wolfSSL FIPS FAQ

2026-01-19

Contents

1 Introduction

This page lists some of the most common issues and questions that are recieved by our wolfSSL security experts, along with their responses. This FAQ is useful for solving general questions that pertain to building/implementing wolfSSL FIPS. If this page does not provide an answer to your question, please feel free to check the wolfSSL Manual, or contact us at support@wolfssl.com.

Last Updated: 8 Dec 2025

1.1 Questions

2 Frequently Asked Questions

2.1 Why did I receive wolfSSL_X.X.X_commercial-fips-OE-v2.7z when we validated with Y.Y.Y?

Example: I received wolfssl-4.8.1-commercial-fips-ARMv8-A-v2 but the validation was for version 4.5.4, why did I receive a 4.8.1 release?

A: The version validated (IE 4.5.4 from the exmaple) applies to the wolfCrypt module ONLY. It was loosely based off the wolfSSL library version at the time of validattion however the wolfSSL version will continue to update as enhancements and fixes are applied in subsequent releases. You will alwasy receive the latest release with the proper wolfCrypt module (v4.5.4 in the exam-ple) inside even though the wolfSSL version (4.8.1 in the example_ continues to update over time! If you have any questions on this, contact us at fips@wolfssl.com or support@wolfssl.com

2.2 How do I know if I am using the FIPS module?

A: Answer, follow the build instructions in the FIPS User Guide (for cert 3389) or the security policy (for cert 2425). At runtime, call an API that only exists in fips like 'wolfCrypt_GetStatus_fips()'.

NOTE: all FIPS bundles ending in -v2 will come with a directory in wolfssl-root like 'v4.1.0.-SP-and-user-guide' which will contain .pdf copies of the Security Policy and User Guide for that FIPS module version

2.3 Does the Power On Self Test (POST) really have to run every time:

- The app restarts?
- For each individual process/application?
- The device power cycles?

A: yes (to all of the above). There is no exception that allows the POST to be skipped. This is a NIST requirement. See https://csrc.nist.gov/CSRC/media/Projects/Cryptographic-Module-Validation-Program/documents/fips140-2/FIPS1402IG.pdf section "9.10 Power-Up Tests for Software Module Libraries" (Excerpt below). "Shall" means "no exceptions"

```
AS09.08: (Levels 1, 2, 3 and 4) Power-up tests shall be performed by a
cryptographic module when the module is powered up (after being
powered off, reset, rebooted, etc.).

AS09.09: (Levels 1, 2, 3 and 4) The power-up tests shall be initiated
automatically and shall not require operator intervention.

TE.09.09.02: The tester shall power-up the module and verify that the
module performs the power-up self-tests without requiring any operator
intervention.
```

2.3.1 Followup Post Q: What about this feature NO_ATTRIBUTE_CONSTRUCTOR? Can I use that to by-pass the POST by not running it in the constructor?

A: Turning this feature on (assuming it does not cause a compile time error) will immediately make the module no longer FIPS compliant.

1. Once this feature is enabled the operating system itself has to be modified and re-compiled, adding a call to wolfCrypt modules fipsEntry() function any time the OS is starting a new process/application/module that uses the crypto module, or if the OS is booting for the first time following a power cycle.

2. This feature is not for allowing the POST to be skipped or optional, it allows for the POST to be called by the OS after an RNG is initialized or a memory zero (that would erase the POST results) has completed. The call to fipsEntry() SHALL still remain beyond operator control (not optional) per TE.09.09.02 in the above question/answer excerpt from https://csrc.nist.gov/CSRC/media/Projects/Cryptographic-Module-Validation-Program/documents/fips140-2/FIPS1402IG.pdf section "9.10 Power-Up Tests for Software Module Libraries".

2.3.2 Followup POST Q: Why is the feature NO_ATTRIBUTE_CONSTRUCTOR there then if I can not use it?

A: It exists for two select exception cases:

1. The first exception is when an operating system would perform a memory zero during an application start that would erase the POST result (pass/fail) and therefore the POST must be called AFTER the memory zero. To remain FIPS compliant the call must be proven to the CMVP to be not optional and not require operator intervention to trigger (has to be called by the OS before control is returnd to the user-space app or kernel module).
2. The second exception is when an entropy source can not be made available to the attribute constructor. In this case the RNG has to be initialized first and then the call to fipsEntry() made. Again the call must be proven to the CMVP to be not optional requiring no operator intervention to trigger (has to be called by the OS before control is returned to the user-space app or kernel module).

2.3.3 Folloup POST Q: Who can determine when NO_ATTRIBUTE_CONSTRUCTOR is allowed?

A: Only wolfSSL staff in collaboration with an NVLAP accredited FIPS lab during an OE operational testing effort can determine the viability of the feature NO_ATTRIBUTE_CONSTRUCTOR. If the feature is warranted for a specific operating environment:

1. It will be noted in the FIPS user guide section for that operating environement
2. The bundle releases the customer receives from wolfSSL will build cleanly with the feature enabled (no compile time error)
3. The default settings for the module will turn the setting on.

2.3.4 Followup Post Q: What about with fips-ready, can I use NO_ATTRIBUTE_CONSTRUCTOR with fips-ready?

A: The term "fips-ready' implies a module is abiding by ALL the FIPS requirements such that it could be submitted to the CMVP without any changes and achieve FIPS certification. Using the feature NO_ATTRIBUTE_CONSTRUCTOR without wolfSSL staff and an NVLAP accredited FIPS lab approving it for that operating environement would imply the fips-ready solution is no longer a fips-ready release, that release would no longer be considered"fips-ready" until reviewed by wolfSSL staff in collaboration with an NVLAP accredited FIPS lab.

2.4 What can go wrong for the end user after basic testing?

A: (Multi-part)

1. The integrity test can fail due to corruption
2. Power On Self Test can fail
    1. This is very unlikely after initial testing.

2. wolfSSL has seen cases where the customer receiving the binary had the wrong hardware for with PAA.
3. wolfSSL has also seen a case where the hardware was correct but the with PAA feature (AESNI) was disabled in the BIOS settings of the end-point system from the factory. In this case, the end customer had to go into the BIOS and enable the AESNI feature on the chip for the FIPS module with PAA to not seg-fault on their system.
4. Windows: ASLR (Address space layout randomization) is enabled on the computer where the test is failing meaning the OS is free to "chop up the FIPS module and scatter it about in memory" and somehow still expect the intgrity hash to be the same value. (This is impossible, please ensure ASLR is disabled on any PC working with the FIPS module or at a minimum ASLR is disabled for the apps/programs consuming the FIPS module actively)
5. Windows: Randomized base addressing in Windows OS. Because the function addresses are included in the integrity check, the module needs to load at the same location in memory each time. There are 2 factors to this one:
    1. Must diable Random base address at the project level. Example: https://github.com/wolfSSL/wolfssl/blob/master/IDE/WIN10/wolfssl-fips.vcxproj#L159
    2. Must assign a physical fixed address (and this is KEY!!) that is NOT in contention with any other DLL on the system. Example: https://github.com/wolfSSL/wolfssl/blob/master/IDE/WIN10/wolfssl-fips.vcxproj#L158

NOTE: If an address is chosen that IS in contention with another DLL on the system, the OS will automatically be allowed to assign a random base address even if random base address is set to false.

HINT for this possibility: To pick a base address not in contention with another DLL the HIGHER in the address space you go the least likely it will be to have a contention.

3. DRBG continuity / health check can fail if connections persist for extended durations.
    1. wolfSSL stringly recommends that (at a MAXIMUM) at least once in a 24 or 48 hour period, persistent connections be shut down and a new handshake performed.
    2. This is a best security practice in general.
    3. An ideal scenario would re-negotiate ephemeral keys every 8-10 minutes (complete and full shutdown and fresh handshake to re-establish the connection).

2.5 Moving from 140-2 to 140-3, what's new?

1. Support for 3DES has been removed, please remove use-cases from applications that involve dependencies on 3DES
2. All applications now need to call wc_SetSeed_Cb now in FIPS mode with the 140-3 module! This should be the first item AFTER setting the FIPS callback that alerts you to any issues with the module and prints the hash out if the integrity check needs updated.

```
/* Old callback same as before */
wolfCrypt_SetCb_fips(myFipsCb);

/* Additional new callback */
+#ifdef WC_RNG_SEED_CB
+   wc_SetSeed_Cb(wc_GenerateSeed);
+#endif
```

1. This is because things are changing with regard to the entropy. Externally loaded entroy must now be explicit and all entropy sources will be validated in an ESV (Entropy Source Validation) stand-alone. FIPS modules will no longer be allowed to make entropy claims on new submissions.

   1. The CMVP was tired of validating and re-validating the same entropy sources over and over. The idea is that now an entropy source will be validated one time and many FIPS modules may now call that entropy source and point to the 1 ESV certificate for any entropy claims.
2. wolfSSL Inc also wanted an option available should one wish to call one of the ESV validated entropy sources so a callback option made sense.

2.6 Will my applications that are linked agaist the 140-2 module still work with the 140-3 module?

A: Absolutely! wolfCrypt FIPS modules v2.x (cert #3389, 140-2 module) has 100% API compatibility minus the 3DES services with wolfCrypt FIPS module v5.2.1 (cert #4718, 140-3 module). In addition to the 140-2 APIs, there are new services (TLS KDFs, AES-OFB, etc) and newer extended APIs. For a full list of new services and new API definitions supported by the 140-3 module please refer to the FIPS 140-3 User Guide [UG] which includes an exhaustive list of every FIPS service and correlated API + description(s). If you need a copy of the [UG] please let the wolfSSL team know by emailing support@wolfssl.com requesting a copy of the FIPS [UG].

2.7 The wc_SetSeed_Cb() callback and the TLS Layer:

If a user-supplied seed routine is to be used in conjunction with the TLS layer, wc_SetSeed_Cb(user_seed_function) must be called after the initial wolfSSL_Init(), and before the application calls wolfSSL_CTX_new().

Example:

```
/* always set first in FIPS */
wolfCrypt_SetCb_fips(myFipsCb);
wolfSSL_Init();
wc_SetSeed_Cb(seedCb);
wolfSSL_CTX_new(&ctx);
...
```

To avoid potential implementation bugs users could alternatively use the setting #define CUSTOM_RAND_GENERATE' and register their own custom RNG seed gen callback.

Example of setting custom seed gen without depending on wc_SetSeed_Cb():

```
extern unsigned int my_rng_seed_gen(void);
#undef CUSTOM_RAND_GENERATE
#define CUSTOM_RAND_GENERATE my_rng_seed_gen
/* Implement my_rng_seed_gen() at the application
level */
```

2.8 The wc_SetSeed_Cb() callback and a custom seed generation function:

To avoid potential implementation bugs users should follow the known good procedure for adding a custom seed function. Step 1) In either user_settings.h or settings.h header add the following:

```
    ```
        /* Seed Source */
        extern unsigned int my_rng_seed_gen(byte* output, word32 sz);
        #undef CUSTOM_RAND_GENERATE_SEED
        #define CUSTOM_RAND_GENERATE_SEED my_rng_seed_gen
    ```

Definition: A ***Consuming Application*** is anything outside the module
   boundary
```

~~~
            that consumes the FIPS 140-3 crypto but is not subject to the
            FIPS 140-3 validation (may be subject to ESV but that is
            separate from 140-3)

Step 2) At the ***Consuming Application*** level implement the callback
   function:

    ```
        /* @param output The buffer to fill with entropy bits one byte at a
           time,
         *           if the solution returns bits instead of bytes be sure
         *           to gather 8 times 'sz' instead of just 'sz'
         * @param sz The number of bytes the output buffer can hold based on
         *           declared size in the Consuming Application
         */
        unsigned int my_rng_seed_gen(byte* output, word32 sz)
        {
            /* Pseudo code */
            fill buffer 'output' with 'sz' bytes of entropy
            if filling fails return the appropriate error code for this system
            otherwise return 0 to indicate success
        }
    ```

Step 3) Finally ***ONLY*** use the wolfSSL supplied callback wc_GenerateSeed()
        as your seeding mechanism. Register it in the ***Consuming Application
           ***
        with:

    ```
        #ifdef WC_RNG_SEED_CB
            wc_SetSeed_Cb(wc_GenerateSeed);
        #else
            #error "Module was not compiled with required setting
               WC_RNG_SEED_CB"
        #endif
    ```
~~~

2.9 The POST

Under 140-2 POST stood for “Power On Self Test” and ran EVERY algorithm self-test leading to slow power-on / boot times. Under 140-3 POST stands for “Pre-Operational Self Test” and only runs the integrity check of the module (and any dependency self-test to support the integrity check). Since HMAC-SHA2-256 self-test must first run and then the integrity check is performed. No other self-tests run at this stage in the 140-3.

2.10 Threading consideration for all CASTs():

Calling a CAST in a thread for the first time or allowing a CAST to run automatically by using a service for the first time in a thread may result in another thread getting a “FIPS_CAST_STATE_PROCESSING” error (meaning that another thread is actively running the CAST) if it attempts to exercise the same CAST in parallel. This will result in the module dropping into the degraded mode of operation.

Once degraded mode is active the only recovery from degraded mode is a power cycle of the module or by re-running the integrity test to simulate a reload/power cycle of the module. To simulate reload or power cycle of the module, shut down all threads then call wolfCrypt_IntegrityTest_fips(); before starting threads up again.

To avoid this problem one can simply call wc_RunAllCast_fips() on startup along with the other FIPS specific initializers.

Example:

```
wolfCrypt_SetCb_fips(myFipsCb);

/* Alternatively call wolfSSL_Init() or comparable API
 * that sets up the seed callback by default
 */
#ifdef WC_RNG_SEED_CB
   wc_SetSeed_Cb(wc_GenerateSeed);
#endif

wc_RunAllCast_fips();
```

Once a CAST has run at least once it will no longer trigger automatically when a service is used. The application may call a CAST at any time (periodically re-check the CAST is still passing) if desired but should always do so in a thread-safe way to avoid thread contention while the CAST is executing.

Another method would be to (at a minimum) run at least the individual CASTs for any algorithms that will be used in threads if you wish to avoid a subset of the CASTs for performance/boot up time considerations where the CAST IDs' are as follows:

- FIPS_CAST_AES_CBC
- FIPS_CAST_AES_GCM
- FIPS_CAST_HMAC_SHA1
- FIPS_CAST_HMAC_SHA2_256
- FIPS_CAST_HMAC_SHA2_512
- FIPS_CAST_HMAC_SHA3_256
- FIPS_CAST_DRBG
- FIPS_CAST_RSA_SIGN_PKCS1v15
- FIPS_CAST_ECC_CDH
- FIPS_CAST_ECC_PRIMITIVE_Z
- FIPS CAST_DH_PRIMITIVE_Z
- FIPS_CAST_ECDSA
- FIPS_CAST_KDF_TLS12
- FIPS_CAST_KDF_TLS13
- FIPS_CAST_KDF_SSH

Example:

```
wolfCrypt_SetCb_fips(myFipsCb);

/* Alternatively call wolfSSL_Init() or comparable API */
#ifdef WC_RNG_SEED_CB
   wc_SetSeed_Cb(wc_GenerateSeed);
#endif

/* Running just one CAST by ID */
if (wc_RunCast_fips(FIPS_CAST_RSA_SIGN_PKCS1v15) != 0){
    err_sys("RSA PKCS v1.5 CAST failed");
}
```

2.11 wc_SetSeedCb() a bit unique with relation to CAST’s:

wc_SetSeed_Cb(); is the first operational use of the DRBG and as such the CAST will run when the callback is set for the first time. To avoid a race condition on the CAST users should set the seed callback one time on startup and not on a per-thread basis or one time globally and then once per thread is also acceptable if the CAST has passed by the time threads are launched. This would be for a scenario where thread-A needs entropy-source-A and thread-B uses a different entropy source. Please remember that calling wolfSSL_Init() will set the seed callback and therefore should not be called on a per-thread basis unless called at least once globally first. A good practice if setting per thread might be:

```
int main(void) {
    int ret;

    /* always call first */
    wolfCrypt_SetCb_fips(fipsCb);

    /* alternatively just call wolfSSL_Init() or equivalent function such as
       wolfMikey_Init() */
#ifdef WC_RNG_SEED_CB
    {
        ret = wc_SetSeed_Cb(wc_GenerateSeed);
        if (ret != 0) {
            printf("wc_SetSeed_Cb failed with %d, %s\n",
                ret,
                wc_GetErrorString(ret));
        }
    }
#endif
}
```

By checking the return value of the call the function should block prior to threads starting up avoiding any race conditions on the CAST completing prior to threads consuming the DRBG.

2.12 Key Access Management

1. Users calling wolfSSL (SSL/TLS) APIs’ do not need to worry about this item
2. Users invoking wolfcrypt (wc_XXX) APIs’ directly that involve loading or using a private key must manage the key access at the application level. To be able to read in or use a private key the application must allow this by calling

```
wolfCrypt_SetPrivateKeyReadEnable_fips(1, WC_KEYTYPE_ALL);
```

prior to loading or accessing a private key.

3. Once complete with key operations users ***shall*** call

```
wolfCrypt_SetPrivateKeyReadEnable_fips(0, WC_KEYTYPE_ALL);
```

to lock key access once again.

4. Macros are available to replace the above long API calls with a shorter macro when including

```
<wolfssl/wolfcrypt/types.h>:
PRIVATE_KEY_UNLOCK() ;
PRIVATE_KEY_LOCK() ;
```

The key access can optionally be unlocked* only once on startup and locked again prior to shutdown** or… If the application wishes to be very strict, these can be called immediately before and after each call that involves a private key load or use. *Be aware that LOCK and UNLOCK are **thread-local**. Aas this is a semaphore, each UNLOCK must be paired with a corresponding LOCK at the same scope to properly decrement the lock count. Alternatively doing a “true lock” (example provided below) may be the best approach for proper lock management. ** “application ***shall*** lock again before terminating” - This is a documentation requirement, this is not enforced at run-time by any error or prevention from exiting. Failing to re-lock key access before exiting makes the application “not FIPS compliant” however.

```
/* true_lock will always decrement the lock counter to 0 regardless of scope
   */
static inline int true_lock(void)
{
    int i;
    int lockStatus = wolfCrypt_GetPrivateKeyReadEnable_fips(WC_KEYTYPE_ALL);
#ifdef VERBOSE_LOGGING
    printf("lockStatus (pre-loop) is %d\n", lockStatus);
#endif
    for (i = lockStatus; i > 0; i--) {
        wolfCrypt_SetPrivateKeyReadEnable_fips(0, WC_KEYTYPE_ALL);
        lockStatus = wolfCrypt_GetPrivateKeyReadEnable_fips(WC_KEYTYPE_ALL);
#ifdef VERBOSE_LOGGING
        printf("lockStatus (loop %d) is %d\n", i, lockStatus);
#endif
    }
    return lockStatus;
}
...
    if (true_lock() != 0) {
        printf("true_lock failed to lock\”n);
        return error_code;
    }
```

5. To support an application that can link to both a wolfSSL FIPS library version and a wolfSSL non-FIPS library version users can implement NO-OP versions of the macros at the application level for the non-FIPS cases like so:

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

2.13 API’s that require UNLOCK before first use (should also be re-LOCKED after use):

```
v5.2.1 (and all other v5.X.X modules)
* wc_PRF
* wc_PRF_TLSv12
* wc_HKDF_Extract
* wc_HKDF_Extract_ex
* wc_HKDF_Expand
* wc_HKDF_Expand_ex
* wc_HKDF
* wc_Tls13_HKDF_Extract
* wc_Tls13_HKDF_Extract_ex
* wc_Tls13_HKDF_Expand_Label
* wc_Tls13_HKDF_Expand_Label_ex
* wc_SSH_KDF
* wc_RsaExportKey
* wc_ecc_export_x963
* wc_ecc_export_ex
* wc_ecc_export_private_only
* wc_ecc_export_private_raw
* wc_ecc_export_x963_ex
* wc_ecc_shared_secret
* wc_ecc_shared_secret_ex
* wc_DhGenerateKeyPair
* wc_DhAgree

v6.0.0 and newer module add some new ones in addition to the above list:
* wc_SRTP_KDF
* wc_SRTCP_KDF
* wc_SRTCP_KDF_ex
* wc_SRTP_KDF_label
* wc_SRTCP_KDF_label
* wc_ed25519_export_private_only
* wc_ed25519_export_private
* wc_ed25519_export_key
* wc_ed448_export_private_only
* wc_ed448_export_private
* wc_ed448_export_key
* wc_PBKDF2_ex
* wc_PBKDF2

v7.0.0 (upcoming)
* Will have some additional services listed here for Post Quantum key material
```