Unisys Linux strongSwan Cryptographic Module Version 5.6.3-6.4

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

April 21, 2021

Table of Contents

1. Introduction

1.1. Document History

Authors	*Date*	*Version*	*Comment*
Unisys Stealth Team	June 24, 2019	0.1	Initial draft.
Unisys Stealth Team	October 30, 2019	0.2	Revised to address comments from Leidos.
Unisys Stealth Team	May 12, 2020	0.3	Revised to address additional input from Leidos.
Unisys Stealth Team	April 21, 2021	0.4	Final document.

1.2. Purpose

This is the non-proprietary FIPS 140-2 Security Policy for the Unisys Linux strongSwan Cryptographic Module, which is referred to as *the Module*. The Module uses the Unisys Linux OpenSSL FIPS Object Module as a bound module, which is referred to as *the bound Module*, for cryptographic primitives and the power-on Integrity Test. The bound Module is a FIPS 140-2 validated module with certificate #3959.

This document describes how this 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.

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 provides a list of the security requirement sections and their associated levels.

Table 1 – Module Compliance Table

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

Security Component	Security Level
Mitigation of Other Attacks	N/A
Overall Level of Certification	1

2. Cryptographic Module Description

The module is a software-shared cryptographic library that provides key derivation functions for the Internet Key Exchange (IKE) protocol in the Ubuntu operating system user space. The module is defined as a multi-chip standalone module, as defined by FIPS PUB 140-2. This module implements a KDF for IKEv1 and IKEv2 that complies with the requirements of SP 800-135rev1.

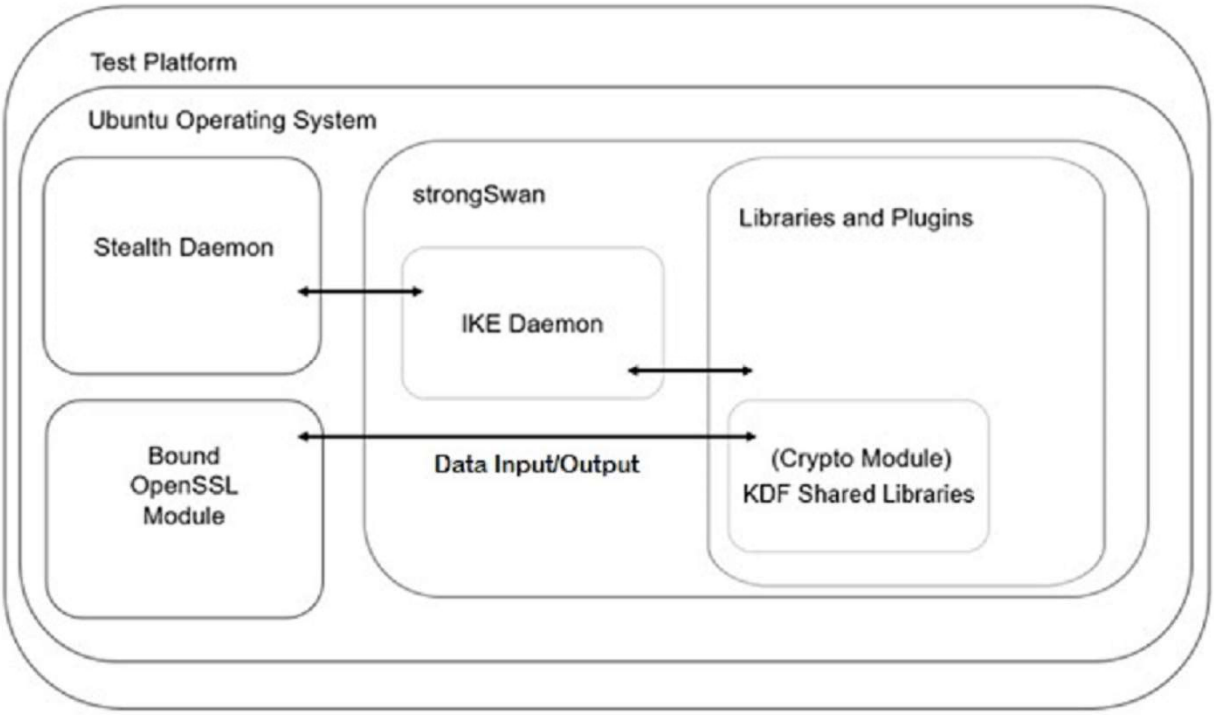
2.1. Cryptographic Boundary

The module is a collection of software-shared cryptographic library that provides key derivation functions for the Internet Key Exchange (IKE) protocol in the Ubuntu operating system user space; this set defines the Module's cryptographic boundary. More specifically, these shared libraries are libstrongswan-fips-kdfv1.so and libstrongswan-fips-kdfv2.so, represented by the "Crypto Module" block under "Libraries and Plugins" in the graphic below. The officially tested platform is Ubuntu Linux 18.04 LTS Server Edition running on a Dell PowerEdge R640 Server with an Intel Xeon Gold 5115.

The bound Module is represented by the "Bound OpenSSL Module" block and is used for cryptographic primitives and the power-on integrity test on the Module.

Figure 1 is the software block diagram of the Module, and it illustrates the Module boundary. Bidirectional arrows in the diagram indicate data input/output.

Figure 1 – Software Block Diagram



2.2. Cryptographic Algorithms

Table 2 provides a list of the CAVS certificates and information about their cryptographic implementation in FIPS mode.

Table 2 – Approved Algorithms Implemented by the Module

Algorithm	CAVS Certificate	Standard
SP 800-135rev1 IKEv1 KDF using HMAC with SHA-1, SHA-256, SHA-384 and SHA-512 (CVL)	C1012	[SP 800-135rev1]
SP 800-135rev1 IKEv2 KDF using HMAC with SHA-1, SHA-256, SHA-384 and SHA-512 (CVL)	C1012	[SP 800-135rev1]

Table 3 provides a list of the CAVS certificates for algorithms provided from the bound Module. These algorithms are used for primitives in support of the SP 800-135rev1 KDFs and for supporting the power-on integrity tests. Note that the bound Module supports additional algorithms not listed in Table 3. The Module only utilizes the algorithms which are listed in this document.

Table 3 – Approved Algorithms Implemented by the bound Module

Algorithm	CAVS Certificate	Standard
SHA-1, SHA-256, SHA-384 and SHA-512	C1011	[FIPS 180-4]
RSA Sig Ver PKCS1.5 Mod Size 4096 with SHA-256	C1011	[FIPS 186-4]

Algorithm	CAVS Certificate	Standard
HMAC - using SHA-1 and SHA-2 algorithms	C1011	[FIPS 198-1]

3. Module Ports and Interfaces

As a software-only module, the Module does not have physical ports.

Table 4 describes the relationship between the logical and physical interfaces.

Table 4 – Mapping Logical Interfaces

FIPS 140-2 Interface	Logical Interface	Physical Interface
Data input interface	KDF Library API	Not Applicable
Data output interface	KDF Library API	Not Applicable
Control input interface	KDF Library API	Not Applicable
Status output interface	System Logs and API return codes	Not Applicable
Power interface	Not Applicable	Not Applicable

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 support a maintenance role.

4.1. Identification and Authentication

The module does not support authentication mechanisms.

4.2. Roles and Services

The module supports the services listed in the following table. The table groups the authorized services by the operator roles and identifies the Cryptographic Keys and CSPs associated with the services. The modes of access are also identified per the explanation, as follows:

- R – Read: The item is read or referenced by the service.
- W – Write: The item is written or updated by the service.
- E – Execute: The item is executed by the service. (The item is used as part of a cryptographic function.)

Table 5 shows the services available to each role and the keys and CSPs associated with each role.

Table 5 – Mapping of Cryptographic Keys and CSPs to Services

Service	Role	Cryptographic Key/CSP	Type of Access (R, W, E)
Key Derivation (IKEv1)	User	• Diffie-Hellman or EC Diffie-Hellman shared secret	R
		• SKEYID • SKEYID_e • SKEYID_a • SKEYID_d	W
Key Derivation (IKEv2)	User	• Diffie-Hellman or EC Diffie-Hellman shared secret	R
		• SKEYSEED • SK_d • SK_ei, SK_er • SK_ai, SK_ar • SK_pi, SK_pr	W
Run Self-Tests	Crypto-Officer	N/A	N/A
Show Status	Crypto-Officer	N/A	N/A

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. Cryptographic Keys and Critical Security Parameters

The module supports the Critical Security Parameters (CSP) listed in Table 6, Table 7, and Table 8. The supported key sizes for each CSP are dependent on the bound Module's supported hash functions. These supported algorithms will produce keys of one of the given sizes in bits: 160, 224, 256, 384, and 512.

Table 6 – IKEv1 Cryptographic Module Keys and CSPs

Key	Generation	STORAGE	Use
Diffie-Hellman or EC Diffie-Hellman shared secret	N/A (generated by the bound Module and entered into Module in plaintext)	Stored in RAM	Used as part of calculation of *SKEYID*
Key Derivation Key (SKEYID)	Key derivation from shared secret using SP 800-135rev1 IKEv1 KDF	Stored in RAM	Used for deriving below keying material
Encryption Key (SKEYID_e)	Key derivation from shared secret using SP 800-135rev1 IKEv1 KDF	Stored in RAM	Keying material used by ISAKMP SA to encrypt messages
Authentication Key (SKEYID_a)	Key derivation from shared secret using SP 800-135rev1 IKEv1 KDF	Stored in RAM	Keying material used by ISAKMP SA to authenticate messages
Key for Further Derivation (SKEYID_d)	Key derivation from shared secret using SP 800-135rev1 IKEv1 KDF	Stored in RAM	Keying material used to derive keys for non-ISAKMPSAs

Table 7 – IKEv2 Cryptographic Module Keys and CSPs

Key	Generation	STORAGE	Use
Diffie-Hellman or EC Diffie-Hellman shared secret	N/A (generated by the bound Module and entered into Module in plaintext)	Stored in RAM	Used as part of calculation of *SKEYSEED*
Key Derivation Key (SKEYSEED)	Key derivation from shared secret using SP 800-135rev1 IKEv2 KDF	Stored in RAM	Used for deriving below keying material
Key for Further Derivation (SK_d)	Key derivation from shared secret using SP 800-135rev1 IKEv2 KDF	Stored in RAM	Used for deriving new keys for the CHILD_SAs established from an IKE_SA
Encryption Keys (SK_ei, SK_er)	Key derivation from shared secret using SP 800-135rev1 IKEv2 KDF	Stored in RAM	Used for encrypting/decrypting exchanges of an IKE_SA
Authentication Keys (SK_ai, SK_ar)	Key derivation from shared secret using SP 800-135rev1 IKEv2 KDF	Stored in RAM	Used as keys to the integrity protection algorithm
Authentication Payload Keys (SK_pi, SK_pr)	Key derivation from shared secret using SP 800-135rev1 IKEv2 KDF	Stored in RAM	Used when generating an AUTH payload

Table 8 – Power On Integrity Test Key

Key	Generation	STORAGE	Use
RSA 4096-bit Public Key	Hardcoded	Stored on file system outside cryptographic boundary	Used to verify the signature files that are used to verify module integrity

7.2. Key Destruction/Zeroization

Keys are allocated space in RAM using operating system calls. This memory is zeroized when the module is unloaded from memory.

7.3. Key Entry and Output

The module does not support manual key entry. Keys are entered into the module electronically via the module API, and are output electronically via the module API. The shared secret is input electronically via the module API.

7.4. Key Generation

The module does not implement any key generation.

7.5. Key Derivation

The module implements SP 800-135rev1 KDF for the IKEv1 and IKEv2 protocol. No parts of these protocols, other than the KDFs, have been tested by the CAVP or CMVP.

8. Self-tests

The module performs power-up self-tests.

8.1. Power-up Self-tests

The module automatically performs power-up self-tests when loaded into memory. These tests examine the Module for corruption.

While the Module performs the power-up tests, input and output are inhibited and the services provided by the Module are not available. If any of these tests fail, the Module will return the error message listed in Section 8.2, "Handling Self-test Errors," enter the error state, and then terminate.

8.1.1. Cryptographic Algorithm KATs

Known Answer Tests (KATs) are not mandatory, and are not implemented for the module. All algorithms from the bound Module have KATs that are run when the bound Module is loaded.

8.1.2. Software Integrity Tests

The module verifies the integrity of its components using digital signatures. A signature is produced at build time for each shared library using a 4096-bit RSA private key and SHA-256 for the hash function. Each signature is compared with a signature computed at load-time, by the bound Module, for each shared library with the 4096-bit RSA public key provided by the Debian package, and SHA-256 as the hash function. The name of the two signature files are libstrongswan-fips-kdfv1.sig and libstrongswan-fips-kdfv2.sig.

8.2. Handling Self-test Errors

On self-test failure, an error message will be reported to indicate the error and then enter the error state. Table 9 lists the possible self-test errors:

Table 9 – Self-test Errors

Error	Description
Public Key File Failure	The public key file could not be opened, or it could not be read.
Signature File Failure	The signature file could not be opened, or it could not be read.
Signature Validation Failure	The signature read from the signature file does not match the signature computed at run time.
Test Vector Failure	The calculated result of a KAT does not match the known answer.

9. Crypto-Officer and User Guidance

9.1. Secure Setup and Initialization

To be used in a FIPS-compliant manner, the proper installation of the Unisys Linux OpenSSL FIPS Object Module software must be performed. After the previously listed module is installed and configured properly, the Crypto-Officer should check the existence of /proc/sys/crypto/fips_enabled. If this file does not contain the character “1”, the operating environment is not correctly configured to support FIPS, and the Module will not operate properly as a FIPS-validated module. The Module can be installed by using the Debian packaging tool, *dpkg*, or by using a Unisys Stealth installation script.

9.2. Module Security Policy Rules

When operating in a FIPS Approved Mode, the user or administrator shall not use any debugging or tracing software to inspect the Module.

10. Mitigation of Other Attacks

The module does not mitigate against any specific attacks.