_2Phase_In * in,
    ZGen_2Phase_Out * out
)
```

5.2.6.33 function TPM2_EncryptDecrypt

```
WOLFTPM_API TPM_RC TPM2_EncryptDecrypt(
    EncryptDecrypt_In * in,
    EncryptDecrypt_Out * out
)
```

5.2.6.34 function TPM2_EncryptDecrypt2

```
WOLFTPM_API TPM_RC TPM2_EncryptDecrypt2(
    EncryptDecrypt2_In * in,
    EncryptDecrypt2_Out * out
)
```

5.2.6.35 function TPM2_Hash

```
WOLFTPM_API TPM_RC TPM2_Hash(
    Hash_In * in,
    Hash_Out * out
)
```

5.2.6.36 function TPM2_HMAC

```
WOLFTPM_API TPM_RC TPM2_HMAC(
    HMAC_In * in,
    HMAC_Out * out
)
```

5.2.6.37 function TPM2_HMAC_Start

```
WOLFTPM_API TPM_RC TPM2_HMAC_Start(
    HMAC_Start_In * in,
    HMAC_Start_Out * out
)
```

5.2.6.38 function TPM2_HashSequenceStart

```
WOLFTPM_API TPM_RC TPM2_HashSequenceStart(
    HashSequenceStart_In * in,
    HashSequenceStart_Out * out
)
```

5.2.6.39 function TPM2_SequenceUpdate

```
WOLFTPM_API TPM_RC TPM2_SequenceUpdate(
    SequenceUpdate_In * in
)
```

5.2.6.40 function TPM2_SequenceComplete

```
WOLFTPM_API TPM_RC TPM2_SequenceComplete(
    SequenceComplete_In * in,
    SequenceComplete_Out * out
)
```

5.2.6.41 function TPM2_EventSequenceComplete

```
WOLFTPM_API TPM_RC TPM2_EventSequenceComplete(
    EventSequenceComplete_In * in,
    EventSequenceComplete_Out * out
)
```

5.2.6.42 function TPM2_Certify

```
WOLFTPM_API TPM_RC TPM2_Certify(
    Certify_In * in,
    Certify_Out * out
)
```

5.2.6.43 function TPM2_CertifyCreation

```
WOLFTPM_API TPM_RC TPM2_CertifyCreation(
    CertifyCreation_In * in,
    CertifyCreation_Out * out
)
```

5.2.6.44 function TPM2_Quote

```
WOLFTPM_API TPM_RC TPM2_Quote(
    Quote_In * in,
    Quote_Out * out
)
```

5.2.6.45 function TPM2_GetSessionAuditDigest

```
WOLFTPM_API TPM_RC TPM2_GetSessionAuditDigest(
    GetSessionAuditDigest_In * in,
    GetSessionAuditDigest_Out * out
)
```

5.2.6.46 function TPM2_GetCommandAuditDigest

```
WOLFTPM_API TPM_RC TPM2_GetCommandAuditDigest(
    GetCommandAuditDigest_In * in,
    GetCommandAuditDigest_Out * out
)
```

5.2.6.47 function TPM2_GetTime

```
WOLFTPM_API TPM_RC TPM2_GetTime(
    GetTime_In * in,
    GetTime_Out * out
)
```

5.2.6.48 function TPM2_Commit

```
WOLFTPM_API TPM_RC TPM2_Commit(
    Commit_In * in,
    Commit_Out * out
)
```

5.2.6.49 function TPM2_EC_Ephemeral

```
WOLFTPM_API TPM_RC TPM2_EC_Ephemeral(
    EC_Ephemeral_In * in,
    EC_Ephemeral_Out * out
)
```

5.2.6.50 function TPM2_VerifySignature

```
WOLFTPM_API TPM_RC TPM2_VerifySignature(
    VerifySignature_In * in,
    VerifySignature_Out * out
)
```

5.2.6.51 function TPM2_Sign

```
WOLFTPM_API TPM_RC TPM2_Sign(
    Sign_In * in,
    Sign_Out * out
)
```

5.2.6.52 function TPM2_SetCommandCodeAuditStatus

```
WOLFTPM_API TPM_RC TPM2_SetCommandCodeAuditStatus(
    SetCommandCodeAuditStatus_In * in
)
```

5.2.6.53 function TPM2_PCR_Event

```
WOLFTPM_API TPM_RC TPM2_PCR_Event(
    PCR_Event_In * in,
    PCR_Event_Out * out
)
```

5.2.6.54 function TPM2_PCR_Allocate

```
WOLFTPM_API TPM_RC TPM2_PCR_Allocate(
    PCR_Allocate_In * in,
    PCR_Allocate_Out * out
)
```

5.2.6.55 function TPM2_PCR_SetAuthPolicy

```
WOLFTPM_API TPM_RC TPM2_PCR_SetAuthPolicy(
    PCR_SetAuthPolicy_In * in
)
```

5.2.6.56 function TPM2_PCR_SetAuthValue

```
WOLFTPM_API TPM_RC TPM2_PCR_SetAuthValue(
    PCR_SetAuthValue_In * in
)
```

5.2.6.57 function TPM2_PCR_Reset

```
WOLFTPM_API TPM_RC TPM2_PCR_Reset(
    PCR_Reset_In * in
)
```

5.2.6.58 function TPM2_PolicySigned

```
WOLFTPM_API TPM_RC TPM2_PolicySigned(
    PolicySigned_In * in,
    PolicySigned_Out * out
)
```

5.2.6.59 function TPM2_PolicySecret

```
WOLFTPM_API TPM_RC TPM2_PolicySecret(
    PolicySecret_In * in,
    PolicySecret_Out * out
)
```

5.2.6.60 function TPM2_PolicyTicket

```
WOLFTPM_API TPM_RC TPM2_PolicyTicket(
    PolicyTicket_In * in
)
```

5.2.6.61 function TPM2_PolicyOR

```
WOLFTPM_API TPM_RC TPM2_PolicyOR(
    PolicyOR_In * in
)
```

5.2.6.62 function TPM2_PolicyPCR

```
WOLFTPM_API TPM_RC TPM2_PolicyPCR(
    PolicyPCR_In * in
)
```

5.2.6.63 function TPM2_PolicyLocality

```
WOLFTPM_API TPM_RC TPM2_PolicyLocality(
    PolicyLocality_In * in
)
```

5.2.6.64 function TPM2_PolicyNV

```
WOLFTPM_API TPM_RC TPM2_PolicyNV(
    PolicyNV_In * in
)
```

5.2.6.65 function TPM2_PolicyCounterTimer

```
WOLFTPM_API TPM_RC TPM2_PolicyCounterTimer(
    PolicyCounterTimer_In * in
)
```

5.2.6.66 function TPM2_PolicyCommandCode

```
WOLFTPM_API TPM_RC TPM2_PolicyCommandCode(
    PolicyCommandCode_In * in
)
```

5.2.6.67 function TPM2_PolicyPhysicalPresence

```
WOLFTPM_API TPM_RC TPM2_PolicyPhysicalPresence(
    PolicyPhysicalPresence_In * in
)
```

5.2.6.68 function TPM2_PolicyCpHash

```
WOLFTPM_API TPM_RC TPM2_PolicyCpHash(
    PolicyCpHash_In * in
)
```

5.2.6.69 function TPM2_PolicyNameHash

```
WOLFTPM_API TPM_RC TPM2_PolicyNameHash(
    PolicyNameHash_In * in
)
```

5.2.6.70 function TPM2_PolicyDuplicationSelect

```
WOLFTPM_API TPM_RC TPM2_PolicyDuplicationSelect(
    PolicyDuplicationSelect_In * in
)
```

5.2.6.71 function TPM2_PolicyAuthorize

```
WOLFTPM_API TPM_RC TPM2_PolicyAuthorize(
    PolicyAuthorize_In * in
)
```

5.2.6.72 function TPM2_PolicyAuthValue

```
WOLFTPM_API TPM_RC TPM2_PolicyAuthValue(
    PolicyAuthValue_In * in
)
```

5.2.6.73 function TPM2_PolicyPassword

```
WOLFTPM_API TPM_RC TPM2_PolicyPassword(
    PolicyPassword_In * in
)
```

5.2.6.74 function TPM2_PolicyGetDigest

```
WOLFTPM_API TPM_RC TPM2_PolicyGetDigest(
    PolicyGetDigest_In * in,
    PolicyGetDigest_Out * out
)
```

5.2.6.75 function TPM2_PolicyNvWritten

```
WOLFTPM_API TPM_RC TPM2_PolicyNvWritten(
    PolicyNvWritten_In * in
)
```

5.2.6.76 function TPM2_PolicyTemplate

```
WOLFTPM_API TPM_RC TPM2_PolicyTemplate(
    PolicyTemplate_In * in
)
```

5.2.6.77 function TPM2_PolicyAuthorizeNV

```
WOLFTPM_API TPM_RC TPM2_PolicyAuthorizeNV(
    PolicyAuthorizeNV_In * in
)
```

5.2.6.78 function _TPM_Hash_Start

```
WOLFTPM_API void _TPM_Hash_Start(
    void
)
```

5.2.6.79 function _TPM_Hash_Data

```
WOLFTPM_API void _TPM_Hash_Data(
    UINT32 dataSize,
    BYTE * data
)
```

5.2.6.80 function _TPM_Hash_End

```
WOLFTPM_API void _TPM_Hash_End(
    void
)
```

5.2.6.81 function TPM2_HierarchyControl

```
WOLFTPM_API TPM_RC TPM2_HierarchyControl(
    HierarchyControl_In * in
)
```

5.2.6.82 function TPM2_SetPrimaryPolicy

```
WOLFTPM_API TPM_RC TPM2_SetPrimaryPolicy(
    SetPrimaryPolicy_In * in
)
```

5.2.6.83 function TPM2_ChangePPS

```
WOLFTPM_API TPM_RC TPM2_ChangePPS(
    ChangePPS_In * in
)
```

5.2.6.84 function TPM2_ChangeEPS

```
WOLFTPM_API TPM_RC TPM2_ChangeEPS(
    ChangeEPS_In * in
)
```

5.2.6.85 function TPM2_Clear

```
WOLFTPM_API TPM_RC TPM2_Clear(
    Clear_In * in
)
```

5.2.6.86 function TPM2_ClearControl

```
WOLFTPM_API TPM_RC TPM2_ClearControl(
    ClearControl_In * in
)
```

5.2.6.87 function TPM2_HierarchyChangeAuth

```
WOLFTPM_API TPM_RC TPM2_HierarchyChangeAuth(
    HierarchyChangeAuth_In * in
)
```

5.2.6.88 function TPM2_DictionaryAttackLockReset

```
WOLFTPM_API TPM_RC TPM2_DictionaryAttackLockReset(
    DictionaryAttackLockReset_In * in
)
```

5.2.6.89 function TPM2_DictionaryAttackParameters

```
WOLFTPM_API TPM_RC TPM2_DictionaryAttackParameters(
    DictionaryAttackParameters_In * in
)
```

5.2.6.90 function TPM2_PP_Commands

```
WOLFTPM_API TPM_RC TPM2_PP_Commands(
    PP_Commands_In * in
)
```

5.2.6.91 function TPM2_SetAlgorithmSet

```
WOLFTPM_API TPM_RC TPM2_SetAlgorithmSet(
    SetAlgorithmSet_In * in
)
```

5.2.6.92 function TPM2_FieldUpgradeStart

```
WOLFTPM_API TPM_RC TPM2_FieldUpgradeStart(
    FieldUpgradeStart_In * in
)
```

5.2.6.93 function TPM2_FieldUpgradeData

```
WOLFTPM_API TPM_RC TPM2_FieldUpgradeData(
    FieldUpgradeData_In * in,
    FieldUpgradeData_Out * out
)
```

5.2.6.94 function TPM2_FirmwareRead

```
WOLFTPM_API TPM_RC TPM2_FirmwareRead(
    FirmwareRead_In * in,
    FirmwareRead_Out * out
)
```

5.2.6.95 function TPM2_ContextSave

```
WOLFTPM_API TPM_RC TPM2_ContextSave(
    ContextSave_In * in,
    ContextSave_Out * out
)
```

5.2.6.96 function TPM2_ContextLoad

```
WOLFTPM_API TPM_RC TPM2_ContextLoad(
    ContextLoad_In * in,
    ContextLoad_Out * out
)
```

5.2.6.97 function TPM2_EvictControl

```
WOLFTPM_API TPM_RC TPM2_EvictControl(
    EvictControl_In * in
)
```

5.2.6.98 function TPM2_ReadClock

```
WOLFTPM_API TPM_RC TPM2_ReadClock(
    ReadClock_Out * out
)
```

5.2.6.99 function TPM2_ClockSet

```
WOLFTPM_API TPM_RC TPM2_ClockSet(
    ClockSet_In * in
)
```

5.2.6.100 function TPM2_ClockRateAdjust

```
WOLFTPM_API TPM_RC TPM2_ClockRateAdjust(
    ClockRateAdjust_In * in
)
```

5.2.6.101 function TPM2_TestParms

```
WOLFTPM_API TPM_RC TPM2_TestParms(
    TestParms_In * in
)
```

5.2.6.102 function TPM2_NV_DefineSpace

```
WOLFTPM_API TPM_RC TPM2_NV_DefineSpace(
    NV_DefineSpace_In * in
)
```

5.2.6.103 function TPM2_NV_UndefineSpace

```
WOLFTPM_API TPM_RC TPM2_NV_UndefineSpace(
    NV_UndefineSpace_In * in
)
```

5.2.6.104 function TPM2_NV_UndefineSpaceSpecial

```
WOLFTPM_API TPM_RC TPM2_NV_UndefineSpaceSpecial(
    NV_UndefineSpaceSpecial_In * in
)
```

5.2.6.105 function TPM2_NV_ReadPublic

```
WOLFTPM_API TPM_RC TPM2_NV_ReadPublic(
    NV_ReadPublic_In * in,
    NV_ReadPublic_Out * out
)
```

5.2.6.106 function TPM2_NV_Write

```
WOLFTPM_API TPM_RC TPM2_NV_Write(
    NV_Write_In * in
)
```

5.2.6.107 function TPM2_NV_Increment

```
WOLFTPM_API TPM_RC TPM2_NV_Increment(
    NV_Increment_In * in
)
```

5.2.6.108 function TPM2_NV_Extend

```
WOLFTPM_API TPM_RC TPM2_NV_Extend(
    NV_Extend_In * in
)
```

5.2.6.109 function TPM2_NV_SetBits

```
WOLFTPM_API TPM_RC TPM2_NV_SetBits(
    NV_SetBits_In * in
)
```

5.2.6.110 function TPM2_NV_WriteLock

```
WOLFTPM_API TPM_RC TPM2_NV_WriteLock(
    NV_WriteLock_In * in
)
```

5.2.6.111 function TPM2_NV_GlobalWriteLock

```
WOLFTPM_API TPM_RC TPM2_NV_GlobalWriteLock(
    NV_GlobalWriteLock_In * in
)
```

5.2.6.112 function TPM2_NV_Read

```
WOLFTPM_API TPM_RC TPM2_NV_Read(
    NV_Read_In * in,
    NV_Read_Out * out
)
```

5.2.6.113 function TPM2_NV_ReadLock

```
WOLFTPM_API TPM_RC TPM2_NV_ReadLock(
    NV_ReadLock_In * in
)
```

5.2.6.114 function TPM2_NV_ChangeAuth

```
WOLFTPM_API TPM_RC TPM2_NV_ChangeAuth(
    NV_ChangeAuth_In * in
)
```

5.2.6.115 function TPM2_NV_Certify

```
WOLFTPM_API TPM_RC TPM2_NV_Certify(
    NV_Certify_In * in,
    NV_Certify_Out * out
)
```

5.2.6.116 function TPM2_SetCommandSet

```
WOLFTPM_API int TPM2_SetCommandSet(
    SetCommandSet_In * in
)
```

5.2.6.117 function TPM2_SetMode

```
WOLFTPM_API int TPM2_SetMode(
    SetMode_In * in
)
```

5.2.6.118 function TPM2_GetRandom2

```
WOLFTPM_API TPM_RC TPM2_GetRandom2(
    GetRandom2_In * in,
    GetRandom2_Out * out
)
```

5.2.6.119 function TPM2_GetProductInfo

```
WOLFTPM_API TPM_RC TPM2_GetProductInfo(
    uint8_t * info,
    uint16_t size
)
```

5.2.6.120 function TPM2_IFX_FieldUpgradeStart

```
WOLFTPM_API int TPM2_IFX_FieldUpgradeStart(
    TPM_HANDLE sessionHandle,
    uint8_t * data,
    uint32_t size
)
```

5.2.6.121 function TPM2_IFX_FieldUpgradeCommand

```
WOLFTPM_API int TPM2_IFX_FieldUpgradeCommand(
    TPM_CC cc,
    uint8_t * data,
    uint32_t size
)
```

5.2.6.122 function TPM2_GPIO_Config

```
WOLFTPM_API int TPM2_GPIO_Config(
    GpioConfig_In * in
)
```

5.2.6.123 function TPM2_NTC2_PreConfig

```
WOLFTPM_API int TPM2_NTC2_PreConfig(
    NTC2_PreConfig_In * in
)
```

5.2.6.124 function TPM2_NTC2_GetConfig

```
WOLFTPM_API int TPM2_NTC2_GetConfig(
    NTC2_GetConfig_Out * out
)
```

5.2.6.125 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
}
```

5.2.6.126 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.

5.2.6.127 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.

5.2.6.128 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
}
```

5.2.6.129 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

5.2.6.130 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.

5.2.6.131 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
}
```

5.2.6.132 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
}
```

5.2.6.133 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);
```

5.2.6.134 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();
```

5.2.6.135 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
}
```

5.2.6.136 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
}
```

5.2.6.137 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
}
```

5.2.6.138 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

Example

```
int rc, nonceSize = 32;
BYTE freshNonce[32];

rc = TPM2_GetNonce(&freshNonce, nonceSize);
if (rc != TPM_RC_SUCCESS) {
    //TPM2_GetNonce failed
}
```

5.2.6.139 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);
```

5.2.6.140 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);
```

5.2.6.141 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;
}
```

5.2.6.142 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));
```

5.2.6.143 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
}
```

5.2.6.144 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
}
```

5.2.6.145 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
}
```

5.2.6.146 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

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
}
```

5.2.6.147 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
}
```

5.2.6.148 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
}
```

5.2.6.149 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
}
```

5.2.6.150 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
}
```

5.2.6.151 function TPM2_GetWolfRng

```
WOLFTPM_API int TPM2_GetWolfRng(
    WC_RNG ** rng
)
```

5.2.6.152 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();
```

5.2.6.153 function TPM2_ForceZero

```
WOLFTPM_LOCAL void TPM2_ForceZero(
    void * mem,
    word32 len
)
```

5.2.6.154 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};
length = sizeof(buffer);

TPM2_PrintBin(&buffer, length);
```

5.2.6.155 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_PrintAuthCmd(&authCmd);
```

5.2.6.156 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);
```

5.2.7 Attributes Documentation

5.2.7.1 variable C

```
C {
#endif
```

```
typedef UINT32 TPM_ALGORITHM_ID;
```

5.2.7.2 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
};
```

5.2.7.3 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
};
```

5.2.7.4 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
};
```

5.2.7.5 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
};
```

5.2.7.6 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
};
```

5.2.7.7 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
};
```

5.2.7.8 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
};
```

5.2.7.9 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
};
```

5.2.7.10 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
};
```

5.2.7.11 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
};
```

5.2.8 Source code

```
/* tpm2.h
 *
 * Copyright (C) 2006-2024 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 2 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
#define TPM_SPEC_VERSION      138
#define TPM_SPEC_YEAR         2016
#define TPM_SPEC_DAY_OF_YEAR  273

#define TPM_GENERATED_VALUE   0xff544347


typedef enum {
    TPM_ALG_ERROR           = 0x0000,
    TPM_ALG_RSA             = 0x0001,
    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_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_CTR             = 0x0040,
    TPM_ALG_OFB             = 0x0041,
    TPM_ALG_CBC             = 0x0042,
    TPM_ALG_CFB             = 0x0043,
    TPM_ALG_ECB             = 0x0044,
} 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_CURVE_T;
typedef UINT16 TPM_ECC_CURVE;

/* 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_CC_LAST                     = TPM_CC_EncryptDecrypt2,

    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
#ifdef WOLFTPM_NUVOTON
    TPM_CC_NTC2_PreConfig           = CC_VEND + 0x0211,
    TPM_CC_NTC2_GetConfig           = CC_VEND + 0x0213,
#endif
#if defined(WOLFTPM_SLB9672) || defined(WOLFTPM_SLB9673)
    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
} 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,
    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_CREATION             = 0x8021,
    TPM_ST_VERIFIED             = 0x8022,
    TPM_ST_AUTH_SECRET          = 0x8023,
    TPM_ST_HASHCHECK            = 0x8024,
    TPM_ST_AUTH_SIGNED          = 0x8025,
    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,
    TPM_CAP_LAST            = TPM_CAP_ACT,

    TPM_CAP_VENDOR_PROPERTY = 0x00000100,
} TPM_CAP_T;
typedef UINT32 TPM_CAP;

/* 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,

    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;

/* 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    = 0x00000010,
    TPMA_ALGORITHM_encrypting = 0x00000020,
    TPMA_ALGORITHM_method     = 0x00000040,
};

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_derivedDataOrigin   = 0x00000200,
    TPMA_OBJECT_noDA                = 0x00000400,
    TPMA_OBJECT_encryptedDuplication= 0x00000800,
```

```
    TPMA_OBJECT_restricted          = 0x00010000,
    TPMA_OBJECT_decrypt             = 0x00020000,
    TPMA_OBJECT_sign                = 0x00040000,
};

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,
};


/* 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;

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;


/* 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_MIN];
} 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;
    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;


/* 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;

/* 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;


/* 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 */
    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 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 */
} 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 union TPMU_SCHEME_KEYEDHASH {
    TPMS_SCHEME_HMAC hmac;
} 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;

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;
} 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 */
} 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;
} 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;

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;
} 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;
} 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;

    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)

/* 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 {
    int fd;
};
#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

#define WOLFTPM_IS_COMMAND_UNAVAILABLE(code) ((code) ==
 ↪  (int)TPM_RC_COMMAND_CODE || (code) == (int)TPM_E_COMMAND_BLOCKED)
#else
#define WOLFTPM_IS_COMMAND_UNAVAILABLE(code) (code == (int)TPM_RC_COMMAND_CODE)
#endif /* WOLFTPM_WINAPI */

/* 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

#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
#ifndef WOLFTPM2_NO_WOLFCRYPT
#ifndef WOLFTPM_NO_LOCK
    wolfSSL_Mutex hwLock;
    int lockCount;
#endif
    #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 WOLFTPM_NO_LOCK
    unsigned int hwLockInit:1;
    #endif
    #ifndef WC_NO_RNG
    unsigned int rngInit:1;
    #endif
#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);


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;
    UINT16 testResult; /* TPM_RC */
} 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);


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);

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_SLB9672) || defined(WOLFTPM_SLB9673) || \
    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 */


/* 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 */


/* 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);

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);

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 const char* TPM2_GetRCString(int rc);

WOLFTPM_API const char* TPM2_GetAlgName(TPM_ALG_ID alg);

WOLFTPM_API int TPM2_GetCurveSize(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_t;


WOLFTPM_API UINT16 TPM2_GetVendorID(void);


/* Internal helper API for ensuring memory is forcefully zero'd */
WOLFTPM_LOCAL void TPM2_ForceZero(void* mem, 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__ */
```

5.3 wolftpm/tpm2_wrap.h

5.3.1 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	**WOLFTPM2_BUFFER**
struct	**WOLFTPM2_CAPS**

	Name
struct	**TpmCryptoDevCtx**

5.3.2 Types

	Name
enum	**WOLFTPM2_MFG** { TPM_MFG_UNKNOWN = 0, TPM_MFG_INFINEON, TPM_MFG_STM, TPM_MFG_MCHP, TPM_MFG_NUVOTON, TPM_MFG_NATIONTECH}
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 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**

5.3.3 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.
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.

	Name
WOLFTPM_API int	**wolfTPM2_GetHandles * handles)Gets a list of handles.
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.
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_CreateAuthSession_EkPolicy * tpmSession)Creates a TPM session with Policy Secret to satisfy the default EK policy.
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_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 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.

	Name
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_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_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_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_RsaDecrypt padScheme, const byte * in, int inSz, byte * msg, int * msgSz)Perform RSA decryption using a TPM 2.0 key.
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_ExtendPCR * dev, int pcrIndex, int hashAlg, const byte * digest, int digestLen)Extend a PCR register with a user provided digest.
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_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.
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.

	Name
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)Deinitializes wolfTPM and wolfcrypt(if enabled)
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**(WOLFTPM2_DEV * key, const byte * in, byte * out, word32 inOutSz, byte * iv, word32 ivSz, int isDecrypt)
WOLFTPM_API int	**wolfTPM2_EncryptDecrypt**(WOLFTPM2_DEV * key, const byte * in, byte * out, word32 inOutSz, byte * iv, word32 ivSz, int isDecrypt)

	Name
WOLFTPM_API int	**wolfTPM2_SetCommand commandCode, int enableFlag)Vendor specific TPM command, used to enable other restricted TPM commands.
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.
WOLFTPM_API int	**wolfTPM2_GetKeyTemplate_ECC_EK * publicTemplate)Prepares a TPM public template for generating the TPM Endorsement Key of ECC type.

	Name
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)
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.

	Name
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 heirarchy 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_LOCAL 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)

	Name
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.	
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.

	Name
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_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_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.

	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)Calculate hash of firmware manifest for upgrade.
WOLFTPM_API int	**wolfTPM2_FirmwareUpgrade cb, void * cb_ctx)Perform TPM firmware upgrade.
WOLFTPM_API int	**wolfTPM2_FirmwareUpgradeRecover cb, void * cb_ctx)Recover from failed TPM firmware upgrade.
WOLFTPM_API int	**wolfTPM2_FirmwareUpgradeCancel * dev)Cancel ongoing TPM firmware upgrade.

5.3.4 Attributes

Name
C

5.3.5 Types Documentation

5.3.5.1 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		

5.3.5.2 typedef WOLFTPM2_SESSION

```
typedef struct WOLFTPM2_SESSION WOLFTPM2_SESSION;
```

5.3.5.3 typedef WOLFTPM2_DEV

```
typedef struct WOLFTPM2_DEV WOLFTPM2_DEV;
```

5.3.5.4 typedef WOLFTPM2_KEY

```
typedef struct WOLFTPM2_KEY WOLFTPM2_KEY;
```

5.3.5.5 typedef WOLFTPM2_PKEY

```
typedef struct WOLFTPM2_PKEY WOLFTPM2_PKEY;
```

5.3.5.6 typedef WOLFTPM2_KEYBLOB

```
typedef struct WOLFTPM2_KEYBLOB WOLFTPM2_KEYBLOB;
```

5.3.5.7 typedef WOLFTPM2_HASH

```
typedef struct WOLFTPM2_HASH WOLFTPM2_HASH;
```

5.3.5.8 typedef WOLFTPM2_NV

```
typedef struct WOLFTPM2_NV WOLFTPM2_NV;
```

5.3.5.9 typedef WOLFTPM2_HMAC

```
typedef struct WOLFTPM2_HMAC WOLFTPM2_HMAC;
```

5.3.5.10 typedef WOLFTPM2_CSR

```
typedef struct WOLFTPM2_CSR WOLFTPM2_CSR;
```

5.3.5.11 typedef WOLFTPM2_BUFFER

```
typedef struct WOLFTPM2_BUFFER WOLFTPM2_BUFFER;
```

5.3.5.12 typedef WOLFTPM2_MFG

```
typedef enum WOLFTPM2_MFG WOLFTPM2_MFG;
```

5.3.5.13 typedef WOLFTPM2_CAPS

```
typedef struct WOLFTPM2_CAPS WOLFTPM2_CAPS;
```

5.3.5.14 typedef TpmCryptoDevCtx

```
typedef struct TpmCryptoDevCtx TpmCryptoDevCtx;
```

5.3.5.15 typedef wolfTPM2FwDataCb

```
typedef int(* wolfTPM2FwDataCb) (uint8_t *data, uint32_t data_req_sz, uint32_t
 ↪  offset, void *cb_ctx);
```

5.3.6 Functions Documentation

5.3.6.1 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

5.3.6.2 function wolfTPM2_Init

```
WOLFTPM_API int wolfTPM2_Init(
    WOLFTPM2_DEV * dev,
    TPM2HalIoCb ioCb,
    void * userCtx
)
```

Complete initialization of a TPM.

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_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;
WOLFTPM2_DEV dev;

rc = wolfTPM2_Init(&dev, TPM2_IoCb, userCtx);
if (rc != TPM_RC_SUCCESS) {
    //wolfTPM2_Init failed
    goto exit;
}
```

5.3.6.3 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

5.3.6.4 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;
}
```

5.3.6.5 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;
}
```

5.3.6.6 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;
}
```

5.3.6.7 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;
}
```

5.3.6.8 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;
}
```

5.3.6.9 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

Example

```
int persistent_handle_count;

// get count of persistent handles
persistent_handle_count = wolfTPM2_GetHandles(PERSISTENT_FIRST, NULL);
```

5.3.6.10 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** integer value, specifying the TPM Authorization slot, between zero and three

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

5.3.6.11 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** integer value, specifying the TPM Authorization slot, between zero and three
- **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

5.3.6.12 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** integer value, specifying the TPM Authorization slot, between zero and three
- **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.

5.3.6.13 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** integer value, specifying the TPM Authorization slot, between zero and three
- **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

5.3.6.14 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** integer value, specifying the TPM Authorization slot, between zero and three

- **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.

5.3.6.15 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** integer value, specifying the TPM Authorization slot, between zero and three
- **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

5.3.6.16 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** integer value, specifying the TPM Authorization slot, between zero and three
- **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

5.3.6.17 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** integer value, specifying the TPM Authorization slot, between zero and three
- **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

5.3.6.18 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

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

5.3.6.19 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

5.3.6.20 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.

5.3.6.21 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.

5.3.6.22 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.

5.3.6.23 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

5.3.6.24 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.

5.3.6.25 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

5.3.6.26 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

5.3.6.27 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.

5.3.6.28 function wolfTPM2_LoadPublicKey_ex

```
WOLFTPM_API int wolfTPM2_LoadPublicKey_ex(
    WOLFTPM2_DEV * dev,
    WOLFTPM2_KEY * key,
    const TPM2B_PUBLIC * pub,
    TPM_HANDLE hierarchy
)
```

5.3.6.29 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 (can be NULL for external keys)
- **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

5.3.6.30 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 (can be NULL for external keys)
- **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

5.3.6.31 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

5.3.6.32 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

5.3.6.33 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 (can be NULL for external keys and the key will be imported under the OWNER hierarchy)
- **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

5.3.6.34 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 (can be NULL for external keys and the key will be imported under the OWNER hierarchy)
- **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

5.3.6.35 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 (can be NULL for external keys and the key will be imported under the OWNER hierarchy)
- **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

5.3.6.36 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 (can be NULL for external keys and the key will be imported under the OWNER hierarchy)
- **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

Return:

- TPM_RC_SUCCESS: successful
- TPM_RC_FAILURE: generic failure (check TPM IO and TPM return code)
- BAD_FUNC_ARG: check the provided arguments

5.3.6.37 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

5.3.6.38 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 (can be NULL for external keys and the key will be imported under the OWNER hierarchy)
- **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

5.3.6.39 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 (can be NULL for external keys and the key will be imported under the OWNER hierarchy)
- **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

5.3.6.40 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 (can be NULL for external keys and the key will be imported under the OWNER hierarchy)
- **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

Return:

- TPM_RC_SUCCESS: successful
- TPM_RC_FAILURE: generic failure (check TPM IO and TPM return code)
- BAD_FUNC_ARG: check the provided arguments

5.3.6.41 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

5.3.6.42 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

5.3.6.43 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

5.3.6.44 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

5.3.6.45 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

5.3.6.46 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

5.3.6.47 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

5.3.6.48 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

5.3.6.49 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

5.3.6.50 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

5.3.6.51 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

5.3.6.52 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

5.3.6.53 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

5.3.6.54 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

5.3.6.55 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)

5.3.6.56 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)

5.3.6.57 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

5.3.6.58 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

5.3.6.59 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

5.3.6.60 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

5.3.6.61 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)

5.3.6.62 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

5.3.6.63 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

5.3.6.64 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

5.3.6.65 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

5.3.6.66 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

5.3.6.67 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

5.3.6.68 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

5.3.6.69 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

5.3.6.70 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

5.3.6.71 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

5.3.6.72 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

5.3.6.73 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

5.3.6.74 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

5.3.6.75 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)

5.3.6.76 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, specifying a valid PCR index between 0 and 15

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: Only PCR registers 0-15 can be reset, and this operation requires platform authorization

5.3.6.77 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

5.3.6.78 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

5.3.6.79 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

5.3.6.80 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

5.3.6.81 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

Note: User data size should be less or equal to the NV Index maxSize specified using wolfTPM2_CreateAuth

5.3.6.82 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

5.3.6.83 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

5.3.6.84 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

Note: User data size should be less or equal to the NV Index maxSize specified using wolfTPM2_CreateAuth

5.3.6.85 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

5.3.6.86 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

5.3.6.87 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

5.3.6.88 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

5.3.6.89 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

5.3.6.90 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

5.3.6.91 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

5.3.6.92 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

5.3.6.93 function wolfTPM2_NVDelete

```
WOLFTPM_API int wolfTPM2_NVDelete(
    WOLFTPM2_DEV * dev,
    TPM_HANDLE authHandle,
    word32 nvIndex
)
```

Deprecated, use newer API.

See: wolfTPM2_NVDeleteAuth

5.3.6.94 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

5.3.6.95 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

5.3.6.96 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

5.3.6.97 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

5.3.6.98 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

5.3.6.99 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_Clear

Return:

- TPM_RC_SUCCESS: successful
- TPM_RC_FAILURE: generic failure (check TPM IO and TPM return code)
- BAD_FUNC_ARG: check the provided arguments

5.3.6.100 function wolfTPM2_Clear

```
WOLFTPM_API int wolfTPM2_Clear(
    WOLFTPM2_DEV * dev
)
```

Deinitializes wolfTPM and wolfcrypt(if enabled)

Parameters:

- **dev** pointer to a TPM2_DEV struct

See: wolfTPM2_Clear

Return:

- TPM_RC_SUCCESS: successful
- TPM_RC_FAILURE: generic failure (check TPM IO and TPM return code)
- BAD_FUNC_ARG: check the provided arguments

5.3.6.101 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

5.3.6.102 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

5.3.6.103 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

5.3.6.104 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

5.3.6.105 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

5.3.6.106 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

5.3.6.107 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

5.3.6.108 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

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

Load a symmetric key into the TPM

5.3.6.109 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
)
```

5.3.6.110 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
)
```

5.3.6.111 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

5.3.6.112 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

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: If doStartup is set, then TPM2_Startup is performed right after TPM2_Shutdown

5.3.6.113 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

5.3.6.114 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: If there are Primary Keys as transient objects, they need to be recreated before TPM keys can be used

5.3.6.115 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

5.3.6.116 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

5.3.6.117 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
- **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

5.3.6.118 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
- **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

5.3.6.119 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

5.3.6.120 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

5.3.6.121 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

5.3.6.122 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

5.3.6.123 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

5.3.6.124 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

5.3.6.125 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

5.3.6.126 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

5.3.6.127 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

5.3.6.128 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

5.3.6.129 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

5.3.6.130 function wolfTPM2_GetKeyTemplate_RSA_IAK

```
WOLFTPM_API int wolfTPM2_GetKeyTemplate_RSA_IAK(
    TPMT_PUBLIC * publicTemplate,
    int keyBits,
    TPM_ALG_ID hashAlg
)
```

5.3.6.131 function wolfTPM2_GetKeyTemplate_ECC_IAK

```
WOLFTPM_API int wolfTPM2_GetKeyTemplate_ECC_IAK(
    TPMT_PUBLIC * publicTemplate,
    TPM_ECC_CURVE curveID,
    TPM_ALG_ID hashAlg
)
```

5.3.6.132 function wolfTPM2_GetKeyTemplate_ECC_IDevID

```
WOLFTPM_API int wolfTPM2_GetKeyTemplate_ECC_IDevID(
    TPMT_PUBLIC * publicTemplate,
    TPM_ECC_CURVE curveID,
    TPM_ALG_ID hashAlg
)
```

5.3.6.133 function wolfTPM2_GetKeyTemplate_RSA_IDevID

```
WOLFTPM_API int wolfTPM2_GetKeyTemplate_RSA_IDevID(
    TPMT_PUBLIC * publicTemplate,
    int keyBits,
    TPM_ALG_ID hashAlg
)
```

5.3.6.134 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

5.3.6.135 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

5.3.6.136 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

5.3.6.137 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

5.3.6.138 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

5.3.6.139 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

5.3.6.140 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

5.3.6.141 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

5.3.6.142 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

5.3.6.143 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

5.3.6.144 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

5.3.6.145 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

5.3.6.146 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

5.3.6.147 function wolfTPM2_ChangePlatformAuth

```
WOLFTPM_API int wolfTPM2_ChangePlatformAuth(
    WOLFTPM2_DEV * dev,
    WOLFTPM2_SESSION * session
)
```

Helper to set the platform heirarchy 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

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

5.3.6.148 function wolfTPM2_EncryptSecret

```
WOLFTPM_LOCAL int wolfTPM2_EncryptSecret(
    WOLFTPM2_DEV * dev,
    const WOLFTPM2_KEY * tpmKey,
    TPM2B_DATA * secret,
    TPM2B_ENCRYPTED_SECRET * encSecret,
    const char * label
)
```

5.3.6.149 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

5.3.6.150 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

5.3.6.151 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

5.3.6.152 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
)
```

5.3.6.153 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
)
```

5.3.6.154 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
)
```

5.3.6.155 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
)
```

5.3.6.156 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
)
```

5.3.6.157 function wolfTPM_PK_SetCb

```
WOLFTPM_API int wolfTPM_PK_SetCb(
    WOLFSSL_CTX * ctx
)
```

5.3.6.158 function wolfTPM_PK_SetCbCtx

```
WOLFTPM_API int wolfTPM_PK_SetCbCtx(
    WOLFSSL * ssl,
    void * userCtx
)
```

5.3.6.159 function wolfTPM2_New

```
WOLFTPM_API WOLFTPM2_DEV * wolfTPM2_New(
    void
)
```

Allocate and initialize a WOLFTPM2_DEV.

See: wolfTPM2_Free

Return:

- pointer to new device struct
- NULL: on any error

5.3.6.160 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

5.3.6.161 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

5.3.6.162 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

5.3.6.163 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

5.3.6.164 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

5.3.6.165 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

5.3.6.166 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

5.3.6.167 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

5.3.6.168 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

5.3.6.169 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

5.3.6.170 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

5.3.6.171 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

5.3.6.172 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

5.3.6.173 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

5.3.6.174 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

5.3.6.175 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

5.3.6.176 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

5.3.6.177 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)

5.3.6.178 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

5.3.6.179 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

5.3.6.180 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

5.3.6.181 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

5.3.6.182 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

5.3.6.183 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

5.3.6.184 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

5.3.6.185 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 hash algorithm to use with pcr policy
- **pcrArray** optional array of pcrs to be used when creating the tpm object
- **pcrArraySz** length of the pcrArray
- **pcrDigest** digest for the PCR(s) collected (can get using wolfTPM2_PCRGetDigest)
- **pcrDigestSz** size of the PCR digest
- **digest** input/out digest
- **digestSz** input/out digest size

See:

- wolfTPM2_PolicyPCRMake
- wolfTPM2_PolicyAuthorizeMake
- 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

5.3.6.186 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

5.3.6.187 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

5.3.6.188 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

5.3.6.189 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

5.3.6.190 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

5.3.6.191 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
- **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: check the provided arguments

Note: Used with IAK and IDevID keys on ST33KTPM devices

5.3.6.192 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

5.3.6.193 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

5.3.6.194 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
)
```

Calculate hash of firmware manifest for upgrade.

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

5.3.6.195 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

Note: Upgrades TPM firmware using provided manifest and data callback

5.3.6.196 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

5.3.6.197 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

5.3.7 Attributes Documentation

5.3.7.1 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;
```

5.3.8 Source code

```
/* tpm2_wrap.h
 *
 * Copyright (C) 2006-2024 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 2 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 __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];
} 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;
#endif

/* 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,
} 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;
#if defined(WOLFTPM_SLB9672) || defined(WOLFTPM_SLB9673)
    word32 keyGroupId;
    word16 fwCounter;
    word16 fwCounterSame;
    byte   opMode;
#endif

    /* bits */
    word16 fips140_2 : 1; /* using FIPS mode */
    word16 cc_eal4   : 1; /* Common Criteria EAL4+ */
    word16 req_wait_state : 1; /* requires SPI wait state */
} 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);

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_API int wolfTPM2_GetHandles(TPM_HANDLE handle, TPML_HANDLE* handles);


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_CreateAuthSession_EkPolicy(WOLFTPM2_DEV* dev,
                                                    WOLFTPM2_SESSION* tpmSession);

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_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);

#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_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_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);

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_RsaDecrypt(WOLFTPM2_DEV* dev, WOLFTPM2_KEY* key,
    TPM_ALG_ID padScheme, const byte* in, int inSz, byte* msg, int* msgSz);


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_ExtendPCR(WOLFTPM2_DEV* dev, int pcrIndex, int
 ↪  hashAlg,
    const byte* digest, int digestLen);

/* 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_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_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_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);
```

```
/* 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_LOCAL 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 */
    #ifndef WOLFTPM2_USE_SW_ECDHE
    WOLFTPM2_KEY* ecdhKey; /* ECDH */
    #endif
#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 */
} TpmCryptoDevCtx;

#endif /* WOLFTPM_CRYPTOCB || HAVE_PK_CALLBACKS */

#ifdef WOLFTPM_CRYPTOCB

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);

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);

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);


/* 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);


#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_FirmwareUpgrade(WOLFTPM2_DEV* dev,
    uint8_t* manifest, uint32_t manifest_sz,
    wolfTPM2FwDataCb cb, void* cb_ctx);

WOLFTPM_API int wolfTPM2_FirmwareUpgradeRecover(WOLFTPM2_DEV* dev,
    uint8_t* manifest, uint32_t manifest_sz,
    wolfTPM2FwDataCb cb, void* cb_ctx);

WOLFTPM_API int wolfTPM2_FirmwareUpgradeCancel(WOLFTPM2_DEV* dev);

#endif /* WOLFTPM_FIRMWARE_UPGRADE */


#ifdef __cplusplus
    }  /* extern "C" */
#endif

#endif /* __TPM2_WRAP_H__ */
```

5.4 hal/tpm_io.h

5.4.1 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_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_Linux_SPI**(TPM2_CTX * ctx, const byte * txBuf, byte * rxBuf, word16 xferSz, void * userCtx)
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_Mmio**(TPM2_CTX * ctx, int isRead, word32 addr, byte * buf, word16 size, void * userCtx)

5.4.2 Attributes

Name
C

5.4.3 Functions Documentation

5.4.3.1 function TPM2_IoCb

```
WOLFTPM_API int TPM2_IoCb(
    TPM2_CTX * ctx,
    const BYTE * txBuf,
    BYTE * rxBuf,
    UINT16 xferSz,
    void * userCtx
)
```

5.4.3.2 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
)
```

5.4.3.3 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
)
```

5.4.3.4 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
)
```

5.4.3.5 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
)
```

5.4.3.6 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
)
```

5.4.3.7 function TPM2_IoCb_Atmel_SPI

```
WOLFTPM_LOCAL int TPM2_IoCb_Atmel_SPI(
    TPM2_CTX * ctx,
    const byte * txBuf,
    byte * rxBuf,
    word16 xferSz,
    void * userCtx
)
```

5.4.3.8 function TPM2_IoCb_Barebox_SPI

```
WOLFTPM_LOCAL int TPM2_IoCb_Barebox_SPI(
    TPM2_CTX * ctx,
    const byte * txBuf,
    byte * rxBuf,
    word16 xferSz,
    void * userCtx
)
```

5.4.3.9 function TPM2_IoCb_Linux_SPI

```
WOLFTPM_LOCAL int TPM2_IoCb_Linux_SPI(
    TPM2_CTX * ctx,
    const byte * txBuf,
    byte * rxBuf,
    word16 xferSz,
    void * userCtx
)
```

5.4.3.10 function TPM2_IoCb_STCubeMX_SPI

```
WOLFTPM_LOCAL int TPM2_IoCb_STCubeMX_SPI(
    TPM2_CTX * ctx,
```

```
    const byte * txBuf,
    byte * rxBuf,
    word16 xferSz,
    void * userCtx
)
```

5.4.3.11 function TPM2_IoCb_QNX_SPI

```
WOLFTPM_LOCAL int TPM2_IoCb_QNX_SPI(
    TPM2_CTX * ctx,
    const byte * txBuf,
    byte * rxBuf,
    word16 xferSz,
    void * userCtx
)
```

5.4.3.12 function TPM2_IoCb_Xilinx_SPI

```
WOLFTPM_LOCAL int TPM2_IoCb_Xilinx_SPI(
    TPM2_CTX * ctx,
    const byte * txBuf,
    byte * rxBuf,
    word16 xferSz,
    void * userCtx
)
```

5.4.3.13 function TPM2_IoCb_Infineon_SPI

```
WOLFTPM_LOCAL int TPM2_IoCb_Infineon_SPI(
    TPM2_CTX * ctx,
    const byte * txBuf,
    byte * rxBuf,
    word16 xferSz,
    void * userCtx
)
```

5.4.3.14 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
)
```

5.4.3.15 function TPM2_IoCb_Microchip_SPI

```
WOLFTPM_LOCAL int TPM2_IoCb_Microchip_SPI(
    TPM2_CTX * ctx,
    const byte * txBuf,
    byte * rxBuf,
    word16 xferSz,
    void * userCtx
)
```

5.4.3.16 function TPM2_IoCb_Mmio

```
WOLFTPM_LOCAL int TPM2_IoCb_Mmio(
    TPM2_CTX * ctx,
    int isRead,
    word32 addr,
    byte * buf,
    word16 size,
    void * userCtx
)
```

5.4.4 Attributes Documentation

5.4.4.1 variable C

```
C {
#endif




#if defined(WOLFTPM_LINUX_DEV) || defined(WOLFTPM_SWTPM) || \
    defined(WOLFTPM_WINAPI)


#define TPM2_IoCb NULL

#else

#ifdef WOLFTPM_EXAMPLE_HAL

#ifdef WOLFTPM_ADV_IO
WOLFTPM_API int TPM2_IoCb(TPM2_CTX* ctx, INT32 isRead, UINT32 addr,
    BYTE* buf, UINT16 size, void* userCtx);
```

5.4.5 Source code

```
/* tpm_io.h
 *
 * Copyright (C) 2006-2024 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 2 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_WINAPI)

/* HAL not required, so use NULL */
#define TPM2_IoCb NULL

#else

#ifdef WOLFTPM_EXAMPLE_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);
#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(__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);
#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

#endif /* WOLFTPM_EXAMPLE_HAL */
#endif /* !(WOLFTPM_LINUX_DEV || WOLFTPM_SWTPM || WOLFTPM_WINAPI) */

#ifdef __cplusplus
```

```
    }  /* extern "C" */
#endif

#endif /* _TPM_IO_H_ */
```

5.5 wolfTPM2 Wrappers

More...

5.5.1 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.
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_API int	**wolfTPM2_GetHandles * handles)Gets a list of handles.
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_CreateAuthSession_EkPolicy * tpmSession)Creates a TPM session with Policy Secret to satisfy the default EK policy.
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.
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.

	Name
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_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.

	Name
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 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.
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_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.

	Name
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_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.
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_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)

	Name
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_RsaDecrypt padScheme, const byte * in, int inSz, byte * msg, int * msgSz)Perform RSA decryption using a TPM 2.0 key.
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_ExtendPCR * dev, int pcrIndex, int hashAlg, const byte * digest, int digestLen)Extend a PCR register with a user provided digest.
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_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.

	Name
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.
WOLFTPM_API int	**wolfTPM2_Clear * dev)Deinitializes wolfTPM and wolfcrypt(if enabled)
WOLFTPM_API int	**wolfTPM2_HashStart hashAlg, const byte * usageAuth, word32 usageAuthSz)Helper function to start a TPM generated hash.

	Name
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_SetCommand commandCode, int enableFlag)Vendor specific TPM command, used to enable other restricted TPM commands.
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.

	Name
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_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.

	Name
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 heirarchy 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.

	Name
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.
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_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_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_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.

	Name
WOLFTPM_API int	**wolfTPM2_FirmwareUpgradeHash cb, void * cb_ctx)Calculate hash of firmware manifest for upgrade.
WOLFTPM_API int	**wolfTPM2_FirmwareUpgrade cb, void * cb_ctx)Perform TPM firmware upgrade.
WOLFTPM_API int	**wolfTPM2_FirmwareUpgradeRecover cb, void * cb_ctx)Recover from failed TPM firmware upgrade.
WOLFTPM_API int	**wolfTPM2_FirmwareUpgradeCancel * dev)Cancel ongoing TPM firmware upgrade.

5.5.2 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.

5.5.3 Functions Documentation

```
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.

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_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;
WOLFTPM2_DEV dev;

rc = wolfTPM2_Init(&dev, TPM2_IoCb, userCtx);
if (rc != TPM_RC_SUCCESS) {
    //wolfTPM2_Init failed
    goto exit;
}
```

```
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_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

Example

```
int persistent_handle_count;

// get count of persistent handles
persistent_handle_count = wolfTPM2_GetHandles(PERSISTENT_FIRST, NULL);
```

```
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** integer value, specifying the TPM Authorization slot, between zero and three

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** integer value, specifying the TPM Authorization slot, between zero and three
- **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** integer value, specifying the TPM Authorization slot, between zero and three
- **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** integer value, specifying the TPM Authorization slot, between zero and three
- **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** integer value, specifying the TPM Authorization slot, between zero and three
- **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** integer value, specifying the TPM Authorization slot, between zero and three
- **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** integer value, specifying the TPM Authorization slot, between zero and three
- **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** integer value, specifying the TPM Authorization slot, between zero and three
- **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

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_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_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 (can be NULL for external keys)
- **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 (can be NULL for external keys)
- **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 (can be NULL for external keys and the key will be imported under the OWNER hierarchy)
- **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

```
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 (can be NULL for external keys and the key will be imported under the OWNER hierarchy)
- **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 (can be NULL for external keys and the key will be imported under the OWNER hierarchy)
- **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 (can be NULL for external keys and the key will be imported under the OWNER hierarchy)
- **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

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_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_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 (can be NULL for external keys and the key will be imported under the OWNER hierarchy)
- **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

```
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 (can be NULL for external keys and the key will be imported under the OWNER hierarchy)

- **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 (can be NULL for external keys and the key will be imported under the OWNER hierarchy)
- **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

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_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 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_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_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

```
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_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_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_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, specifying a valid PCR index between 0 and 15

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: Only PCR registers 0-15 can be reset, and this operation requires platform authorization

```
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_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_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

```
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

Note: User data size should be less or equal to the NV Index maxSize specified using wolfTPM2_CreateAuth

```
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

```
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

```
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

Note: User data size should be less or equal to the NV Index maxSize specified using wolfTPM2_CreateAuth

```
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

```
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_Clear

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
)
```

Deinitializes wolfTPM and wolfcrypt(if enabled)

Parameters:

- **dev** pointer to a TPM2_DEV struct

See: wolfTPM2_Clear

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_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

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

Load a symmetric key into the TPM

```
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_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

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: If doStartup is set, then TPM2_Startup is performed right after TPM2_Shutdown

```
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: 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
- **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
- **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 heirarchy 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

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_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

```
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

Return:

- pointer to new device struct
- NULL: on any error

```
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_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 hash algorithm to use with pcr policy
- **pcrArray** optional array of pcrs to be used when creating the tpm object
- **pcrArraySz** length of the pcrArray
- **pcrDigest** digest for the PCR(s) collected (can get using wolfTPM2_PCRGetDigest)
- **pcrDigestSz** size of the PCR digest
- **digest** input/out digest
- **digestSz** input/out digest size

See:

- wolfTPM2_PolicyPCRMake
- wolfTPM2_PolicyAuthorizeMake
- 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_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_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
- **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: check the provided arguments

Note: Used with IAK and IDevID keys on ST33KTPM devices

```
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
)
```

Calculate hash of firmware manifest for upgrade.

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

```
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

Note: Upgrades TPM firmware using provided manifest and data callback

```
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_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

6 Cited Sources

[1.] Wikipedia contributors. (2018, May 30). Trusted Platform Module. In *Wikipedia, The Free Encyclopedia*. Retrieved 22:46, June 20, 2018, from https://en.wikipedia.org/w/index.php?title=Trusted_Platform_Module&oldid=

[2.] Arthur W., Challener D., Goldman K. (2015) Platform Configuration Registers. In: A Practical Guide to TPM 2.0. Apress, Berkeley, CA

