wolfSSL FIPS Ready User Guide

2025-12-12

Contents

1 wolfSSL FIPS Ready

FIPS 認証の取得が求められるプロジェクトには、wolfSSL FIPS Ready 版が役立ちます。wolfSSL ソースツリーに含まれる FIPS 対応の暗号レイヤーを用いて、FIPS 認証を取得する手間を低減できます。FIPS Ready 版は FIPS 版と同一の構成ですが、認証は取得していません。必要なタイミングで FIPS 認証プロセスを実行する準備が整っています。FIPS Ready 版には FIPS コードが組み込まれており、FIPS 140-3 に示されるデフォルト・エントリーポイントの実用例や Conditional Algorithm Self-tests(CAST) を含んでいます。必要なタイミングで実行環境をテストし、wolfSSL の既存の FIPS 140-3 認証に追加、または新たな FIPS140-3 認証を申請することですべての手続きが完了します。

FIPS Ready 版はオープンソースと商用ライセンスのデュアルライセンスで提供しており、GPLv3 ライセンス下で動作検証を行っていただけます。その後商用環境でご利用いただくには、商用ライセンス契約が必要です。

FIPS は複雑なトピックです。このドキュメントをご覧いただいた後、ご質問がありましたら info@wolfssl.jp までお問い合わせください。

1.1 非 FIPS 版との違い

wolfCrypt FIPS API は、FIPS 環境内に存在するすべてのアルゴリズム関数のラッパーを提供します。FIPS API をご利用にならず、非 FIPS 版にも存在する API を用いてコードを記述いただくことも可能です。いずれの場合でも、コンパイル時に自動的に wolfCrypt FIPS API に置換します。FIPS API は、実際の関数を呼び出す前に内部ステータスのセルフテストを行います。そして各アルゴリズムの初回実行時に CAST が実行します。このようなタイミングでの実行を避けたい場合は、起動時に予め各テストを実行することも可能です。

wolfCrypt FIPS 140-3 Ready 版のコードには、メモリ内の実行可能ファイルの整合性を自動的にチェックする必須の電源投入時自己テスト (POST) が含まれています。これにはアルゴリズムの Known Answere Tests (KAT) が含まれており、140-2 以降変更しておりません。POST で使用しないものは、使用が条件付きになりました。実行可能ファイルは、FIPS 環境内のコードがメモリ内で隣接するように設計しています。FIPS コードを使用するアプリケーションが起動するか、共有ライブラリがロードされると、ライブラリのデフォルトエントリーポイントを呼び出し、POST を自動的に実行します。これには、コア内のメモリチェックと POST で使用するアルゴリズム (HMAC-SHA256) のための KAT の 2 つが含まれます。

まず HMAC-SHA256 の POST を実行し、次いでコア内メモリテストを実行します。メモリ内のコードは HMAC-SHA256 でハッシュを取り、一致した場合のみテストを続行します。一致しなければ FIPS モジュールはエラー状態となり、整合性が取れるまですべての呼び出しは成功しません。

FIPS 環境内の他のアルゴリズムは、初回使用時または開発者が指定したタイミングで既定のデータを用いてテストします。出力は事前に計算された回答と比較します。テスト値はすべて FIPS 環境内に存在し、呼び出された際にチェックします。署名と検証などのいくつかのテストではランダム性を含むため、既知のデータを署名し既知のデータで検証します。鍵の生成も同様にテストします。

2 wolfSSL FIPS Ready 版のビルド手順

通常版と大きな差はありませんが、いくつかの追加手順が必要です。

ここでは、Linux または macOS 環境で wolfSSL FIPS Ready GPL v3 版を利用し、システムに共有ライブラリとしてインストールする手順を示します。

2.1 ソースコードの解凍

```
$ tar xzvf wolfssl-5.6.4-gplv3-fips-ready.tar.gz
$ cd wolfssl-5.6.0-gplv3-fips-ready
```

ソースコードを解凍し、生成されたディレクトリに移動します。商用リリースを受け取った場合は、gplv3 を commercial に、.tar.gz を.7z に、tar xzvf を 7z x -p に置き換えてください。

2.2 ビルドを構成

```
$ ./configure --enable-fips=ready
```

このコマンドは、FIPS Ready 版の wolfSSL を構築するように Makefile を構成します。

2.3 ライブラリのビルド

```
$ make
```

これにより、すべてのソースがコンパイルされライブラリがリンクされます。また、サンプルツールとテストツールも構築します。

2.4 コアメモリのハッシュを更新

```
$ ./fips-hash.sh

$ make # Re-build once the hash has been updated
```

このステップでコア内メモリ テストのハッシュを計算し、更新する必要があります。wolfCrypt テストは、fips-hash.sh スクリプトによって呼び出されたときに失敗するはずです。次のメッセージが出力されます。(ハッシュ値は一意であることにご注意ください)

```
in my Fips callback, ok = 0, err = -203

message = In Core Integrity check FIPS error

hash = 8D29242F610EAEA179605BB1A99974EBC72B0ECDB26B483B226A729F36FC82A2

In core integrity hash check failure, copy above hash

into verifyCore[] in fips_test.c and rebuild
```

ビルド時に他のオプションを追加するとハッシュ値が変化する可能性があるため、この手順を繰り返す必要があります。またアプリケーションに変更を加えると、アプリケーションを再コンパイルしたときにメモリ内の FIPS 境界が移動する可能性があります。アプリケーションのみが更新された場合のハッシュの変化は、モジュールが影響を受けていることを示すものではなく、メモリ内の所定の位置にシフトされたことを

示します。これは、静的ライブラリとして使用する場合に発生します。共有ライブラリではこの問題は発生しません。

4.1 提供している fips-hash.sh スクリプトを使用せずに上記を実行する場合は、ファイル wolfcrypt/src/-fips_test.c を編集してハッシュを手動で更新するか、次のような設定を使用することができます。

```
$ ./configure --enable-fips=ready CFLAGS=”-DWOLFCRYPT_FIPS_CORE_HASH_VALUE=8
    D29242F610EAEA179605BB1A99974EBC72B0ECDB26B483B226A729F36FC82A2 ”
```

4.2 再び make を実行します。

2.5 ビルドの検証

```
$ make check
```

Makefile 内のチェック ターゲットは、wolfSSL および wolfCrypt で提供されるすべてのテストツールとスクリプトです。すべてが正常であれば、次の出力が表示されるはずです。

```
PASS: scripts/resume.test

PASS: scripts/external.test

PASS: scripts/google.test

PASS: testsuite/testsuite.test

PASS: scripts/openssl.test

PASS: tests/unit.test

============================================================================

Testsuite summary for wolfssl 4.0.0

============================================================================

# TOTAL: 6

# PASS:  6

# SKIP:  0

# XFAIL: 0

# FAIL:  0

# XPASS: 0

# ERROR: 0

============================================================================
```

2.6 ライブラリのインストール

```
$ make install
```

Makefile 内の install ターゲットにより、すべてのヘッダーとライブラリがシステムにインストールされます。デフォルトでは /usr/local 配下にインストールします。

これで、アプリケーションのビルド時に wolfSSL FIPS Ready 版をご利用いただけます。

3 FIPS 140-2 Ready 版からの変更点

1. FIPS モードで実行する場合は、アプリケーションレベルで wc_SetSeed_Cb を呼び出す必要があります。

a. +#ifdef WC_RNG_SEED_CB

b. • wc_SetSeed_Cb(wc_GenerateSeed);

c. +#endif

2. 鍵アクセス管理

a. wolfSSL(SSL/TLS)API を呼び出す場合には、この項による影響はありません。しかし Crypt API(wc_XXX) を呼び出す場合には、次のステップをご参照ください。

b. 秘密鍵のロードまたは使用を伴う wolfCrypt (wc_XXX) API を直接呼び出す場合は、アプリケーションレベルで鍵へのアクセスを管理する必要があります。具体的には、秘密鍵のロードや使用前に、アプリケーションで PRIVATE_KEY_UNLOCK() を呼び出してロックを解除しなければなりません。使用後には PRIVATE_KEY_LOCK() を呼び出して再度ロックする必要があります。

i. PRIVATE_KEY_UNLOCK() と PRIVATE_KEY_LOCK() は、アプリケーションの起動時及びシャットダウン時にそれぞれ1回だけ呼び出すことも可能です。

ii. 秘密鍵のロードや使用の前後で都度これらを呼び出すことで、より厳密なアクセス管理を実現できます。

** アプリケーション終了前に、必ずロックする必要があります。 - これはドキュメント要件であり、実行時にエラーや終了防止によって強制されるものではありません。 終了する前にキーを再ロックしないと、アプリケーションは「FIPS に準拠していない」ことになります。

c. アプリケーションを wolfSSL FIPS 版と非 FIPS 版の両方にリンクできるようにするには、次のように実装します。

```
#if !defined(PRIVATE_KEY_LOCK) && !defined(PRIVATE_KEY_UNLOCK)
    #define PRIVATE_KEY_LOCK() do {} while (0)
    #define PRIVATE_KEY_UNLOCK() do {} while (0)
#endif
```

4 wolfCrypt FIPS Ready API Documentation

以下は、wolfCrypt FIPS Ready API の概要です。詳細については、wolfCrypt API ドキュメントをご覧ください。

4.1 Map of API to FIPS 140-3 module services

4.1.1 Digital Signature Service

API Call	Description
InitRsaKey_fips	オプションのヒープ ヒント p で使用する RSA 鍵オブジェクトを初期化します。成功すると 0、エラーの場合は負値を返します。
FreeRsaKey_fips	RSA 鍵リソースを解放します。成功すると 0、エラーの場合は負値を返します。
RsaSSL_Sign_fips	サイズ inLen の入力 in に対して RSA 鍵署名操作を実行し、rng を使用してサイズ outLen の out に出力します。成功すると 0、エラーの場合は負値を返します。
RsaSSL_VerifyInline_fips	サイズ inLen の入力 in に一時メモリを割り当てずに RSA 鍵検証を実行し、出力 out に書き込みます。成功すると 0、エラーの場合は負値を返します。
RsaSSL_Verify_fips	サイズ inLen の入力 in に対して RSA 鍵検証を実行し、サイズ outLen の出力 out に書き込みます。成功すると 0、エラーの場合は負値を返します。
SS_Sign_fips	サイズ inLen の入力 in に対して PSS パディングを使用して RSA 鍵署名操作を実行し、rng を使用してサイズ outLen の out に出力します。マスク生成関数 mgf を使用したハッシュアルゴリズム hash を使用します。成功すると 0、エラーの場合は負値を返します。
RsaPSS_SignEx_fips	サイズ inLen の入力 in に対して PSS パディングを使用して RSA 鍵署名操作を実行し、rng を使用してサイズ outLen の out に出力します。これは、マスク生成関数 mgf とソルト長の SaltLen を備えたハッシュアルゴリズム hash を使用します。成功すると 0、エラーの場合は負値を返します。
RsaPSS_VerifyInline_fips	サイズ inLen の入力 in に一時メモリを割り当てずに RSA 鍵検証を実行し、出力 out に書き込みます。マスク生成関数 mgf を使用したハッシュアルゴリズム hash を使用します。成功すると 0、エラーの場合は負値を返します。

API Call	Description
RsaPSS_VerifyInlineEx_fips	サイズ inLen の入力 in に対して RSA 鍵検証を実行し、サイズ outLen の出力 out に書き込みます。ハッシュアルゴリズム hash とマスク生成関数 mgf およびソルト長の SaltLen を使用します。成功すると 0、エラーの場合は負値を返します。
RsaPSS_Verify_fips	サイズ inLen の入力 in に対して RSA 鍵検証を実行し、サイズ outLen の出力 out に書き込みます。ハッシュアルゴリズム hash とマスク生成関数 mgf を使用します。成功すると 0、エラーの場合は負値を返します。
RsaPSS_VerifyEx_fips	サイズ inLen の入力 in に対して RSA 鍵検証を実行し、サイズ outLen の出力 out に書き込みます。ハッシュアルゴリズム hash とマスク生成関数 mgf およびソルト長の SaltLen を使用します。成功すると 0、エラーの場合は負値を返します。
RsaPSS_CheckPadding_fips	ハッシュ hashType を使用して、サイズ sigSz の署名 sig をサイズ inSz の入力 in での RSA 鍵検証後のパディングをチェックします。成功すると 0、エラーの場合は負値を返します。
RsaPSS_CheckPaddingEx_fips	ハッシュ hashType と長さ saltLen の salt を使用して、サイズ sigSz の署名 sig をサイズ inSz の入力 in での RSA 鍵検証後のパディングをチェックします。成功すると 0、エラーの場合は負値を返します。
RsaEncryptSize_fips	RSA 鍵の出力サイズを取得します。成功した場合は鍵の出力サイズ (正値)、エラーの場合は負値を返します。
wc_RsaPrivateKeyDecode	サイズ inSz のインデックス inOutIdx で始まるバッファ入力 in から RSA 秘密鍵をデコードします。成功すると 0、エラーの場合は負値を返します。
wc_RsaPublicKeyDecode	サイズ inSz のインデックス inOutIdx で始まるバッファ入力 in から RSA 公開鍵をデコードします。成功すると 0、エラーの場合は負値を返します。
ecc_init_fips	ECC 鍵オブジェクトを使用するために初期化します。成功すると 0、エラーの場合は負値を返します。
ecc_free_fips	ECC 鍵オブジェクトのリソースを解放します。成功すると 0、エラーの場合は負値を返します。
ecc_import_x963_fips	サイズ inLen の入力 in から ANSI X9.63 形式の ECC 公開鍵をインポートします。成功すると 0、エラーの場合は負値を返します。
ecc_sign_hash_fips	長さ inlen の入力 in に対して ECC 鍵署名操作を実行し、rng を使用して長さ outlen の out に出力します。成功すると 0、エラーの場合は負値を返します。

API Call	Description
ecc_verify_hash_fips	長さ hashlen のハッシュを使用して、サイズ siglen の sig の ECC 鍵検証を実行します。署名検証結果は res で返します。成功すると 0、エラーの場合は負値を返します。

4.1.2 鍵生成

API Call	Description
MakeRsaKey_fips	乱数発生器 rng にモジュラス長 size と指数 e を使用して RSA 鍵を生成します。成功すると 0、エラーの場合は負値を返します。
CheckProbablePrime_fips	長さ nlen のポテンシャル係数については、サイズ pRawSz の候補数 pRaw とサイズ qRawSz の qRaw をチェックして、それらがおそらく素数であるかどうかを確認します。両方とも、サイズ eRawSz が 1 の指数 eRaw を持つ GCD を持つ必要があります。主な候補は Miller-Rabin でチェックします。結果は isPrime に書き込みます。成功すると 0、エラーの場合は負値を返します。
RsaExportKey_fips	RSA 鍵を、eSz の e、nSz の n、dSz の d、pSz の p、qSz の q のコンポーネントとして出力します。サイズはバッファのサイズである必要があり、数値の実際の長さに更新されます。成功すると 0、エラーの場合は負値を返します。
ecc_make_key_fips	rng を使用して、サイズ keysize の鍵 key に対して ECC 鍵生成操作を実行します。成功すると 0、エラーの場合は負値を返します。
ecc_make_key_ex_fips	rng を使用して、楕円曲線 curve_id を持つサイズ keysize の鍵 key に対して ECC 鍵生成操作を実行します。成功すると 0、エラーの場合は負値を返します。
ecc_export_x963_fips	ECC 公開鍵を ANSI X9.63 形式でサイズ outLen の out に出力します。成功すると 0、エラーの場合は負値を返します。
InitDhKey_fips	DH 鍵オブジェクトを使用できるように初期化します。成功すると 0、エラーの場合は負値を返します。
FreeDhKey_fips	DH 鍵リソースを解放します。成功すると 0、エラーの場合は負値を返します。

API Call	Description
DhSetKeyEx_fips	サイズ pSz の符号なしバイナリ入力 p、サイズ qSz の q、およびサイズ gSz の g から DH 鍵のグループパラメーターを設定します。成功すると 0、エラーの場合は負値を返します。
DhGenerateKeyPair_fips	DH 鍵の rng を使用して、サイズ pubSz の公開鍵 pub、サイズ privSz の秘密鍵 priv を生成します。成功すると 0、エラーの場合は負値を返します。
CheckRsaKey_fips	RSA 鍵に対してペアごとの鍵検証を実行します。成功すると 0、エラーの場合は負値を返します。
ecc_check_key_fips	ECC 鍵に対してペアごとの鍵検証を実行します。成功すると 0、エラーの場合は負値を返します。
DhCheckPubKeyEx_fips	鍵内のドメインパラメーターまたはサイズ primeSz の素数 prime を用いて、サイズ pubSz の pub 値に対して数学的鍵検証を実行します。
DhCheckPrivKeyEx_fips	鍵内のドメインパラメーターまたはサイズ primeSz の素数 prime を用いて、サイズ privSz の priv 値に対して数学的鍵検証を実行します。
DhCheckKeyPair_fips	ドメインパラメーターを使用して、サイズ pubSz の pub 値とサイズ privSz の priv 値の間でペアごとの鍵検証を実行します。成功すると 0、エラーの場合は負値を返します。
HKDF_fips	ハッシュ type とサイズ inKeySz の inKey、長さ saltSz のソルト salt と infoSz の情報 info を使用して、HMAC ベースの鍵導出関数を実行します。鍵はサイズ outSz の出力 out に書き込みます。成功すると 0、エラーの場合は負値を返します。

4.1.3 鍵合意

API Call	Description
ecc_shared_secret_fips	privKey とペアの pubKey を使用して ECDHE 鍵合意操作を実行し、結果を長さ sharedSz の sharedSecret に保存します。成功すると 0、エラーの場合は負値を返します。
DhAgree_fips	サイズ privSz の DH 秘密鍵 priv とサイズ pubSz のペアの公開鍵 otherPub を使用して、サイズ acceptSz の合意 agree を作成します。成功すると 0、エラーの場合は負値を返します。

4.1.4 鍵転送

API Call	Description
RsaPublicEncrypt_fips	サイズ inLen の入力 in に対して RSA 公開鍵を用いて暗号化し、rng を使用してサイズ outLen の出力 out に書き込みます。成功すると 0、エラーの場合は負値を返します。
RsaPublicEncryptEx_fips	サイズ inLen の入力 in に対して RSA 公開鍵を用いて暗号化し、rng を使用してサイズ outLen の出力 out に書き込みます。ハッシュ type のパディングを使用します。PSS パディングを使用する場合は、サイズ labelSz のラベル label を持つハッシュと mgf を使用します。成功すると 0、エラーの場合は負値を返します。
RsaPrivateDecryptInline_fips	サイズ inLen の入力 in に一時メモリを割り当てずに RSA 秘密鍵を用いて復号し、出力 out に書き込みます。成功すると 0、エラーの場合は負値を返します。
RsaPrivateDecryptInlineEx_fips	サイズ inLen の入力 in に一時メモリを割り当てずに RSA 秘密鍵を用いて復号し、出力 out に書き込みます。type のパディングを使用します。PSS パディングを使用する場合は、サイズ labelSz のラベル label を持つハッシュと mgf を使用します。成功すると 0、エラーの場合は負値を返します。
RsaPrivateDecrypt_fips	サイズ inLen の入力 in に対して RSA 秘密鍵を用いて復号し、サイズ outLen の出力 out に書き込みます。成功すると 0、エラーの場合は負値を返します。
RsaPrivateDecryptEx_fips	サイズ inLen の入力 in に対して RSA 秘密鍵を用いて復号し、サイズ outLen の出力 out に書き込みます。type のパディングを使用します。PSS パディングを使用する場合は、サイズ labelSz のラベル label を持つハッシュと mgf を使用します。成功すると 0、エラーの場合は負値を返します。

4.1.5 鍵付きハッシュ

API Call	Description
HmacSetKey_fips	ハッシュ type を使用して、サイズ keySz の鍵 key で hmac オブジェクトを初期化します。成功すると 0、エラーの場合は負値を返します。
HmacUpdate_fips	サイズ len の入力 data に対して hmac Update を実行します。成功すると 0、エラーの場合は負値を返します。
HmacFinal_fips	hmac Final を実行し、ダイジェストをハッシュに出力します。成功すると 0、エラーの場合は負値を返します。

API Call	Description
Gmac_fips	サイズ authInSz の入力 authIn に対して GMAC を実行し、サイズ authTagSz の authTag を出力します。長さ keySz の鍵 key を使用し、乱数発生器 rng を使用して iv に格納する長さ ivSz の IV をランダムに生成します。成功すると 0、エラーの場合は負値を返します。
GmacVerify_fips	長さ keySz の鍵 key と長さ ivSz の iv を使用して、サイズ authInSz の入力 authIn 上の長さ authTagSz の GMAC authTag を検証します。成功すると 0、エラーの場合は負値を返します。
InitCmac_fips	ハッシュ type を使用して、サイズ keySz の鍵 key で cmac オブジェクトを初期化します。成功すると 0、エラーの場合は負値を返します。
CmacUpdate_fips	サイズ inSz の入力 in に対して cmac Update を実行します。成功すると 0、エラーの場合は負値を返します。
CmacFinal_fips	cmac Final を実行し、サイズ outSz の出力 out にダイジェストを出力します。これは実際の出力サイズで更新されます。成功すると 0、エラーの場合は負値を返します。

4.1.6 メッセージダイジェスト

API Call	Description
InitSha_fips	sha オブジェクトを使用できるように初期化します。成功すると 0、エラーの場合は負値を返します。
ShaUpdate_fips	サイズ len の入力 data に対して SHA-Update を実行します。成功すると 0、エラーの場合は負値を返します。
ShaFinal_fips	sha Final を実行し、ダイジェストをハッシュに出力します。成功すると 0、エラーの場合は負値を返します。
InitSha224_fips	sha224 オブジェクトを使用できるように初期化します。成功すると 0、エラーの場合は負値を返します。
Sha224Update_fips	サイズ len の入力 data に対して sha224 Update を実行します。成功すると 0、エラーの場合は負値を返します。
Sha224Final_fips	sha224 Final を実行し、ダイジェストをハッシュに出力します。成功すると 0、エラーの場合は負値を返します。
InitSha256_fips	sha256 オブジェクトを使用できるように初期化します。成功すると 0、エラーの場合は負値を返します。

API Call	Description
Sha256Update_fips	サイズ len の入力 data に対して sha256 Update を実行します。成功すると 0、エラーの場合は負値を返します。
Sha256Final_fips	sha256 Final を実行し、ダイジェストをハッシュに出力します。成功すると 0、エラーの場合は負値を返します。
InitSha384_fips	sha384 オブジェクトを使用できるように初期化します。成功すると 0、エラーの場合は負値を返します。
Sha384Update_fips	サイズ len の入力 data に対して sha384 Update を実行します。成功すると 0、エラーの場合は負値を返します。
Sha384Final_fips	sha384 Final を実行し、ダイジェストをハッシュに出力します。成功すると 0、エラーの場合は負値を返します。
InitSha512_fips	sha512 オブジェクトを使用できるように初期化します。成功すると 0、エラーの場合は負値を返します。
Sha512Update_fips	サイズ len の入力 data に対して sha512 Update を実行します。成功すると 0、エラーの場合は負値を返します。
Sha512Final_fips	sha512 Final を実行し、ダイジェストをハッシュに出力します。成功すると 0、エラーの場合は負値を返します。
InitSha3_224_fips	sha3 (224 ビット) オブジェクトを使用できるように初期化します。成功すると 0、エラーの場合は負値を返します。
Sha3_224_Update_fips	サイズ len の入力 data に対して sha3 (224 ビット) 更新を実行します。成功すると 0、エラーの場合は負値を返します。
Sha3_224_Final_fips	sha3 (224 ビット) Final を実行し、ダイジェストをハッシュに出力します。成功すると 0、エラーの場合は負値を返します。
InitSha3_256_fips	sha3 (256 ビット) オブジェクトを使用できるように初期化します。成功すると 0、エラーの場合は負値を返します。
Sha3_256_Update_fips	サイズ len の入力 data に対して sha3 (256 ビット) 更新を実行します。成功すると 0、エラーの場合は負値を返します。

API Call	Description
Sha3_256_Final_fips	sha3 (256 ビット) Final を実行し、ダイジェストをハッシュに出力します。成功すると 0、エラーの場合は負値を返します。
InitSha3_384_fips	sha3 (384 ビット) オブジェクトを使用できるように初期化します。成功すると 0、エラーの場合は負値を返します。
Sha3_384_Update_fips	サイズ len の入力 data に対して sha3 (384 ビット) 更新を実行します。成功すると 0、エラーの場合は負値を返します。
Sha3_384_Final_fips	sha3 (384 ビット) Final を実行し、ダイジェストをハッシュに出力します。成功すると 0、エラーの場合は負値を返します。
InitSha3_512_fips	sha3 (512 ビット) オブジェクトを使用できるように初期化します。成功すると 0、エラーの場合は負値を返します。
Sha3_512_Update_fips	サイズ len の入力 data に対して sha3 (512 ビット) 更新を実行します。成功すると 0、エラーの場合は負値を返します。
Sha3_512_Final_fips	sha3 (512 ビット) Final を実行し、ダイジェストをハッシュに出力します。成功すると 0、エラーの場合は負値を返します。

4.1.7 乱数生成

API Call	Description
InitRng_fips	RNG オブジェクトを使用できるように初期化します。成功すると 0、エラーの場合は負値を返します。
InitRngNonce_fips	サイズ nonceSz の nonce で使用するために RNG オブジェクトを初期化します。成功すると 0、エラーの場合は負値を返します。
FreeRng_fips	RNG リソースを解放し、状態をゼロにします。成功すると 0、エラーの場合は負値を返します。ゼロライズサービスの一部でもあります。
RNG_GenerateBlock_fips	ユーザー用の RNG 出力のブロックを bufSz バイトの buf に取得します。成功すると 0、エラーの場合は負値を返します。

API Call	Description
RNG_HealthTest_fips	reseed が 0 の場合、entropyASz バイトのシードバッファ entropyA を使用して、outputSz バイトの予想される出力に対して RNG の一時インスタンスの出力をテストします。このとき、entropyB と entropyBSz は無視されます。reseed が 1 の場合、テストではサイズ entropyBSz のシード バッファ entropyB を使用して RNG の一時インスタンスも再シードし、サイズが outSz バイトの予想される出力に対して RNG をテストします。成功すると 0、エラーの場合は負値を返します。

4.1.8 ステータス出力

API Call	Description
wolfCrypt_GetStatus_fips	モジュールの現在のステータスを返します。返り値 0 は、モジュールがエラーのない状態にあることを意味します。その他の返り値は、モジュールが特定のエラー状態にあることを示します。
wolfCrypt_GetVersion_fips	wolfCrypt ライブラリバージョンの null 終端文字列へのポインタを返します。
wolfCrypt_GetCoreHash_fips	コアハッシュの null 終端文字列へのポインタを 16 進数で返します。

4.1.9 共通鍵暗号

API Call	Description
AesSetKey_fips	長さ keylen の userKey を使用して AES オブジェクトを初期化します。dir は方向 (暗号化/復号) を示し、iv はオプションです。成功すると 0、エラーの場合は負値を返します。
AesSetIV_fips	ユーザー iv を使用して AES オブジェクトを初期化します。成功すると 0、エラーの場合は負値を返します。
AesCbcEncrypt_fips	入力 in に対して AES CBC 暗号化を実行し、サイズ sz の out に出力します。成功すると 0、エラーの場合は負値を返します。
AesCbcDecrypt_fips	入力 in に対して AES CBC 復号を実行し、サイズ sz の out に出力します。成功すると 0、エラーの場合は負値を返します。

API Call	Description
AesEcbEncrypt_fips	入力 in に対して AES ECB 暗号化を実行し、サイズ sz の out に出力します。成功すると 0、エラーの場合は負値を返します。
AesEcbDecrypt_fips	入力 in に対して AES ECB 復号を実行し、サイズ sz の out に出力します。成功すると 0、エラーの場合は負値を返します。
AesCtrEncrypt_fips	入力 in に対して AES CTR 暗号化を実行し、サイズ sz の out に出力します。成功すると 0、エラーの場合は負値を返します。この API は CTR 復号も実行します。
AesGcmSetKey_fips	長さ len の鍵を使用して AES オブジェクトを初期化します。成功すると 0、エラーの場合は負値を返します。
AesGcmSetExtIV_fips	外部で生成された長さ ivSz の iv を使用して AES オブジェクトを初期化します。成功すると 0、エラーの場合は負値を返します。
AesGcmSetIV_fips	最初の ivFixedSz バイトとして ivFixed を使用し、残りは rng からのランダム バイトであることを使用して、長さ ivSz の内部生成された IV で AES オブジェクトを初期化します。成功すると 0、エラーの場合は負値を返します。
AesGcmEncrypt_fips	入力 in からサイズ outSz の 出力 out に対して AES GCM 暗号化を実行します。現在の IV は、長さ ivOutSz のバッファ ivOut に格納します。認証タグは、サイズ authTagSz のバッファ authTag に格納します。authIn からの authInSz バイトが計算されて認証タグに組み込まれます。成功すると 0、エラーの場合は負値を返します。
AesGcmDecrypt_fips	サイズ ivSz の iv を使用して、入力 in に対して AES GCM 復号を実行し、サイズ sz の out に出力します。サイズ authTagSz の authTag は、authInSz バイトの入力 authIn を使用してチェックします。成功すると 0、エラーの場合は負値を返します。
AesCcmSetKey_fips	長さ keySz の鍵 key を使用して AES オブジェクトを初期化します。成功すると 0、エラーの場合は負値を返します。
AesCcmSetNonce_fips	外部で生成された長さ nonceSz の nonce を使用して AES オブジェクトを初期化します。成功すると 0、エラーの場合は負値を返します。

API Call	Description
AesCcmEncrypt_fips	入力 in からサイズ outSz の 出力 out に対して AES CCM 暗号化を実行します。現在の IV は、長さ nonceSz のバッファ nonce に格納します。認証タグは、サイズ authTagSz のバッファ authTag に格納します。authIn からの authInSz バイトが計算されて認証タグに組み込まれます。成功すると 0、エラーの場合は負値を返します。
AesCcmDecrypt_fips	サイズ nonceSz の nonce を使用して、サイズ inSz の入力 in に対して AES CCM 復号を実行し、out に出力します。サイズ authTagSz の authTag は、authIn の入力バイトと authInSz バイトを使用してチェックします。成功すると 0、エラーの場合は負値を返します。
Des3_SetKey_fips	des3 オブジェクトを鍵 key で初期化します。dir は方向 (暗号化/復号) を示し、iv はオプションです。成功すると 0、エラーの場合は負値を返します。
Des3_SetIV_fips	des3 オブジェクトをユーザー iv で初期化します。成功すると 0、エラーの場合は負値を返します。
Des3_CbcEncrypt_fips	サイズ sz の入力 in から出力 out へ DES3 CBC 暗号化を実行します。成功すると 0、エラーの場合は負値を返します。
Des3_CbcDecrypt_fips	サイズ sz の入力 in から出力 out へ DES3 CBC 復号を実行します。成功すると 0、エラーの場合は負値を返します。

4.1.10 ゼロライズ

API Call	Description
FreeRng_fips	RNG CSP を破棄します。他のすべてのサービスは、メモリ バインドされた CSP を自動的に上書きします。成功すると 0、エラーの場合は負値を返します。 スタックのクリーンアップはアプリケーションの義務です。汎用コンピュータを再起動すると、RAM 内のすべての CSP がクリアされます。 API Calls for Allowed Security Functions