

wolfEngine Documentation



2023-03-22

Contents

| | | |
|-----------|--|-----------|
| 1 | イントロダクション | 3 |
| 2 | OpenSSL 版との互換性 | 5 |
| 3 | wolfEngine のビルド | 6 |
| 3.1 | wolfEngine のソースコードの取得 | 6 |
| 3.2 | wolfEngine パッケージ構成 | 6 |
| 3.3 | OpenSSL のバージョンに関する注意事項 | 6 |
| 3.4 | *nix 上でのビルド | 6 |
| 3.4.1 | OpenSSL をビルド | 6 |
| 3.4.2 | wolfSSL をビルド | 6 |
| 3.4.3 | wolfEngine をビルド | 7 |
| 3.5 | WinCE 上でのビルド | 8 |
| 3.6 | ビルドオプション (./configure に指定するオプション) | 9 |
| 3.7 | ビルド用マクロ定義 | 10 |
| 4 | FIPS 140-2 のサポート | 13 |
| 5 | Engine コントロールコマンド | 14 |
| 6 | ロギング | 15 |
| 6.1 | デバッグログの有効化/無効化 | 15 |
| 6.2 | ロギングレベルの制御 | 15 |
| 6.3 | コンポーネント単位のロギングの制御 | 16 |
| 6.4 | カスタムロギングコールバックの設定 | 17 |
| 7 | 移植性 | 18 |
| 7.1 | スレッド対応 | 18 |
| 7.2 | 動的メモリ使用 | 18 |
| 7.3 | ロギング | 18 |
| 8 | wolfEngine のロード | 19 |
| 8.1 | OpenSSL をエンジン使用可能に構成 | 19 |
| 8.2 | OpenSSL コンフィギュレーションファイルからの wolfEngine のロード | 19 |
| 8.3 | wolfEngine 静的エントリーポイント | 20 |
| 9 | wolfEngine の設計 | 21 |
| 9.1 | wolfEngine エントリーポイント | 21 |
| 9.2 | wolfEngine アルゴリズム コールバック登録 | 21 |
| 10 | オープンソース統合に関する注意事項 | 23 |
| 10.1 | cURL | 23 |
| 10.2 | stunnel | 23 |
| 10.3 | OpenSSH | 23 |
| 11 | サポートと OpenSSL バージョン追加 | 24 |

1 イントロダクション

wolfCrypt エンジン (wolfEngine) は、wolfCrypt および wolfCrypt FIPS 暗号化ライブラリを OpenSSL エンジンフレームワークに適合させるためのライブラリです。wolfEngine は、共有または静的ライブラリとして OpenSSL エンジンの実装を提供し、現在 OpenSSL を使用しているアプリケーションが FIPS および非 FIPS ユースケースで wolfCrypt 暗号化ライブラリを活用できるようにします。

wolfEngine は、wolfSSL (libwolfssl) と OpenSSL にリンクする個別のスタンドアロン ライブラリとして構成されています。wolfEngine は、wolfCrypt ネイティブ API を内部的にラップする **OpenSSL エンジンの実装** を実装および公開します。wolfEngine の概要図と、それがアプリケーションおよび OpenSSL とどのように関連しているかを下の図 1 に示します。

wolfEngine の設計とアーキテクチャの詳細については、wolfEngine の設計 の章を参照してください。

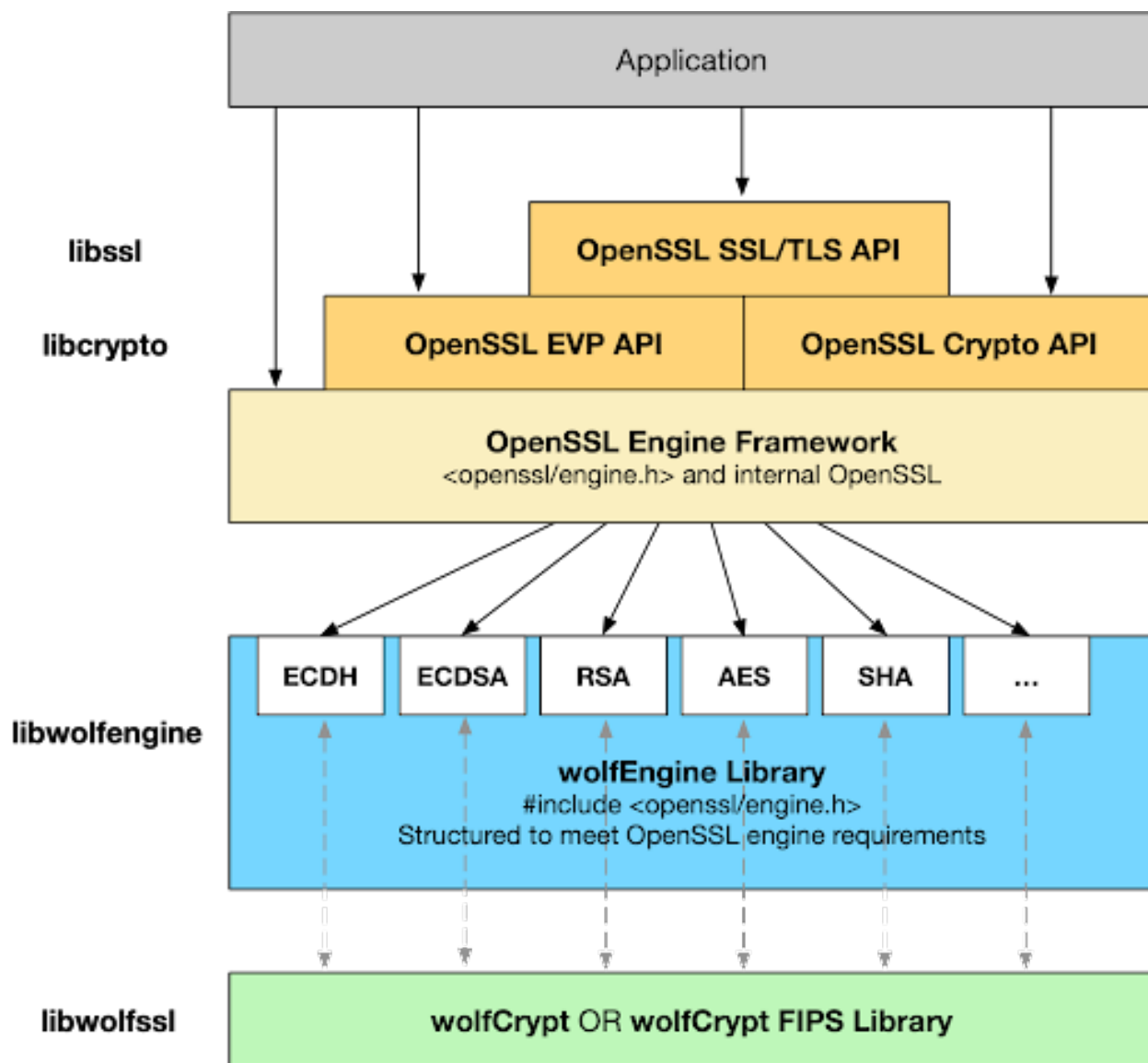


Figure 1: wolfEngine Overview

wolfEngine は、デフォルトで **libwolfengine** と呼ばれる共有ライブラリとしてコンパイルされます。これは、アプリケーションまたはコンフィギュレーションファイルを介して OpenSSL によって実行時に動的に

登録できます。wolfEngine は、アプリケーションが静的ビルドでコンパイルされたときにエンジンをロードするためのエントリ ポイントも提供します。

2 OpenSSL 版との互換性

wolfEngine は、以下のバージョンの OpenSSL に対してテストされています。wolfEngine は他のバージョンでも動作する可能性がありますが、変更や調整が必要になる場合があります：

- OpenSSL 1.0.2h
- OpenSSL 1.1.1b

wolfEngine に他の OpenSSL バージョンのサポート追加を希望される場合は、facts@wolfssl.com にご連絡ください。

3 wolfEngine のビルド

3.1 wolfEngine のソースコードの取得

wolfEngine の最新バージョンは、wolfSSL Inc. から直接入手できます。詳細については、facts@wolfssl.com までお問い合わせください。

3.2 wolfEngine パッケージ構成

一般的な wolfEngine パッケージは次のように構成されています:

| | |
|------------------|--|
| certs/ | (ユニットテストで使用されるテスト用証明書、鍵) |
| engine.conf | (wolfEngineを使用する場合のOpenSSLコンフィギュレーション ファイルサンプル) |
| include/ | |
| wolfengine/ | (wolfEngineヘッダーファイル) |
| openssl_patches/ | |
| 1.0.2h/tests/ | (OpenSSL 1.0.2h テストアプリ用パッチ) |
| 1.1.1b/tests/ | (OpenSSL 1.1.1b テストアプリ用パッチ) |
| scripts/ | (wolfEngine テストスクリプト) |
| src/ | (wolfEngine ソースファイル) |
| test/ | (wolfEngine テストファイル) |
| user_settings.h | (user_settings.hサンプル) |

3.3 OpenSSL のバージョンに関する注意事項

wolfEngine で使用されている OpenSSL のバージョンに応じて、次のようないくつかのアルゴリズム サポートの注意事項があります:

- SHA-3 は OpenSSL versions 1.1.1 以降でサポートされます
- EC_KEY_METHOD は OpenSSL versions 1.1.1 以降でサポートされます

3.4 *nix 上でのビルド

3.4.1 OpenSSL をビルド

OpenSSL のプリインストールされたバージョンを wolfEngine で使用することも (上記のアルゴリズムの警告を除いて)、または OpenSSL を再コンパイルして wolfEngine で使用することもできます。*nix のようなプラットフォームで OpenSSL をコンパイルするための一般的な手順は、次のようになります。完全に包括的な OpenSSL のビルド手順については、OpenSSL INSTALL ファイルとドキュメントを参照してください。

```
git clone https://github.com/openssl/openssl.git
cd openssl
./config no-fips -shared
make
sudo make install
```

3.4.2 wolfSSL をビルド

wolfEngine で wolfSSL の FIPS 検証済みバージョンを使用する場合は、特定の FIPS 検証済みソースバンドルとセキュリティポリシーで提供されるビルド手順に従ってください。正しい「-enable-fips」設定オプションに加えて、wolfEngine は **“WOLFSSL_PUBLIC_MP”** が定義された状態で wolfSSL をコンパイルする必要があります。たとえば、Linux で「wolfCrypt Linux FIPsv2」バンドルをビルドする場合:

```

cd wolfssl-X.X.X-commercial-fips-linuxv
./configure **--enable-fips=v2 CFLAGS=" -DWOLFSSL_PUBLIC_MP " **
make
./wolfcrypt/test/testwolfcrypt
#< fips_test.c 内の verifyCore を hash output from testwolfcrypt スクリプトが出力するハッシュ値に更新してください >--

make
./wolfcrypt/test/testwolfcrypt

#< 全アルゴリズムでパスするはずです>--

sudo make install

```

wolfEngine で使用する非 FIPS wolfSSL をビルドするには:

```

cd wolfssl-X.X.X

./configure --enable-cmac --enable-keygen --enable-sha --enable-des --enable-aesctr --enable-aesccm --enable-x963kdf CPPFLAGS="-DHAVE_AES_ECB -DWOLFSSL_AES_DIRECT -DWC_RSA_NO_PADDING -DWOLFSSL_PUBLIC_MP -DECC_MIN_KEY_SZ=192 -DWOLFSSL_PSS_LONG_SALT -DWOLFSSL_PSS_SALT_LEN_DISCOVER"
"

make
sudo make install

```

GitHub から wolfSSL をクローンする場合、./configure を実行する前に autogen.sh スクリプトを実行する必要があります。これにより、configure スクリプトが生成されます:

```
./autogen.sh
```

3.4.3 wolfEngine をビルド

Linux またはその他の *nix ライクなシステムで wolfEngine をビルドする場合は、autoconf システムを使用してください。wolfEngine を構成およびコンパイルするには、wolfEngine ルート ディレクトリから次の 2 つのコマンドを実行します:

```
./configure
make
```

GitHub から wolfEngine を取得してビルドする場合は、configure を実行する前に autogen.sh を実行します:

```
./autogen.sh
```

任意の数のビルドオプションを./configure に追加できます。利用可能なビルド オプションのリストについては、以下の「ビルド オプション」セクションを参照するか、次のコマンドを実行して、./configure スクリプトに渡す利用可能なビルド オプションのリストを表示してください:

```
./configure --help
```

“-with-openssl” オプションで変更しない限り、wolfEngine はシステムのデフォルトの OpenSSL ライブラリのインストールを使用します:

```
./configure --with-openssl=/usr/local/ssl
```

カスタム OpenSSL のインストール場所も、ライブラリ検索パスに追加する必要がある場合があります。Linux では、LD_LIBRARY_PATH が使用されます:

```
export LD_LIBRARY_PATH=/usr/local/ssl:$LD_LIBRARY_PATH
```

wolfEngine をビルドしてインストールするには、以下を実行します:

```
make  
make install
```

インストールにはスーパーユーザー権限が必要な場合があります。その場合は、コマンドの前に sudo を付けます:

```
sudo make install
```

ビルドをテストするには、ルート wolfEngine ディレクトリからビルトインテストを実行します:

```
./test/unit.test
```

または autoconf を使用してテストを実行します:

```
make check
```

error while loading shared libraries: libssl.so.3 のようなエラーが発生した場合は、ライブラリが見つからなかった為です。上記のセクションで説明したように、LD_LIBRARY_PATH 環境変数を使用します:

3.5 WinCE 上でのビルド

wolfEngine との完全な互換性のために、wolfCrypt の user_settings.h ファイルに以下の定義があることを確認してください:

```
#define WOLFSSL_CMAC  
#define WOLFSSL_KEY_GEN  
#undef NO_SHA  
#undef NO_DES  
#define WOLFSSL_AES_COUNTER  
#define HAVE_AESCCM  
#define HAVE_AES_ECB  
#define WOLFSSL_AES_DIRECT  
#define WC_RSA_NO_PADDING  
#define WOLFSSL_PUBLIC_MP  
#define ECC_MIN_KEY_SZ=192
```

使用するアルゴリズムと機能に応じて、user_settings.h ファイルに wolfEngine フラグを追加します。wolfEngine のディレクトリにある user_settings.h ファイルで、wolfEngine ユーザー設定フラグが参照できます。

Windows CE 用の wcecompat、wolfCrypt、および OpenSSL をビルドし、それらのパスを参照できるようにします。

wolfEngine ディレクトリでソースファイルを開き、OpenSSL、wolfCrypt、および user_settings.h パスを使用しているディレクトリに変更します。INCLUDES セクションと TARGETLIBS セクションのパスを更新する必要があります。

Visual Studio で wolfEngine プロジェクトをロードします。ベンチマークまたは単体テストを実行するかどうかに応じて、「bench.c」、または「unit.h」と「unit.c」のいずれかを含めます。

プロジェクトをビルドすると、wolfEngine.exe 実行可能ファイルが作成されます。この実行可能ファイルを -help で実行すると、オプションの完全なリストが表示されます。wolfEngine を静的エンジンとして使用するには、--static フラグを付けて実行する必要があります。

3.6 ビルドオプション (./configure に指定するオプション)

以下は、wolfEngine ライブラリの構築方法をカスタマイズする目的で ./configure スクリプトに追加できるオプションです。

デフォルトでは、wolfEngine は共有ライブラリのみを構築し、静的ライブラリの構築は無効になっています。これにより、ビルド時間が2倍速くなります。どちらのモードも、必要に応じて明示的に無効または有効にすることができます。

| オプション | デフォルト | 意味 |
|------------------------|-----------|---|
| -enable-static | 無効 | スタティックライブラリとしてビルド |
| -enable-shared | 有効 | 共有ライブラリとしてビルド |
| -enable-debug | 無効 | wolfEngine のデバッグ出力を有効にする |
| -enable-coverage | 無効 | コードカバレッジレポートを作成する用ビルド |
| -enable-usersettings | 無効 | user_settings.h を使用し Makefile の CFLAGS を使用しない |
| -enable-dynamic-engine | 有効 | wolfEngine をダイナミックエンジンとしてロードする |
| -enable-singlethreaded | 無効 | wolfEngine をシングルスレッド環境で使用する |
| -enable-digest | 有効 | ダイジェストの生成に wc_Hash API を使用する |
| -enable-sha | 有効 | SHA-1 を有効にする |
| -enable-sha224 | 有効 | SHA2-224 を有効にする |
| -enable-sha256 | 有効 | SHA2-256 を有効にする |
| -enable-sha384 | 有効 | SHA2-384 を有効にする |
| -enable-sha512 | 有効 | SHA2-512 を有効にする |
| -enable-sha3 | 有効 | SHA3 を有効にする |
| -enable-sha3-224 | 有効 | SHA3-224 を有効にする |
| -enable-sha3-256 | 有効 | SHA3-256 を有効にする |
| -enable-sha3-384 | 有効 | SHA3-384 を有効にする |
| -enable-sha3-512 | 有効 | SHA3-512 を有効にする |
| -enable-cmac | 有効 | CMAC を有効にする |
| -enable-hmac | 有効 | HMAC を有効にする |
| -enable-des3cbc | 有効 | 3DES-CBC を有効にする |
| -enable-aesecb | 有効 | AES-ECB を有効にする |
| -enable-aescbc | 有効 | AES-CBC を有効にする |
| -enable-aesctr | 有効 | AES-CTR を有効にする |
| -enable-aesgcm | 無効 | AES-GCM を有効にする |
| -enable-aesccm | 無効 | AES-CCM を有効にする |
| -enable-rand | 有効 | RAND を有効にする |
| -enable-rsa | 有効 | RSA を有効にする |
| -enable-dh | 有効 | DH を有効にする |
| -enable-ec-pkey | 有効 | EVP_PKEY APIs を有効にする |
| -enable-ec-key | 有効 | ECC using EC_KEY を有効にする |
| -enable-ecdsa | 有効 | ECDSA を有効にする |
| -enable-ecdh | 有効 | ECDH を有効にする |
| -enable-eckg | 有効 | EC Key Generation を有効にする |
| -enable-p192 | 有効 | EC Curve P-192 を有効にする |
| -enable-p224 | 有効 | EC Curve P-224 を有効にする |
| -enable-p256 | 有効 | EC Curve P-256 を有効にする |

| オプション | デフォルト | 意味 |
|-------------------|-------|--|
| -enable-p384 | 有効 | EC Curve P-384 を有効にする |
| -enable-p521 | 有効 | EC Curve P-521 を有効にする |
| -with-openssl=DIR | | OpenSSL のインストール場所を指定。指定しない場合はシステムのデフォルトライブラリパスとインクルードパスが使われます。 |

3.7 ビルド用マクロ定義

wolfEngine は、ユーザーが wolfEngine の構築方法を設定できるようにするいくつかのプリプロセッサマクロを公開しています。これらについては、次の表で説明します：

| マクロ定義 | 意味 |
|----------------------|--|
| WOLFENGINE_DEBUG | デバッグ シンボル、最適化レベル、デバッグロギングを使用して wolfEngine をビルドします |
| WE_NO_DYNAMIC_ENGINE | wolfEngine をダイナミックエンジンとしてビルドしない。ダイナミックエンジンとは OpenSSL が実行時に動的にロードするエンジンです。 |
| WE_SINGLE_THREADED | wolfEngine をシングルスレッドモードでビルドする。このマクロ定義によりグローバルリソースの使用の排他用に内部的に使用するロック機構を取り除きます。 |
| WE_USE_HASH | ハッシュアルゴリズムを wc_Hash API を使って有効にする |
| WE_HAVE_SHA1 | SHA-1 を有効にする |
| WE_HAVE_SHA224 | SHA-2 224 を有効にする |
| WE_HAVE_SHA256 | SHA-2 256 を有効にする |
| WE_HAVE_SHA384 | SHA-2 384 を有効にする |
| WE_HAVE_SHA512 | SHA-2 512 を有効にする |
| WE_SHA1_DIRECT | SHA-1 を wc_Sha API を使って有効にする。WE_USE_HASH とは互換性ではない |

| マクロ定義 | 意味 |
|------------------|--|
| WE_SHA224_DIRECT | SHA-2 224 を wc_Sha224 API を使って有効にする。 |
| WE_SHA256_DIRECT | SHA-2 256 を wc_Sha256 API を使って有効にする。 |
| WE_HAVE_SHA3_224 | WE_USE_HASH とはコンパチブルではない SHA-3 224 を有効にする (OpenSSL 1.0.2 では利用不可) |
| WE_HAVE_SHA3_256 | SHA-3 256 を有効にする (OpenSSL 1.0.2 では利用不可) |
| WE_HAVE_SHA3_384 | SHA-3 384 を有効にする (OpenSSL 1.0.2 では利用不可) |
| WE_HAVE_SHA3_512 | SHA-3 512 を有効にする (OpenSSL 1.0.2 では利用不可) |
| WE_HAVE_EVP_PKEY | EVP_PKEY API を使用する機能を有効にする (RSA, DH 等も含む) |
| WE_HAVE_CMAC | CMAC を有効にする |
| WE_HAVE_HMAC | HMAC を有効にする |
| WE_HAVE_DES3CBC | DES3-CBC を有効にする |
| WE_HAVE_AESECB | AES-ECB を有効にする |
| WE_HAVE_AESCBC | AES-CBC を有効にする |
| WE_HAVE_AESCTR | AES-countee mode を有効にする |
| WE_HAVE_AESGCM | AES-GCM を有効にする |
| WE_HAVE_AESCCM | AES-CCM を有効にする |
| WE_HAVE_RANDOM | wolfCrypt の疑似乱数生成実装を有効にする |
| WE_HAVE_RSA | RSA 操作 (すなわち 署名, 検証, 鍵生成等) を有効にする |
| WE_HAVE_DH | Diffie-Hellman 操作 (すなわち 鍵生成, 共有シークレット計算等) を有効にする |
| WE_HAVE_ECC | 楕円曲線暗号を有効にする |
| WE_HAVE_EC_KEY | EC_KEY_METHOD のサポートを有効にする (OpenSSL 1.0.2 では利用不可) |
| WE_HAVE_ECDSA | ECDSA を有効にする |

| マクロ定義 | 意味 |
|--------------------------|--|
| WE_HAVE_ECDH | EC Diffie-Hellman operations を有効にする |
| WE_HAVE_ECKEYGEN | EC key generation を有効にする |
| WE_HAVE_EC_P192 | EC Curve P192 を有効にする |
| WE_HAVE_EC_P224 | EC Curve P224 を有効にする |
| WE_HAVE_EC_P256 | EC Curve P256 を有効にする |
| WE_HAVE_EC_P384 | EC Curve P384 を有効にする |
| WE_HAVE_EC_P512 | EC Curve P512 を有効にする |
| WE_HAVE_DIGEST | ダイジェストアルゴリズムをベンチマークとユニットテストのコードに含めてコンパイルする |
| WOLFENGINE_USER_SETTINGS | ユーザーの指定した定義を user_settings.h ファイルから読み込む |

4 FIPS 140-2 のサポート

wolfEngine は、FIPS で検証されたバージョンの wolfCrypt に対してコンパイルされた場合に、FIPS 140-2 で検証されたバージョンの wolfCrypt で動作するように設計されています。この使用シナリオには、wolfSSL Inc. から入手した、適切にライセンスされ、検証されたバージョンの wolfCrypt が必要です。

wolfCrypt FIPS ライブラリは、非 FIPS モードに「切り替える」ことができないことに注意してください。wolfCrypt FIPS と通常の wolfCrypt は、2 つの別個のソース コード パッケージです。

wolfEngine が wolfCrypt FIPS を使用するようにコンパイルされると、FIPS で検証されたアルゴリズム、モード、およびキー サイズのサポートおよび登録エンジン コールバックのみが含まれます。OpenSSL ベースのアプリケーションが非 FIPS 検証済みアルゴリズムを呼び出す場合、実行は wolfEngine に入らず、OpenSSL 構成に基づいて、デフォルトの OpenSSL エンジンまたは他の登録済みエンジン プロバイダーによって処理される可能性があります。

注: FIPS 準拠を対象としており、wolfCrypt 以外の FIPS アルゴリズムが別のエンジンから呼び出される場合、それらのアルゴリズムは wolfEngine および wolfCrypt FIPS であり、FIPS で検証されていない可能性があります。

wolfCrypt FIPS (140-2 / 140-3) の使用に関する詳細については、wolfSSL (facts@wolfssl.com) までお問い合わせください。

5 Engine コントロールコマンド

wolfEngine は、アプリケーションが wolfEngine の動作を変更したり、内部設定を調整したりできるように、いくつかのエンジン制御コマンドを公開しています。現在、次の制御コマンドがサポートされています:

| コントロールコマンド | 意味 | 設定値 |
|----------------|---------------------------|--|
| enable_debug | wolfEngine のデバッグ出力を有効にします | 1 = 有効, 0 = 無効。詳細は Chapter 6 を参照 |
| log_level | ロギングレベルを設定します | "include/wolfengine/we_logging.h" ファイルに定義された wolfEngine_LogType のビットマスク値。詳細は Chapter 6 を参照 |
| log_components | ログ出力するコンポーネントを指定します | "include/wolfengine/we_logging.h" ファイルに定義された wolfEngine_LogComponents のビットマスク値。詳細は Chapter 6 を参照 |
| set_logging_cb | ロギングコールバックをセットします | ログメッセージの出力に使用される関数への関数ポインタ。関数は、we_logging.h の wolfEngine_Logging_cb プロトタイプと一致する必要があります。詳細は Chapter 6 を参照 |

エンジン制御コマンドは、OpenSSL の ENGINE_ctrl_cmd() API を使用して設定できます。たとえば、デバッグ ロギングを有効にするには、次のように呼び出します:

```
int ret = 0;
ret = ENGINE_ctrl_cmd(e, "enable_debug", 1, NULL, NULL, 0);
if (ret != 1) {
    printf("Failed to enable debug logging\n");
}
```

一部の制御コマンドは、OpenSSL 構成ファイルを介して設定することもできます。OpenSSL エンジン制御コマンドの使用法に関するその他のドキュメントは、OpenSSL の man ページにあります:

<https://www.openssl.org/docs/man1.0.2/man3/engine.html>

https://www.openssl.org/docs/man1.1.1/man3/ENGINE_ctrl_cmd.html

https://www.openssl.org/docs/man1.1.1/man3/ENGINE_ctrl_cmd_string.html

6 ロギング

wolfEngine は、情報提供とデバッグを目的としたログ メッセージの出力をサポートしています。デバッグ ロギングを有効にするには、最初にデバッグ サポートを有効にして wolfEngine をコンパイルする必要があります。Autoconf を使用している場合、これは ./configure に --enable-debug オプションを使用して行われます：

```
./configure --enable-debug
```

Autoconf/configure を使用しない場合は、wolfEngine ライブラリをコンパイルするときに WOLFENGINE_DEBUG を定義します。

6.1 デバッグログの有効化/無効化

デバッグ サポートがライブラリにコンパイルされたら、セクション 5 で指定された wolfEngine コントロール コマンドを使用して実行時にデバッグを有効にする必要があります。0" を指定すると、ロギングが無効になります。ENGINE_ctrl_cmd() API を使用してロギングを有効にするには：

```
int ret = 0;
ret = ENGINE_ctrl_cmd(e, "enable_debug", 1, NULL, NULL, 0);
if (ret != 1) {
    printf("Failed to enable debug logging\n");
}
```

wolfEngine がデバッグ サポートを有効にしてコンパイルされていない場合、ENGINE_ctrl_cmd() で enable_debug を設定しようとすると失敗 (0) が返されます。

6.2 ロギングレベルの制御

wolfEngine は以下のロギング レベルをサポートします。これらは、“include/wolfengine/we_logging.h” ヘッダー ファイルで、wolfEngine_LogType enum の一部として定義されています：

| ロギングレベル | 意味 | レベル値 |
|----------------------|-------------------------------|---|
| WE_LOG_ERROR | エラーをロギングする | 0x0001 |
| WE_LOG_ENTER | 関数に入った際にロギングする | 0x0002 |
| WE_LOG_LEAVE | 関数を抜ける際にロギングする | 0x0004 |
| WE_LOG_INFO | 情報提供のメッセージをロギングする | 0x0008 |
| WE_LOG_VERBOSE | 暗号化/復号のデータを含めた詳細ログ | 0x0010 |
| WE_LOG_LEVEL_DEFAULT | デフォルトのログレベル (VERBOSE 以外を全て含む) | WE_LOG_ERROR WE_LOG_ENTER WE_LOG_LEAVE WE_LOG_INFO |
| WE_LOG_LEVEL_ALL | 全ログレベルが有効 | WE_LOG_ENTER WE_LOG_LEAVE WE_LOG_INFO WE_LOG_VERBOSE |

デフォルトの wolfEngine ロギング レベルには、“WE_LOG_ERROR”、“WE_LOG_ENTER”、“WE_LOG_LEAVE”、および “WE_LOG_INFO” が含まれます。これには、詳細ログ (WE_LOG_VERBOSE) を除くすべてのログ レベルが含まれます。

ログ レベルは、ENGINE_ctrl_cmd() API または OpenSSL 構成ファイル設定のいずれかを介して、実行時に “log_level” エンジン制御コマンドを使用して制御できます。たとえば、“log_level” 制御コマンドを使用してエラー ログと情報ログのみを有効にするには、アプリケーションで次のように呼び出します：

```
#include <wolfengine/we_logging.h>

ret = ENGINE_ctrl_cmd(e, "log_level", WE_LOG_ERROR | WE_LOG_INFO,
NULL, NULL, 0);
if (ret != 1) {
    printf("Failed to set logging level\n");
}
```

6.3 コンポーネント単位のロギングの制御

wolfEngine では、コンポーネントごとにログを記録できます。コンポーネントは `include/wolfengine/we_logging.h` の `wolfEngine_LogComponents` 列挙で定義されます:

| ログ対象コンポーネント | 意味 | コンポーネントを示す値 |
|---------------------------|----------------------------|--|
| WE_LOG_RNG | ランダム数生成コンポーネント | 0x0001 |
| WE_LOG_DIGEST | ダイジェストコンポーネント (SHA-1/2/3) | 0x0002 |
| WE_LOG_MAC | MAC 機能コンポーネント (HMAC, CMAC) | 0x0004 |
| WE_LOG_CIPHER | 暗号化コンポーネント (AES, 3DES) | 0x0008 |
| WE_LOG_PK | 公開鍵コンポーネント (RSA, ECC) | 0x0010 |
| WE_LOG_KEY | 鍵合意コンポーネント (DH, ECDH) | 0x0020 |
| WE_LOG_ENGINE | エンジン特有 | 0x0040 |
| WE_LOG_COMPONENTS_ALL | 全コンポーネント | WE_LOG_RNG WE_LOG_DIGEST WE_LOG_MAC WE_LOG_CIPHER WE_LOG_PK WE_LOG_KEY WE_LOG_ENGINE |
| WE_LOG_COMPONENTS_DEFAULT | デフォルトコンポーネント (all). | WE_LOG_COMPONENTS_ALL |

デフォルトの wolfEngine ロギング構成は、すべてのコンポーネントをログに記録します (WE_LOG_COMPONENTS_DEFAULT)

ログに記録されたコンポーネントは、ENGINE_ctrl_cmd() API または OpenSSL 構成ファイル設定のいずれかを介して、実行時に “log_components” エンジン制御コマンドを使用して制御できます。たとえば、Digest および Cipher アルゴリズムのロギングのみをオンにするには、次のようにします：

```
#include <wolfengine/we_logging.h>

ret = ENGINE_ctrl_cmd(e, “ log_components ” , WE_LOG_DIGEST | WE_LOG_CIPHER,
NULL, NULL, 0);
if (ret != 1) {
    printf(“ Failed to set log components\n ” );
}
```

6.4 カスタムロギングコールバックの設定

デフォルトでは、wolfEngine は **fprintf()** を使用してデバッグ ログ メッセージを **stderr** に出力します。

ログ メッセージの出力方法や出力場所をより詳細に制御したいアプリケーションは、カスタム ロギング コールバックを記述して wolfEngine に登録できます。ロギング コールバックは、include/wolfengine/we_logging.h の wolfEngine_Logging_cb のプロトタイプと一致する必要があります：

```
/**
 * wolfEngine logging callback.
 * logLevel - [IN] - Log level of message
 * component - [IN] - Component that log message is coming from
 * logMessage - [IN] - Log message
 */
typedef void (*wolfEngine_Logging_cb)(const int logLevel, const int component,
    const char *const logMessage);
```

その後、“set_logging_cb” エンジン制御コマンドを使用して、コールバックを wolfEngine に登録できます。たとえば、ENGINE_ctrl_cmd() API を使用してカスタム ロギング コールバックを設定するには、次のようにします：

```
void customLogCallback(const int logLevel, const int component,
const char* const logMessage)
{
    (void)logLevel;
    (void)component;
    fprintf(stderr, “ wolfEngine log message: %d\n” , logMessage);
}

int **main** (void)
{
    int ret;
    ENGINE* e;
    ...
    ret = ENGINE_ctrl_cmd(e, “ set_logging_cb ” , 0, NULL, (void*)(void))
        my_Logging_cb, 0);
    if (ret != 1) {
        /* failed to set logging callback */
    }
    ...
}
```

7 移植性

wolfEngine は、関連する wolfCrypt および OpenSSL ライブラリの移植性を活用するように設計されています。

7.1 スレッド対応

wolfEngine はスレッドセーフであり、必要に応じて wolfCrypt(wc_LockMutex(), wc_UnLockMutex()) のミューテックス ロック メカニズムを使用します。wolfCrypt には、サポートされているプラットフォーム用に抽象化されたミューテックス操作があります。

7.2 動的メモリ使用

wolfEngine は、OpenSSL のメモリ割り当て関数を使用して、OpenSSL の動作との一貫性を維持します。wolfEngine の内部で使用される割り当て関数には、“OPENSSL_malloc()”、“OPENSSL_free()”、“OPENSSL_zalloc()”、および “OPENSSL_realloc()” が含まれます。

7.3 ロギング

wolfEngine はデフォルトで fprintf() 経由で stderr にログを記録します。アプリケーションは、カスタム ロギング関数を登録することでこれをオーバーライドできます (第 6 章を参照)。

ログの動作を調整するために wolfEngine をコンパイルするときに定義できる追加のマクロには、次のものがあります：

WOLFENGINE_USER_LOG - ログ出力の関数名を定義するマクロ。ユーザーは、これを fprintf の代わりに使用するカスタム ログ関数に定義できます

WOLFENGINE_LOG_PRINTF - fprintf(stderr) を切り替えて、代わりに printf(stdout) を使用するように定義します。WOLFENGINE_USER_LOG またはカスタム ロギング コールバックを使用している場合は適用されません。

8 wolfEngine のロード

8.1 OpenSSL をエンジン使用可能に構成

アプリケーションが OpenSSL エンジンを使用および使用する方法に関するドキュメントについては、OpenSSL のドキュメントを参照してください：

[OpenSSL 1.0.2](#) [OpenSSL 1.1.1](#)

アプリケーションがエンジンの使用を消費、登録、および構成するために選択できる方法はいくつかあります。最も単純な使用方法では、OpenSSL にバンドルされているすべての ENGINE 実装をロードして登録するには、アプリケーションで次を呼び出す必要があります (上記の OpenSSL ドキュメントから引用)：

```
/* For OpenSSL 1.0.2, need to make the "dynamic" ENGINE available */
ENGINE_load_dynamic();

/* Load all bundled ENGINEs into memory and make them visible */
ENGINE_load_builtin_engines();

/* Register all of them for every algorithm they collectively implement */
ENGINE_register_all_complete();
```

この時点で、アプリケーションが OpenSSL 構成ファイルを読み取り/使用するよう構成されている場合は、そこで追加のエンジンセットアップ手順を実行できます。OpenSSL 構成ドキュメントについては、OpenSSL ドキュメントを参照してください：

[OpenSSL 1.0.2 ドキュメント](#) [OpenSSL 1.1.1 ドキュメント](#)

たとえば、アプリケーションは、デフォルトの OpenSSL 構成ファイル (openssl.cnf) または OPENSSL_CONF 環境変数によって設定された構成、およびデフォルトの [openssl_conf] セクションを呼び出して、読み取り、使用できます

```
OPENSSL_config(NULL);
```

OpenSSL コンフィギュレーションファイルを使用する代わりに、アプリケーションは希望の ENGINE_* API を使用して明示的に wolfEngine を初期化および登録できます。一例として、wolfEngine の初期化とすべてのアルゴリズムの登録は、以下を使用して行うことができます：

```
ENGINE* e = NULL;

e = ENGINE_by_id( "wolfengine" );
if ( e == NULL ) {
    printf( "Failed to find wolfEngine\n" );
    /* error */
}
ENGINE_set_default(e, ENGINE_METHOD_ALL);

/* normal application execution / behavior */

ENGINE_finish(e);
ENGINE_cleanup();
```

8.2 OpenSSL コンフィギュレーションファイルからの wolfEngine のロード

OpenSSL を使用するアプリケーションがコンフィギュレーションファイルを処理するように設定されている場合、wolfEngine は OpenSSL コンフィギュレーションファイルからロードできます。wolfEngine ライ

ブラリをコンフィギュレーションファイルに追加する方法の例を以下に示します。[wolfssl_section] は、必要に応じてエンジン制御コマンド (enable_debug) を設定するように変更できます。

```
openssl_conf = openssl_init

[openssl_init]
engines = engine_section

[engine_section]
wolfSSL = wolfssl_section

[wolfssl_section]
# If using OpenSSL <= 1.0.2, change engine_id to wolfengine
# (drop the "lib").
engine_id = libwolfengine
# dynamic_path = .libs/libwolfengine.so
init = 1
# Use wolfEngine as the default for all algorithms it provides.
default_algorithms = ALL
# Only enable when debugging application - produces large
# amounts of output.
# enable_debug = 1
```

8.3 wolfEngine 静的エン트리ポイント

wolfEngine がスタティック ライブラリとして使用される場合、アプリケーションは次のエン트리 ポイントを呼び出して wolfEngine をロードできます：

```
#include <wolfengine/we_wolfengine.h>
ENGINE_load_wolfengine();
```

9 wolfEngine の設計

wolfEngine は次のソース ファイルで構成され、すべて wolfEngine パッケージの “src” サブディレクトリの下にあります。

| ソースファイル | 詳細 |
|-------------------|--|
| we_wolfengine.c | ライブラリ エントリ ポイントが含まれます。OpenSSL エンジン フレームワークを使用してライブラリを動的にロードするために OpenSSL IMPLEMENT_DYNAMIC_BIND_FN を呼び出します。コンパイルしてスタティック ライブラリとして使用する場合のスタティック エントリ ポイントも含まれます |
| we_internal.c | エンジン アルゴリズム コールバックの登録を処理する wolfengine_bind() 関数が含まれています。他の wolfengine の内部機能も含まれています。 |
| we_logging.c | wolfEngine ロギング フレームワークと関数の実装 |
| we_openssl_bc.c | wolfEngine OpenSSL バイナリ互換抽象化レイヤー。複数の OpenSSL バージョンで wolfEngine をサポートするために使用されます。 |
| we_aes_block.c | wolfEngine AES-ECB および AES-CBC の実装 |
| we_aes_cbc_hmac.c | wolfEngine AES-CBC-HMAC 実装 |
| we_aes_ccm.c | wolfEngine AES-CCM 実装 |
| we_aes_ctr.c | wolfEngine AES-CTR 実装 |
| we_aes_gcm.c | wolfEngine AES-GCM 実装 |
| we_des3_cbc.c | wolfEngine 3DES-CBC の実装 |
| we_dh.c | wolfEngine DH の実装 |
| we_digest.c | wolfEngine メッセージ ダイジェストの実装 (SHA-1、SHA-2、SHA-3) |
| we_ecc.c | wolfEngine ECDSA および ECDH の実装 |
| we_mac.c | wolfEngine HMAC および CMAC の実装 |
| we_random.c | wolfEngine RAND 実装 |
| we_rsa.c | wolfEngine RSA 実装 |
| we_tls_prf.c | wolfEngine TLS 1.0 PRF 実装 |

一般的な wolfEngine アーキテクチャは次のとおりで、動的エントリ ポイントと静的エントリ ポイントの両方を示しています:

9.1 wolfEngine エントリーポイント

wolfEngine ライブラリへの主なエントリ ポイントは、**wolfengine_bind()** または **ENGINE_load_wolfengine()** のいずれかです。wolfEngine が動的にロードされている場合、wolfengine_bind() は OpenSSL によって自動的に呼び出されます。ENGINE_load_wolfengine() は、wolfEngine が動的ではなく静的に構築および使用されている場合に、アプリケーションが呼び出す必要があるエントリ ポイントです。

9.2 wolfEngine アルゴリズム コールバック登録

wolfEngine は、wolfCrypt FIPS でサポートされているすべてのコンポーネントに対して、アルゴリズム構造体とコールバックを OpenSSL エンジン フレームワークに登録します。この登録は、we_internal.c の wolfengine_bind() 内で行われます。wolfengine_bind() は、wolfEngine エンジンを表す ENGINE 構造体ポインタを受け取ります。次に、個々のアルゴリズム/コンポーネントのコールバックまたは構造体が、<openssl/engine.h> の適切な API を使用してその ENGINE 構造体に登録されます。

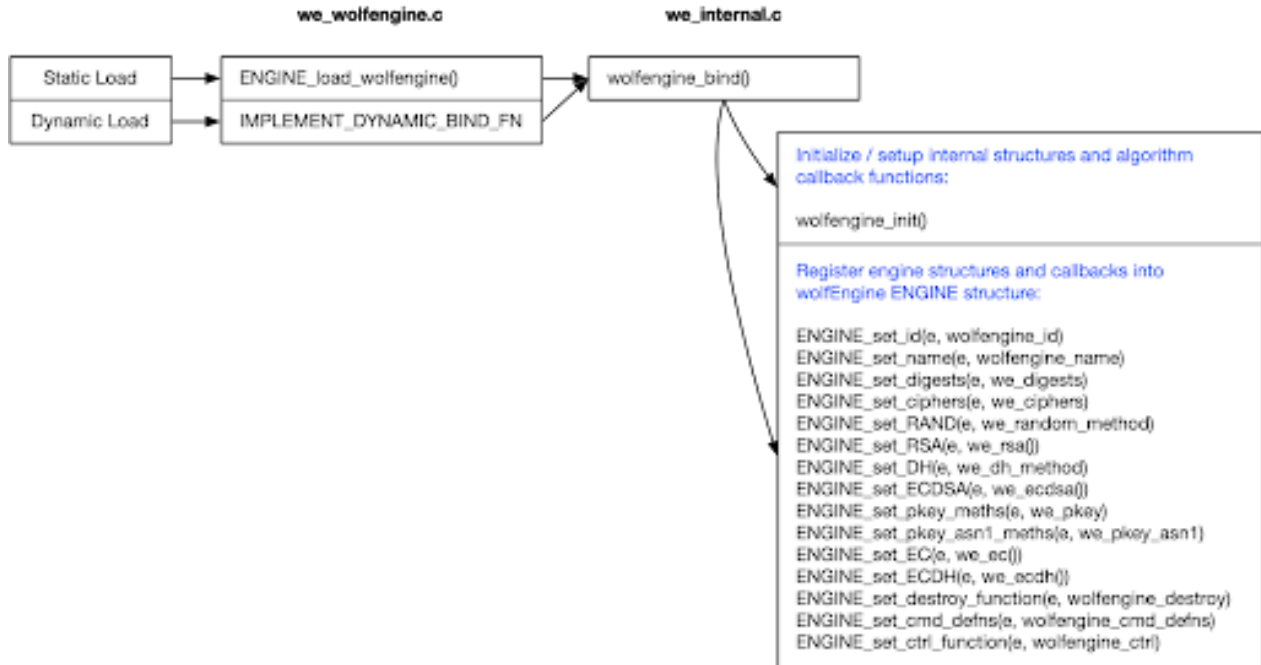


Figure 2: wolfEngine Architecture

これらの API 呼び出しには、次のものが含まれます：

```
ENGINE_set_id(e, wolfengine_id)
ENGINE_set_name(e, wolfengine_name)
ENGINE_set_digests(e, we_digests)
ENGINE_set_ciphers(e, we_ciphers)
ENGINE_set_RAND(e, we_random_method)
ENGINE_set_RSA(e, we_rsa())
ENGINE_set_DH(e, we_dh_method)
ENGINE_set_ECDSA(e, we_ecdsa())
ENGINE_set_pkey_meths(e, we_pkey)
ENGINE_set_pkey_asn1_meths(e, we_pkey_asn1)
ENGINE_set_EC(e, we_ec())
ENGINE_set_ECDH(e, we_ecdh())
ENGINE_set_destroy_function(e, wolfengine_destroy)
ENGINE_set_cmd_defns(e, wolfengine_cmd_defns)
ENGINE_set_ctrl_function(e, wolfengine_ctrl)
```

上記の呼び出しで使用される各アルゴリズム/コンポーネントのコールバック関数または構造体 (例: `we_digests`、`we_ciphers` など) は、`we_internal.c` またはそれぞれのアルゴリズム ソース ファイルに実装されています。

10 オープンソース統合に関する注意事項

wolfEngine は、一般的な OpenSSL エンジン フレームワークとアーキテクチャに準拠しています。そのため、OpenSSL を使用するアプリケーションから、OpenSSL 構成ファイルを介して、または ENGINE API 呼び出しを介して他のエンジン実装と同様に wolfEngine をプログラムで利用することができます。

wolfSSL は、いくつかのオープンソースプロジェクトで wolfEngine をテストしました。この章には、wolfEngine 統合に関する注意事項とヒントが含まれています。この章は、wolfEngine によるすべてのオープンソースプロジェクトのサポートを網羅しているわけではなく、wolfSSL またはコミュニティが追加のオープンソースプロジェクトで wolfEngine をテストおよび使用することを報告するにつれて拡張されます。

10.1 cURL

cURL は、OpenSSL 構成ファイルを利用するように既にセットアップされています。wolfEngine を利用するには:

1. wolfEngine エンジン情報を OpenSSL 設定ファイルに追加します
2. 必要に応じて、OPENSSL_CONF 環境変数が OpenSSL 構成ファイルを指すように設定します:

```
$ export OPENSSL_CONF=/path/to/openssl.cnf
```

3. OPENSSL_ENGINES 環境変数を wolfEngine 共有ライブラリ ファイルの場所を指すように設定します:

```
$ export OPENSSL_ENGINES=/path/to/wolfengine/library/dir
```

10.2 stunnel

stunnel は wolfEngine でテストされています。ノートは近日公開予定。

10.3 OpenSSH

OpenSSH は、`--with-ssl-engine` 構成オプションを使用して、OpenSSL エンジン サポートでコンパイルする必要があります。必要に応じて、`--with-ssl-dir=DIR` を使用して、使用されている OpenSSL ライブラリのインストール場所を指定することもできます:

```
$ cd openssl
$ ./configure --prefix=/install/path --with-ssl-dir=/path/to/openssl/install
--with-ssl-engine
$ make
$ sudo make install
```

OpenSSH には、wolfEngine を活用するための OpenSSL 構成ファイルのセットアップも必要です。必要に応じて、OPENSSL_CONF 環境変数を構成ファイルを指すように設定できます。OPENSSL_ENGINES 環境変数も、wolfEngine 共有ライブラリの場所に設定する必要があります:

```
$ export OPENSSL_CONF=/path/to/openssl.cnf
$ export OPENSSL_ENGINES=/path/to/wolfengine/library/dir
```

11 サポートと OpenSSL バージョン追加

wolfEngine のサポートについては、support@wolfssl.com 宛てお問い合わせください。サポートが必要となる OpenSSL バージョンの追加を希望される場合はfacts@wolfssl.com 宛てにご連絡ください。