# Unisys Linux Kernel Cryptographic API Module Version 2.0

# FIPS 140-2 Level 1 Validation Non-Proprietary Security Policy

July 13, 2021

# Table of Contents

# 1. Introduction

## 1.1. Document History

| *Authors* | *Date* | *Version* | *Comment* |
|---|---|---|---|
| Unisys Stealth Team | March 7, 2019 | 0.1 | Initial draft. |
| Unisys Stealth Team | June 25, 2019 | 0.2 | Revised with comments. |
| Unisys Stealth Team | June 26, 2020 | 0.3 | Revised with comments. |
| Unisys Stealth Team | July 22, 2020 | 0.4 | Final document. |
| Unisys Stealth Team | July 13, 2021 | 0.5 | Revised final document. |

## 1.2. Purpose

This is the non-proprietary security policy for the Unisys Linux Kernel Cryptographic API Module Version 2.0, which is referred to as *the module*. This document describes how the module meets the security requirements of Federal Information Processing Standards (FIPS) Publication 140-2. This document also describes how to run the module in a secure, FIPS-approved mode of operation. This Policy forms a part of the submission package to the validating lab. The module uses the Unisys Linux strongSwan Cryptographic Module (version 5.6.3-6.4) as a bound module, which is referred to as *the bound module*. The bound module is a FIPS 140-2 validated module (cert. #3971) with CAVS certificate C1012.

FIPS 140-2 specifies the security requirements for a cryptographic module protecting sensitive information. Based on four security levels for cryptographic modules this standard identifies requirements in eleven sections. For more information about the standard visit www.nist.gov/cmvp

The product meets the overall requirements applicable to Level 1 security for FIPS 140-2. The module does not support authentication mechanisms.

**Table 1 – Module Compliance Table**

| Security Component | Security Level |
|---|---|
| Cryptographic Module Specification | 1 |
| Cryptographic Module Ports and Interfaces | 1 |
| Roles, Services, and Authentication | 1 |
| Finite State Model | 1 |
| Physical Security | N/A |
| Operational Environment | 1 |
| Cryptographic Key Management | 1 |
| EMI/EMC | 1 |
| Self-Tests | 1 |
| Design Assurance | 1 |
| Mitigation of Other Attacks | N/A |

# 2. Cryptographic Module Description

The module meets overall Level 1 requirements for FIPS 140-2, and the following table describes the level achieved by the module in each of the eleven sections of the FIPS 140-2 requirements.

The following table describes the multi-chip standalone platforms on which the module has been tested. It includes processor options with and without AES-NI, PCLMULQDQ, and SSSE3, in combinations that can be invoked.

**Table 2 – Tested Operational Environments**

| Manufacturer | Model | Operating System |
|---|---|---|
| Intel® Xeon® Gold 5115 Processor | Dell PowerEdge R640 Server | Ubuntu 18.04 LTS Server distribution |

The module is a software-only cryptographic module that comprises a set of Linux kernel modules. It provides general purpose cryptographic services to the remainder of the Linux kernel.

The module performs a software integrity check on itself using an HMAC SHA-512. The Linux kernel is configured so that the Linux kernel modules are loaded separately from other kernel functions. Only FIPS-approved and validated algorithms are loadable.

## *2.1. Cryptographic Boundary*

The module is a software-only cryptographic module that comprises a set of Linux kernel modules; this set defines the module's cryptographic boundary. It provides cryptographic functionality for any application that calls into it. The module is embodied by the Linux kernel modules implementing the ciphers in /lib/modules/4.15.0-54-stealth/kernel/crypto and /lib/modules/4.15.0-54-stealth/kernel/arch/x86/crypto. Only the Linux kernel modules implementing the approved mechanisms are available and loaded at boot time.

The Linux kernel modules are:

/lib/modules/4.15.0-54-stealth/kernel/crypto/:

| | | | |
|---|---|---|---|
| 842.ko | ablk_helper.ko | aead.ko | aes_generic.ko |
| aes_ti.ko | af_alg.ko | akcipher.ko | algif_aead.ko |
| algif_hash.ko | algif_rng.ko | algif_skcipher.ko | ansi_cprng.ko |
| asymmetric_keys | async_tx | authencesn.ko | authenc.ko |
| cbc.ko | ccm.ko | cfb.ko | cryptd.ko |
| crypto_acompress.ko | crypto_algapi.ko | crypto_blkcipher.ko | crypto_engine.ko |
| crypto_hash.ko | crypto.ko | cryptomgr.ko | crypto_null.ko |
| crypto_simd.ko | crypto_wq.ko | ctr.ko | deflate.ko |
| drbg.ko | ecb.ko | echainiv.ko | fipsavs.ko |
| gcm.ko | gf128mul.ko | ghash-generic.ko | hmac.ko |
| hw_jitter_rng.ko | jitterentropy_rng.ko | keywrap.ko | kpp.ko |
| mcryptd.ko | ofb.ko | pcrypt.ko | poly1305_generic.ko |
| rng.ko | rsa_generic.ko | seqiv.ko | sha1_generic.ko |
| sha256_generic.ko | sha3_generic.ko | sha512_generic.ko | sm3_generic.ko |
| tcrypt.ko | xor.ko | xts.ko | |

/lib/modules/4.15.0-54-stealth/kernel/crypto/asymmetric_keys:
asymmetric_keys.ko

/lib/modules/4.15.0-54-stealth/kernel/crypto/async_tx:

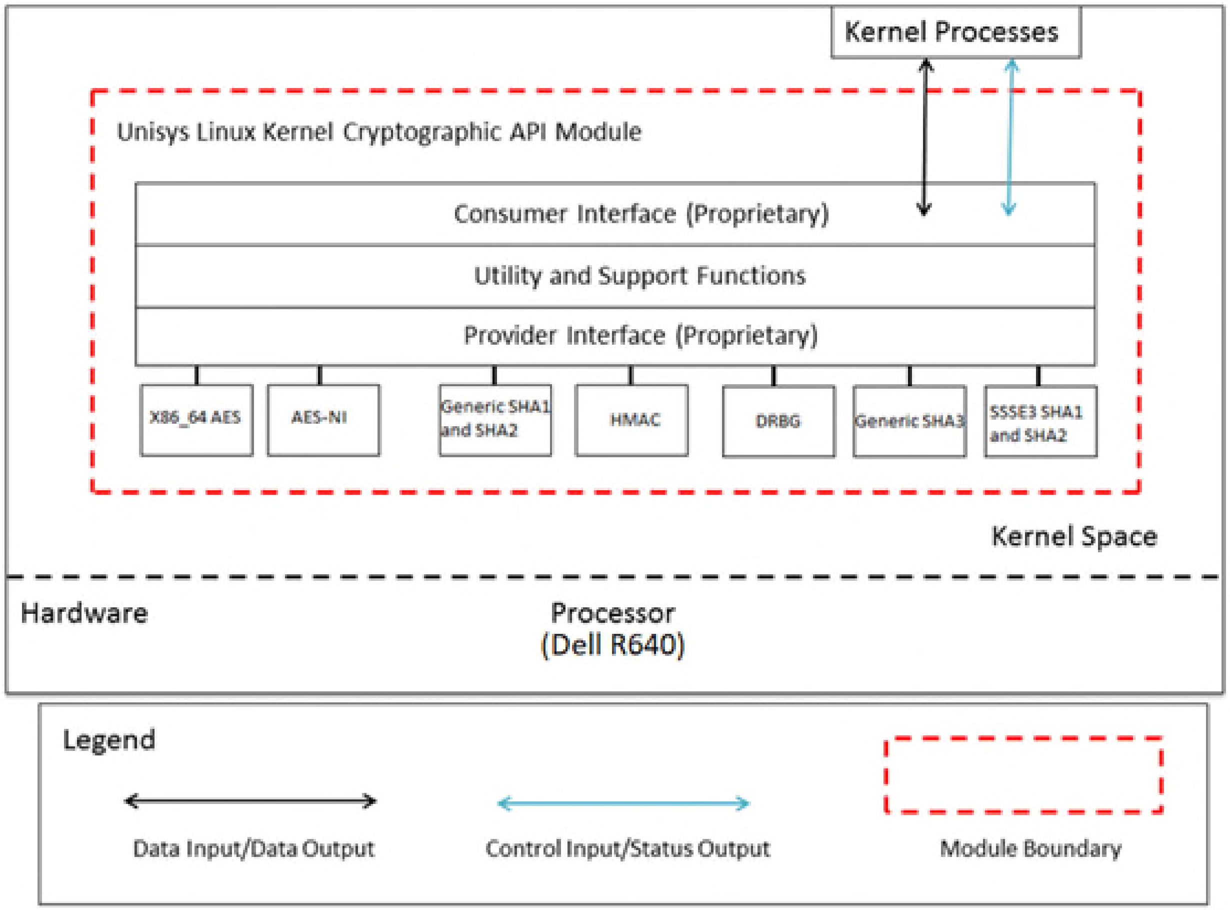
| | | | |
|---|---|---|---|
| async_memcpy.ko | async_pq.ko | async_raid6_recov.ko | async_tx.ko |
| async_xor.ko | | | |

/lib/modules/4.15.0-54-stealth/kernel/arch/x86/crypto/:

| | | | |
|---|---|---|---|
| aesni-intel.ko | aes-x86_64.ko | ghash-clmulni-intel.ko | glue_helper.ko |
| poly1305-x86_64.ko | sha1-ssse3.ko | sha256-ssse3.ko | sha512-ssse3.ko |

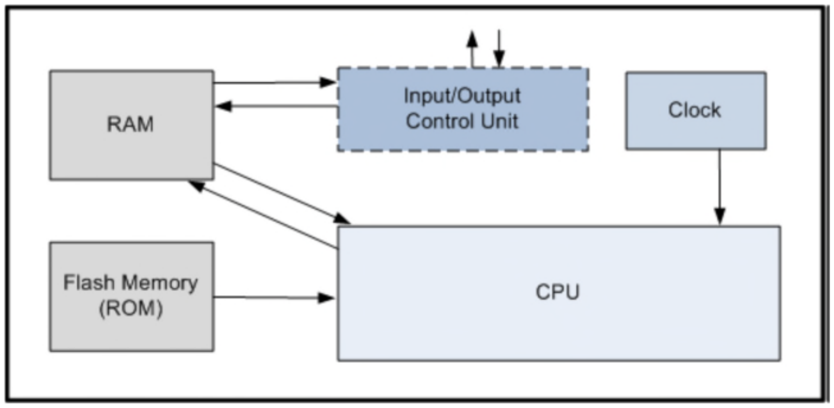
Figure 1 is the software block diagram of the module, and it illustrates the module boundary. The proprietary portions of the Consumer and Provider interfaces are contained in the Linux kernel modules and implement the described and approved mechanisms for the software integrity check, the valid modes of operation, and the self-tests.

**Figure 1 – Software Block Diagram**



The physical boundary of the module is defined by the surface of the case of the platform. Figure 2 illustrates the hardware block diagram that comprises the platform.

**Figure 2 – Hardware Block Diagram**



## 2.2. Description of Modes of Operation

The module supports only a FIPS-approved mode, and the module must always be configured as described in 10.1, "Secure Setup."

The module supports the following approved functions listed in Table 3:

**Table 3 – FIPS-approved Algorithm Implementations**

| Algorithm | Modes | Certificate Number |
|---|---|---|
| Generic implementation of AES | CBC, CCM, CFB, CTR, ECB, GCM, OFB encrypt/decrypt (128, 192, 256-bits)<br>XTS encrypt/decrypt (128bits) | C873 |
| AES-NI implementation of AES | CBC, CFB, CTR, ECB, GCM, OFB encrypt/decrypt (128, 192, 256-bits)<br>XTS encrypt/decrypt (128bits) | |
| Generic implementation of SHA | SHA-1, SHA-224, SHA-256, SHA-384, SHA-512, SHA3-224, SHA3-256, SHA3-384, SHA3-512 | |
| SSSE3 implementation of SHA | SHA-1, SHA-224, SHA-256, SHA-384, SHA-512 | |
| HMAC | SHA-1, SHA-224, S HA-256, SHA-384, SHA-512, SHA3-224, SHA3-256, SHA3-384, SHA3-512 | |
| | | |

| | | |
|---|---|---|
| RSA | 2048, 3072-bit modulus, PKCS#1 1.5 Signature Verification (SHA-256, SHA-512) | |

The module also implements cipher algorithms other than those listed previously. These ciphers are technically unavailable. When calling these ciphers, the module returns an error.

The module maintains a process flag to indicate that the module is in a FIPS-approved mode. The flag is provided in the file /proc/sys/crypto/fips_enabled. If this file contains a value of 1, the module is operational in a FIPS-approved mode. If it contains a value of 0, then the FIPS-approved mode was disabled. This indicates an error condition. The Crypto-Officer must enable the FIPS-approved mode (for example, by reinstalling the module), and the operating system must be rebooted. If the power-up self-tests failed, the module will enter an error mode, and the Crypto-Officer must reboot the operating system to perform power-up self-tests. See Section 9, "Self-tests," and Section 10, "Crypto-Officer and User Guidance," for more information.

The bound module provides the following approved functions in Table 4 which the module utilizes:

**Table 4 – FIPS-approved Algorithm Implementations from Bound Module**

| Algorithm | Modes | Certificate Number |
|---|---|---|
| SP 800-135rev1 IKEv2 KDF | HMAC with SHA-1, SHA-256, SHA-384 and SHA-512 | C1012 |

***Note:*** *The bound module supports additional algorithms not listed in the above table. The module only utilizes the algorithms which are listed in the table.*

## 2.3. AES Implementations

The module supports the following two implementations of AES:

- AES using the new Intel instruction set when the aesni-intel kernel module is loaded (which is only used if the underlying processor provides the AES-NI and PCLMULQDQ instruction sets).
- AES implemented with generic C code when the generic AES kernel modules are loaded.

Note that if more than one of the previously listed kernel modules are loaded, the respective implementation can be requested by using the following cipher mechanism strings with the initialization calls (for example, crypto_alloc_skcipher):

- aesni-intel kernel module: “aes-aesni”
- Generic AES kernel module: “aes-generic”
- Automation selection: “aes”

Mode chaining also follows this convention. The algorithm “cbc(aes-generic)” is used to get the generic AES implementation, and “cbc(aes-aesni)” is used to get the AES-NI implementation. The generic CBC implementation is used in either case. The algorithm “cbc-aes-aesni” is used to get an entirely AES-NI implementation. The algorithm “cbc(aes)” automatically selects the implementation for both CBC and

AES.

For example, if the kernel module aesni-intel is loaded, and if the caller uses the initialization call (for example, crypto_alloc_blkcipher) with the cipher string of "aes", then the aesni-intel implementation is used. Or, if only the aes-x86_64 kernel module is loaded, the cipher string of "aes" implies that the aes-x86_64 implementation is used.

The discussion about the naming and priorities of the AES implementation also applies when cipher strings are used that include the block chaining mode, such as "cbc(aes)", "cbc(aes-generic)", or "cbc(aes-aesni)".

The full list of algorithms and implementations is provided in the file /proc/crypto.

## *2.4. SHA1 and SHA2 Implementations*

The module supports the following two implementations of SHA1 and SHA2 as follows:

- SHA using the new Intel instruction set when the SSSE3 kernel modules are loaded (which is only used if the underlying processor provides the SSSE3 instruction set).

- SHA implemented with generic C code when the generic SHA1 and SHA2 kernel modules are loaded.

Note that if more than one of the previously listed kernel modules are loaded, the respective implementation can be requested by using the following cipher mechanism strings with the initialization calls (for example, crypto_alloc_ahash):

- SSSE3 kernel modules: “sha*X*-ssse3”

- Generic SHA kernel module: “sha*X*-generic”

- Automation selection: “sha*X*”

Where *X* is: 1, 224, 256, 384, or 512.

Mode chaining also follows this convention. The algorithm “hmac(sha1-generic)” is used to get the generic SHA1 implementation and “hmac(sha1-ssse3)” is used to get the SSSE3 implementation.
The full list of algorithms and implementations is provided in the file /proc/crypto.

# 3. Module Ports and Interfaces

The module is considered to be a multi-chip standalone module designed to meet FIPS 140-2 Level 1 requirements. The physical ports of the module are the same as the computer system on which the software module is executing. The logical interface is an application program interface (API) as shown in Table 5.

**Table 5 – Mapping Physical and Logical Interfaces**

| FIPS 140-2 Logical Interface | Module Logical Interface | Physical Ports |
|---|---|---|
| Data Input | Consumer Interface | SAS port, DVD port, Network Port, USB port, Serial Port |

| | | |
|---|---|---|
| Data Output | Consumer Interface | SAS port, Network Port, USB port, Serial Port |
| Control Input | Consumer Interface | Network Port, USB port, Serial Port |
| Status Output | Consumer Interface | SAS port, Network Port, USB port, Serial Port status LEDs, Network Port status LEDs, Video port |
| Power Input | Not Applicable | Power Supply |

When the module is performing self-tests, all output on the logical data output interface is inhibited by sequencing the loading of self-tests and software modules. When the module is in an error state, all output on the logical data output interface is inhibited, because the module forces a kernel panic. See Section 9.1, "Power-up Self-tests" for more information. As a software module, it cannot control the physical ports.

# 4. Roles, Services, and Authentication

There are two roles in the module (as required by FIPS 140-2) that operators may assume: a Crypto-Officer role and a User role. The Crypto-Officer and User roles are implicitly assumed by the entity accessing the services implemented by the module. No further authentication is required for a Level 1 validation. The module does not allow concurrent operators.

The following section describes the services available to each role, and each service's corresponding interface, which is depicted in Figure 1.

This module supports a Crypto-Officer role and a User role.

## *4.1 Crypto-Officer Role*

The Crypto-Officer is any operator on the host appliance with the permissions to check the status of the module. Descriptions of the services available to the Crypto-Officer role are provided in Table 6. The Crypto-Officer also has access to all User services, as described in Table 7.

Note that the Type of Access to CSP column in Table 6 and Table 7 indicates the type of access each service has to its Critical Security Parameter (CSP) using the following notation:

- R – Read: The CSP is read.
- W – Write: The CSP is established, generated, modified, or zeroized.
- X – Execute: The CSP is used within an approved or allowed security function or authentication mechanism.

For more information on each CSP, see Section 7.1, "Critical Security Parameters."

**Table 6 – Crypto-Officer Services**

| Service | Description | Keys/CSPs | Type of Access to CSP | API Calls |
|---|---|---|---|---|

| | | | | |
|---|---|---|---|---|
| Initialize FIPS-approved mode | Performs integrity check and power-up self-tests. Sets the FIPS-approved mode flag to on. | N/A | N/A | N/A |
| Run self-tests | Restarting the appliance will force the self-tests to run when the module is loaded. | HMAC Integrity Key | R, X | N/A |
| Show Status | Uses the "/opt/unisys/fips status" command to return the current status of the module from the dmesg log file | N/A | N/A | N/A |
| Zeroize keys | Cycling the power zeroizes and de-allocates memory containing sensitive data. | All keys/CSPs | W | N/A |

The credentials for the Crypto-Officer are not considered CSPs, as requirements for module authentication are not enforced for Level 1 validation. The credentials are provided to the host operating system, and are not part of the module.

## 4.2 User Role

The User role is able to utilize the cryptographic operations of the module through its APIs. Descriptions of the services available to the User role are provided in Table 7.

**Table 7 – User Services**

| Service | Description | Keys/CSPs | Type of Access to CSP | API Calls |
|---|---|---|---|---|
| Encryption/ Decryption | Encrypt or decrypt a block of data using a symmetric algorithm. | AES key | RWX | crypto_alloc_skcipher<br>crypto_free_skcipher<br>crypto_has_skcipher<br>crypto_skcipher_ivsize<br>crypto_skcipher_blocksize<br>crypto_skcipher_setkey<br>crypto_skcipher_reqftm<br>crypto_skcipher_encrypt<br>crypto_skcipher_decrypt<br>crypto_skcipher_reqsize<br>crypto_request_set_tfm<br>crypto_request_alloc<br>crypto_request_free<br>crypto_request_set_callback<br>crypto_request_set_crypt |
| Signature verification | Verify a digital signature using an asymmetric algorithm. | RSA public key | RX | crypto_alloc_akcipher<br>crypto_free_akcipher<br>crypto_akcipher_set_pub_key<br>crypto_akcipher_maxsize<br>crypto_akcipher_verify<br>akcipher_request_alloc |

| Service | Description | Keys/CSPs | Type of Access to CSP | API Calls |
|---|---|---|---|---|
| | | | | akcipher_request_free<br>akcipher_request_set_callback<br>akciper_request_set_crypt |
| Authenticated Encryption with Associated Data (AEAD) | A combined cryptographic protocol that only supports the approved algorithms used in the module. | AES key, AES-GCM IV | RWX | crypto_alloc_aead<br>crypto_free_aead<br>crypto_aead_ivsize<br>crypto_aead_authsize<br>crypto_aead_blocksize<br>crypto_aead_setkey<br>crypto_aead_setauthsize<br>crypto_aead_encrypt<br>crypto_aead_decrypt<br>crypto_aead_reqsize<br>aead_request_set_tfm<br>aead_request_alloc<br>aead_request_free<br>aead_request_set_callback<br>aead_request_set_crypt<br>aead_request_set_ad |
| Hashing | Perform a hashing operation on a block of data, using SHA-1, SHA-224, SHA-256, SHA-384, SHA-512, SHA3-224, SHA3-256, SHA3-384, or SHA3-512. | N/A | N/A | crypto_alloc_ahash<br>crypto_free_ahash<br>crypto_ahash_digestsize<br>crypto_ahash_statesize<br>crypto_ahash_reqtfm<br>crypto_ahash_reqsize<br>crypto_ahash_setkey<br>crypto_ahash_finup<br>crypto_ahash_final<br>crypto_ahash_digest<br>crypto_ahash_export<br>crypto_ahash_import<br>crypto_ahash_init<br>ahash_request_set_tfm<br>ahash_request_alloc<br>ahash_request_free<br>ahash_request_set_callback<br>ahash_request_set_cryot<br>crypto_alloc_shash<br>crypto_free_shash<br>crypto_shash_blocksize<br>crypto_shash_digestsize<br>crypto_shash_decsize<br>crypto_shash_setkey<br>crypto_shash_digest<br>crypto_shash_export<br>crypto_shash_import<br>crypto_shash_init<br>crypto_shash_update<br>crypto_shash_final<br>crypto_shash_finup |

| Service | Description | Keys/CSPs | Type of Access to CSP | API Calls |
|---|---|---|---|---|
| HMAC signing | Perform a hashing operation on a block of data, using a keyed Hashed Message Authentication Code with SHA-1, SHA-224, SHA-256, SHA-384, SHA-512, SHA3-224, SHA3-256, SHA3-384, or SHA3-512. | HMAC key | RWX | crypto_alloc_ahash<br>crypto_free_ahash<br>crypto_ahash_digestsize<br>crypto_ahash_statesize<br>crypto_ahash_reqtfm<br>crypto_ahash_reqsize<br>crypto_ahash_setkey<br>crypto_ahash_finup<br>crypto_ahash_final<br>crypto_ahash_digest<br>crypto_ahash_export<br>crypto_ahash_import<br>crypto_ahash_init<br>ahash_request_set_tfm<br>ahash_request_alloc<br>ahash_request_free<br>ahash_request_set_callback<br>ahash_request_set_cryot<br>crypto_alloc_shash<br>crypto_free_shash<br>crypto_shash_blocksize<br>crypto_shash_digestsize<br>crypto_shash_decsize<br>crypto_shash_setkey<br>crypto_shash_digest<br>crypto_shash_export<br>crypto_shash_import<br>crypto_shash_init<br>crypto_shash_update<br>crypto_shash_final<br>crypto_shash_finup |
| | | | | |

## 5. Physical Security

This is a software module and provides no physical security.

# 6. Operational Environment

This module will operate in a modifiable operational environment per the FIPS 140-2 definition.

The operating system shall be restricted to a single operator mode of operation (that is, concurrent operators are explicitly excluded).

The external application that makes calls to the cryptographic module is the single user of the cryptographic module, even when the application is serving multiple clients.

# 7. Cryptographic Key Management

## *7.1 Critical Security Parameters*

The module supports the CSPs listed in Table 8.

**Table 8 – Listing of Key and Critical Security Parameters**

| Key or CSP | Key/IV Type | Generation/ Entry | Output | Storage | Zeroization | Use |
|---|---|---|---|---|---|---|
| AES key | AES 128-, 192-, 256-bit key | Input via API in plaintext. | Never | The module does not store keys. | Reboot operating system; Cycle host power | Encryption/ Decryption |
| AES-GCM IV | AES-GCM Initialization Vector as per RFC 4106 | Internally constructed 96-bit IV: 64-bit invocation counter, 32-bit context value concatenated. | Never | The module does not store IVs. | Reboot operating system; Cycle host power | IV input to AES GCM function |
| HMAC key | HMAC key | Input via API in plaintext | Never | The module does not store keys. | Reboot operating system; Cycle host power | Message Integrity/ Authentication with SHS |
| RSA public key | RSA 2048-bit key | RSA key pair is maintained per company policy. The public key is inserted into read-only data section of the kernel dynamic module integrity check code during the build. | Never | During integrity check initialization, the public key is loaded into the system trusted keyring. | N/A | PKCS 1.5 kernel dynamic module integrity check |

| Key or CSP | Key/IV Type | Generation/ Entry | Output | Storage | Zeroization | Use |
|---|---|---|---|---|---|---|
| HMAC Integrity Key | HMAC key | Hardcoded (pre-loaded) into Module binary at factory. | Never | Hardcoded into Module binary | N/A | Module power-on integrity test |

***Note:*** *The fixed key lengths for HMAC are equal to the block size of the underlying hash function (that is, the fixed key length for the SHA-1, SHA-224, SHA-256, is 64 bytes; SHA-384, SHA-512, SHA-512/224, SHA-512/256, is 128 bytes; SHA3-224 is 144 bytes; SHA3-256 is 136 bytes; SHA3-384 is 104 bytes; and SHA3-512 is 72 bytes).*

## 7.2 Random Number Generation

The module provides the following:

- A non-approved RNG, the *jitterentropy_rng* module, which uses operating system collected hardware jitter to generate random bits.

- A non-approved Deterministic Random Bit Generator (DRBG) based on [SP800-90A]. The DRBG supports the Hash_DRBG, HMAC_DRBG and CTR_DRBG mechanisms.

The module uses a non-approved RNG, the jitterentropy_rng module, as one of the entropy sources for seeding the DRBG. The jitterentropy_rng RNG provides at least 128 bits of entropy to the DRBG during initialization (seed) and reseeding (reseed).

The module uses a non-approved RNG, the kernel urandom RNG, as the second of the entropy sources for seeding the DRBG. The urandom RNG provides at least 128 bits of entropy to the DRBG during initialization (seed) and reseeding (reseed).

The module performs continuous self-tests on the output of RNG jitterentropy_rng to ensure that consecutive random numbers do not repeat.

The Module performs DRBG self-tests as defined in section 11.3 of [SP800-90A]. The testing is performed once when the DRBG software module is initialized. Subsequent self-tests, as defined in section 11.3 of [SP800-90A], are not performed because it is mathematically impossible that the initial self-tests were successful and subsequent tests are not successful.

## 7.3 Key Generation

Keying material may not be generated with the SP 800-90A DRBG. Keying material may be entered into the module via API. The module does not support any explicit key generation functions.

## 7.4 Key Entry and Output

Keys are passed into the module's logical boundary in plaintext via the exposed APIs, but only from applications resident on the host platform. However, the module does not support key entry or key output across the host platform's physical boundary. Similarly, keys and CSPs exit the module in plaintext (but remain in the physical boundary) via the well-defined exported APIs.

## 7.5 Key Storage

Keys are not persistently stored by the module.

### 7.6 Key Zeroization

The module does not persistently store keys (with the exception of the module integrity key). Keys are provided to the module by the calling application and are destroyed when released by the appropriate API function calls. No keys enter or exit the physical boundary of the module's tested platform. All memory is managed by the host operating system. Volatile memory used to store keys and CSPs is zeroized (destroyed) by power-cycling the host platform.

# 8. Electromagnetic Interference and Electromagnetic Compatibility

The module's electromagnetic interference (EMI) and electromagnetic compatibility (EMC) features are summarized in Table 9.

**Table 9 – Electromagnetic Interference and Compatibility**

| Testing Platform | Model Number | EMI/EMC Notes |
|---|---|---|
| Intel Xeon Gold 5115 Processor | Dell PowerEdge R640 Server | FCC Class A |

# 9. Self-tests

In order to prevent any secure data from being released, it is important to test the cryptographic components of a security module to ensure all components are functioning correctly. All kernel modules are loaded as a part of the operating system boot sequence, and power-up self-tests are performed automatically by the module, without requiring any operator intervention.

### 9.1 Power-up Self-tests

To confirm correct functionality, the software library performs the following self-tests:

- Software Integrity Tests using a HMAC SHA-512 and RSA PKCS#1 1.5 on all of the module's components
- Known Answer Tests (KATs)
    - AES (all supported modes) encrypt KATs
    - AES (all supported modes) decrypt KATs
    - SHA (SHA-1, SHA-224, SHA-256, SHA-384, SHA-512, SHA3-224, SHA3-256, SHA3-384, SHA3-512) KATs
    - HMAC (SHA-1, SHA-224, SHA-256, SHA-384, SHA-512, SHA3-224, SHA3-256, SHA3-384, SHA3-512) KATs
    - DRBG (CTR, Hash, HMAC) KATs

  - RSA PKCS#1 1.5 Signature Verification (SHA-256, SHA-512) KATs

- DRBG tests based on section 11.3 of [SP800-90A]

Data output from the module's data output interface is inhibited while performing self-tests. We are relying on the kernel initialization being single threaded. We insure this by using a hook script in the initrd that stops kernel initialization until after the module has loaded and completed self-tests. All kernel object modules must pass power-up self-tests before the system is allowed to enter any user modes. We have added dependencies to the boot process that assures that initialization stalls until crypto self-tests are complete and the module is loaded into the kernel. If any of the power-up self-tests fail, the module enters an error state and ceases operation, inhibiting any further data output from the module. The module does not perform any cryptographic operations while in an error state. When entering an error state, the module forces the kernel to panic. If the boot process is subverted in some way that allows the module to start, the module will still enter an error state and the kernel will panic during the crypto_init phase of any API call by a user. After self-tests are complete, they are disabled through the use of a compliance flag. Subsequent attempts to run self-tests will have no effect, and an error indicator will be returned to the user.

If the module enters an error state, the Crypto-Officer must reboot the system to perform power-up self-tests. Successful completion of the power-up self-tests will return the module to normal operation.

## *9.2 Continuous Self-tests*

The module performs continuous self-tests on the output of non-approved RNG jitterentropy_rng to ensure that consecutive random numbers do not repeat.

# 10. Crypto-Officer and User Guidance

The module consists of several Linux kernel object modules that provide cryptographic services as part of the Unisys Stealth Secure Virtual Gateway software appliance.

The sections below describe how to install, configure, and keep the module in a FIPS-approved mode of operation.

## *10.1 Secure Setup*

To operate the module, the operating system must be restricted to a single-user mode of operation.

Installation and operation of the module requires the proper installation of the following Linux Kernel Debian packages:

| | |
|---|---|
| - linux-buildinfo | 4.15.0-54-stealth |
| - linux-cloud-tools | 4.15.0-54-stealth |
| - linux-headers | 4.15.0-54-stealth |
| - linux-image | 4.15.0-54-stealth |
| - linux-modules | 4.15.0-54-stealth |
| - linux-tools | 4.15.0-54-stealth |
| - fips-support | 1.0.2 |

The ptrace(2) system call, the debugger (gdb(1)), and strace(1) shall not be used. In addition, other tracing mechanisms offered by the Linux environment, such as ftrace or systemtap shall not be used.

The operating system command line parameter "fips=1" shall not be modified. Changing the parameter will disable FIPS-approved mode of operation.

### *10.2 Initialization*

The module is initialized during the operating system boot sequence, before any cryptographic functionality is available. The module is designed with a default entry point (DEP) that ensures that the power-up self-tests are initiated automatically when the module is loaded.

The module enters a FIPS-approved mode upon successful completion of the self-tests. If the self-tests fail, the module will enter an error mode, and the Crypto-Officer must reboot the system to perform power-up self-tests. Successful completion of the power-up self-tests will return the module to normal operation.

### *10.3 AES-GCM Key/IV Usage*

The module generated IVs for use in the AES-GCM algorithm. The IVs are generated in compliance with RFC 4106. The bound module implements RFC 7296 compliant IKEv2 to establish the shared secret SKEYSEED from which the module's AES-GCM encryption keys are derived. The module implements a 64-bit counter i.e. nonce.

All the keys and the constructed IVs used are ephemeral and have a limited lifetime. When the host platform is powered off or rebooted, these keys and encryption contexts are destroyed. New encryption contexts need to be created by the calling application when the operating system is rebooted.

To ensure the uniqueness of the AES-GCM key/IV pair for each encryption sent to the module, users of the module shall not reuse keys between encryption contexts, even those on separate host systems. Techniques for achieving this are documented in Section 7, "Generation of Keys for Symmetric-Key Algorithms" in NIST Special Publication 800-133.

If the same encryption context is used more than $2^{64}$-1 times, the encryption operation will fail and a new encryption context must be established.

### *10.4 AES-XTS*

The AES-XTS algorithm is only used for cryptographic protection of storage devices. Only 128-bit keys can be used. The largest data unit which can be encrypted is 16 MB.  The module implements a check to insure the two AES keys are not the same.

# 11. Mitigation of Other Attacks

This section is not applicable. The module does not claim to mitigate any attacks beyond the FIPS 140-2 Level 1 requirements for this validation.

# Appendix A. Glossary and Abbreviations

- AES – Advanced Encryption Standard
- AES-NI – Advanced Encryption Standard New Instruction set
- API – Application Program Interface
- CBC – Cipher Block Chaining

- CCM – Counter with CBC-MAC
- CFB – Cipher Feedback
- CKG – Cryptographic Key Generation
- CMVP – Cryptographic Module Validation Program
- CSP – Critical Security Parameter
- CTR – Counter
- ECB – Electronic Code Book
- GCM – Galois/Counter Mode
- GMAC – Galois Message Authentication Code
- HMAC – Hash Message Authentication Code
- IV – Initialization Vector
- KAT – Known Answer Test
- MAC – Message Authentication Code
- NIST – National Institute of Science and Technology
- OFB – Output Feedback
- OS – Operating System
- PCLMULQDQ – Carry-less Multiplication Quadword
- PKCS – Public Key Cryptography Standards
- RSA - Rivest–Shamir–Adleman public key cryptographic system
- SHA – Secure Hash Algorithm
- SHS – Secure Hash Standard
- SSSE3 – Supplemental Streaming SIMD Extensions 3
- XTS – XEX-based tweaked-codebook mode with ciphertext stealing