Unisys Linux Kernel Cryptographic API Module Version 1.0

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

February 11, 2016

Table of Contents

Figures

Tables

1. Introduction

This is the non-proprietary security policy for the Unisys Linux Kernel Cryptographic API Module Version 1.0, which is referred to as *the module*. This document describes how the module meets the security requirements of Federal Information Processing Standards (FIPS) Publication140-2. This document also describes how to run the module in a secure, FIPS-approved mode of operation.

2. Cryptographic Module Specification

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.

Table 1 – Security Levels

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

The following table describes the multi-chip standalone platforms on which the module has been tested.

Table 2 – Tested Operational Environments

Manufacturer	Model	Operating System
Intel® Pentium® Processor G3420	R220	Ubuntu 12.04 LTS distribution
Intel Pentium Processor G3420 (with PCLMULQDQ and SSSE3)	R220	Ubuntu 12.04 LTS distribution
Intel Xeon® Processor E5-2697 v3 (with AES-NI, PCLMULQDQ, and SSSE3)	R630	Ubuntu 12.04 LTS distribution
Intel Xeon Processor E5-2697 v3 (with AES-NI and PCLMULQDQ)	R630	Ubuntu 12.04 LTS distribution

Intel Xeon Processor E5-4627 v2 (with AES-NI, PCLMULQDQ, and SSSE3)	R820	VMware ESXi 5.5, Ubuntu 12.04 LTS distribution
Intel Xeon Processor E5-4627 v2 (with AES-NI and PCLMULQDQ)	R820	VMware ESXi 5.5, Ubuntu 12.04 LTS distribution

2.1 Module Description

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.

2.2 Description of Modes of Operation

The module supports only a FIPS-approved mode, and the module must always be configured as described in Section 11.

The module supports the following approved functions:

- AES for x86_64 – supporting 128-, 192-, and 256-bit keys in x86-64 assembly and C code (Cert. #3513)
- AES using AES-NI – supporting 128-, 192-, and 256-bit keys in x86-64 assembly and C code using the AES-NI instruction set when the underlying processor supports AES-NI (see Section 2.3 for more information) (Cert. #3519)
- Generic implementation of SHA – supporting SHA-1, SHA-224, SHA-256, SHA-384, and SHA-512 in C code (Cert. #2901)
- SSSE3 implementation of SHA – supporting SHA-1, SHA-224, SHA-256, SHA-384, and SHA-512 in x86-64 assembly and C code when the underlying processor supports the SSSE3 instruction set (Cert. #2900)
- HMAC using any of the secure hashing algorithms listed previously (Cert. #2247 and #2246)

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 power-up self-tests failed, and the system must be rebooted. See Section 9 and Section 11 for more information.

2.3 AES Implementations

The module supports two implementations of AES as follows:

- AES using the new Intel instruction set when the aesni-intel kernel module is loaded (this is only used if the underlying processor provides the AES-NI instruction set)
- AES implemented with streamlined assembler code when the aes-x86_64 kernel module is 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_blkcipher):

- aesni-intel kernel module: “__aes-aesni”
- aes-x86_64 kernel module: “aes-asm”

For example: If the kernel modules aesni-intel and aes-asm are loaded, and 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.

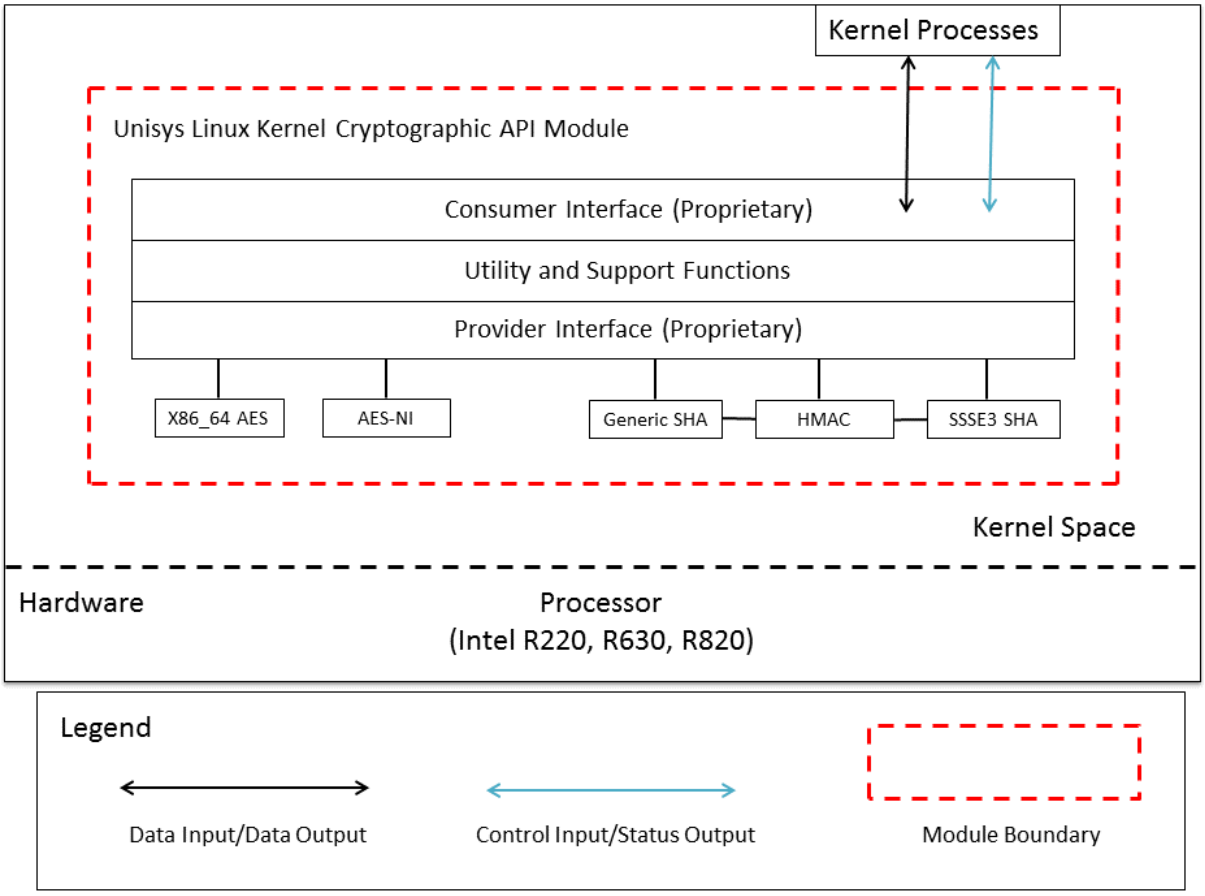
The discussion about the naming and priorities of the AES implementation also applies when cipher strings are used that include the block chaining mode, such as "cbc(aes-asm)", "cbc(aes)", or "cbc(__aes-aesni)".

2.4 Module 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/$(uname -r)/kernel/crypto and /lib/modules/$(uname -r)/kernel/arch/x86/crypto. Only the Linux kernel modules implementing the approved mechanisms are available and loaded at boot time.

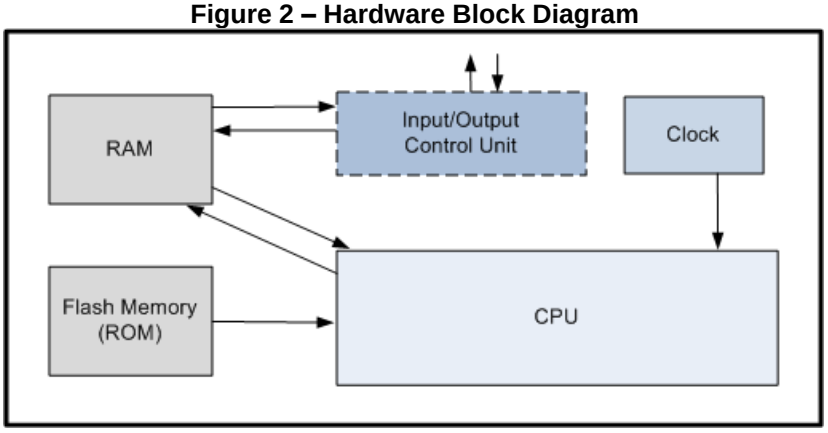
The following figure, Figure 1, is the software block diagram of the module, and it illustrates the module boundary.

Figure 1 – Software Block Diagram



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

Figure 2 – Hardware Block Diagram



3. Ports and Interfaces

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 the following table, Table 3.

Table 3 – Ports and Interfaces

FIPS 140-2 Logical Interface	Module Logical Interface	Physical Ports
Data Input	Provider Interface and Consumer Interface	SAS port, DVD port, Network Port, USB port, Serial Port
Data Output	Provider Interface and Consumer Interface	SAS port, Network Port, USB port, Serial Port
Control Input	Provider Interface and Consumer Interface	Network Port, USB port, Serial Port
Status Output	Provider Interface and Consumer Interface	SAS port, Network Port, USB port, Serial Port status LEDs, VGA port
Power Input	Not Applicable	Power Supply

When the module is performing self-tests or is in an error state, all output on the logical data output interface is inhibited. As a software module, it cannot control the physical ports.

4. Roles and Services

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.

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 4. The Crypto-Officer also has access to all User services, as described in Table 5.

Note that the last column in Table 4 and Table 5 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.

Table 4 – Crypto-Officer Services

Service	Description	Software Block Diagram Interface	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.	Provider Interface (Cryptomgr)	WX	N/A
Run self-tests	Restarting the appliance will force the self-tests to run when the module is loaded.	Provider Interface (Cryptomgr)	WX	N/A
Show Status	Uses the "/opt/unisys/fips status" command to return the current status of the module from the dmesg log file	Provider Interface (Cryptomgr)	R	N/A
Zeroize keys	Cycling the power zeroizes and de-allocates memory containing sensitive data.	N/A	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 5.

Table 5 – User Services

Service	Description	Software Block Diagram Interface	Type of Access to CSP	API Calls
Encryption/ Decryption	Encrypt or decrypt a block of data using a symmetric algorithm.	Consumer Interface	RWX	crypto_alloc_ablkcipher crypto_alloc_blkcipher crypto_free_ablkcipher crypto_free_blkcipher crypto_ablkcipher_setkey crypto_blkcipher_setkey crypto_ablkcipher_encrypt crypto_blkcipher_encrypt crypto_blkcipher_encrypt_iv crypto_ablkcipher_decrypt crypto_blkcipher_decrypt crypto_blkcipher_decrypt_iv crypto_blkcipher_set_iv ablkcipher_request_alloc ablkcipher_request_free ablkcipher_request_set_callback ablkcipher_request_set_crypt
Authenticated Encryption with Associated Data (AEAD)	A combined cryptographic protocol that only supports the approved algorithms used in the module. Keys for each approved algorithm are associated as required by Section 8.2.1, “Deterministic Construction” in NIST Special Publication 800-38D.	Consumer Interface	RWX	crypto_alloc_aead aead_givcrypt_alloc crypto_free_aead aead_givcrypt_free crypto_aead_setkey crypto_aead_encrypt crypto_aead_givencrypt aead_request_alloc aead_request_free aead_request_set_callback aead_givcrypt_set_callback aead_request_set_crypt aead_givcrypt_set_crypt crypto_aead_setauthsize aead_request_set_assoc aead_givcrypt_set_assoc

Service	Description	Software Block Diagram Interface	Type of Access to CSP	API Calls
Hashing	Perform a hashing operation on a block of data, using SHA-1, SHA-224, SHA-256, SHA-384, or SHA-512.	Consumer Interface	RWX	crypto_alloc_hash crypto_alloc_ahash crypto_alloc_shash crypto_free_hash crypto_free_ahash crypto_free_shash crypto_hash_init crypto_ahash_init crypto_shash_init crypto_hash_update crypto_ahash_update crypto_shash_update crypto_hash_final crypto_ahash_final crypto_shash_final crypto_ahash_finup crypto_shash_finup crypto_ahash_digest crypto_shash_digest ahash_request_alloc ahash_request_free ahash_request_set_callback ahash_request_set_crypt
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, or SHA-512.	Consumer Interface	RWX	crypto_alloc_hash crypto_alloc_ahash crypto_alloc_shash crypto_free_hash crypto_free_ahash crypto_free_shash crypto_hash_init crypto_ahash_init crypto_shash_init crypto_hash_update crypto_ahash_update crypto_shash_update crypto_hash_final crypto_ahash_final crypto_shash_final crypto_ahash_finup crypto_shash_finup crypto_ahash_digest crypto_shash_digest crypto_hash_setkey crypto_ahash_setkey crypto_shash_setkey ahash_request_alloc ahash_request_free ahash_request_set_callback ahash_request_set_crypt

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 (i.e., 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

The module implements the following FIPS-approved algorithms, as described in Table 6.

Table 6 – FIPS-approved Algorithm Implementations

Algorithm	Modes	Certificate Number
AES for x86_64	ECB, CBC, CTR, and GCM	#3513
AES-NI	ECB, CBC, CTR, and GCM	#3519
Generic implementation of SHA	SHA-1, SHA-224, SHA-256, SHA-384, SHA-512	#2901
SSSE3 implementation of SHA	SHA-1, SHA-224, SHA-256, SHA-384, SHA-512	#2900
HMAC on any of the above hashing functions	SHA-1, SHA-224, SHA-256, SHA-384, SHA-512	#2247 for generic implementation of SHA #2246 for SSSE3 implementation of SHA

7.1 Critical Security Parameters

The module supports the CSPs listed in Table 7.

Table 7 – 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; API call; Cycle host power	Encryption/ Decryption
AES-GCM IV	Deterministic Construction (in compliance with Section 8.2.1, "Deterministic Construction" in NIST Special Publication 800-38D)	64-bit IV: 32-bit invocation counter, 32-bit context counter concatenated	Never	The module does not store keys	Reboot operating system; API call; 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; API call; Cycle host power	Message Integrity/ Authentication with SHS

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, and SHA-256 block sizes is 64 bits, while the fixed key length for the SHA-384 and SHA-512 block sizes is 128 bits).*

7.2 Key Generation

The module does not generate keys. All keys are generated externally to the module.

7.3 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.

7.4 Key Storage

Keys are not persistently stored by the module.

7.5 Key Zeroization

The module does not persistently store keys (with the exception of the module integrity key and HMAC digests). 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 the following table, Table 8.

Table 8 – Electromagnetic Interference and Compatibility

Testing Platform	Model Number	EMI/EMC Notes
Intel Pentium Processor G3420	R220	FCC Class A
Intel Xeon Processor E5-2697 v3 (with AES-NI)	R630	FCC Class A
Intel Xeon Processor E5-4627 v2 (with AES-NI)	R820	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 Test using an HMAC SHA-512 on all of the module's components
- Known Answer Tests (KATs)
    - AES encrypt KAT;
    - AES decrypt KAT;
    - SHA (SHA-1, SHA-224, SHA-256, SHA-384, SHA-512) KAT; and
    - HMAC (SHA-1, SHA-224, SHA-256, SHA-384, SHA-512) KAT

Data output from the module's data output interface is inhibited while performing self-tests. All kernel object modules must pass power-up self-tests before the system is allowed to enter any user modes. 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.

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.

10. 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.

11. Secure Operation

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.

11.1 Crypto-Officer Guidance

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 Secure Virtual Gateway software appliance.

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.

For complete instructions on installing and configuring the Secure Virtual Gateway, see the *Unisys Stealth Secure Virtual Gateway Installation and User's Guide*. This can be found on the Unisys Product Support website (http://support.unisys.com/).

11.2 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. To confirm that each module component passed the self-tests, the operator must check the process flag in the /proc/sys/crypto/fips_enabled file. 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 power-up self-tests failed, and the system must be rebooted.

11.3 AES-GCM Key/IV Usage

The module implements a 32-bit context counter for the fixed field to construct IVs for AES-GCM. This counter exists entirely within the module's cryptographic boundary. The AES-GCM algorithm can only be accessed through the module's defined API, which controls the IV construction in compliance with Section 8.2.1, "Deterministic Construction" in NIST Special Publication 800-38D. This counter is persistent between power removal or reboots to ensure that the context counter does not repeat until all 2^{32} combinations are exhausted.

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^{32}-1 times, the encryption operation will fail and a new encryption context must be established.

12. Glossary and Abbreviations

- AES – Advanced Encryption Standard
- AES-NI – Advanced Encryption Standard New Instruction set
- API – Application Program Interface
- CBC – Cipher Block Chaining
- 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
- OS – Operating System
- PCLMULQDQ – Carry-less Multiplication Quadword
- SHA – Secure Hash Algorithm
- SHS – Secure Hash Standard
- SSSE3 – Supplemental Streaming SIMD Extensions 3