ubuntu 20.04 Strongswan Cryptographic Module

version 3.0

FIPS 140-2 Non-Proprietary Security Policy

Document Version 1.2

Last update: September 29, 2021

Prepared by:

atsec information security corporation

9130 Jollyville Road, Suite 260

Austin, TX 78759

www.atsec.com

Table of Contents

Copyrights and Trademarks

Ubuntu and Canonical are registered trademarks of Canonical Ltd.

Linux is a registered trademark of Linus Torvalds.

1. Cryptographic Module Specification

This document is the non-proprietary FIPS 140-2 (Federal Information Processing Standards Publication 140-2) Security Policy for version 3.0 of the Ubuntu 20.04 Strongswan Cryptographic Module. It contains the security rules under which the module must operate and describes how this module meets the requirements as specified in FIPS 140-2 for a Security Level 1 module.

The following sections describe the cryptographic module and how it conforms to the FIPS 140-2 specification in each of the required areas.

1.1. Module Overview

The Ubuntu 20.04 Strongswan Cryptographic Module (also referred to as “the module”) provides cryptographic services for the Internet Key Exchange (IKE) protocol in the Ubuntu Operating System user space.

The module uses the Ubuntu 20.04 OpenSSL Cryptographic Module as a bound module (also referred to as “the bound OpenSSL module”), which provides the underlying cryptographic algorithms necessary for establishing and maintaining IKE sessions. The Ubuntu 20.04 OpenSSL Cryptographic Module is a FIPS-validated module with certificate #3966.

The module also uses the Ubuntu 20.04 Kernel Crypto API Cryptographic Module as a bound module (also referred to as “the bound Kernel Crypto API module”) for performing integrity tests. The Ubuntu 20.04 Kernel Crypto API Cryptographic Module is a FIPS-validated module with certificate #3928.

For the purpose of the FIPS 140-2 validation, the module is a software-only, multi-chip standalone cryptographic module validated at overall Security Level 1. The table below shows the security level claimed for each of the eleven sections that comprise the FIPS 140-2 standard:

FIPS 140-2 Section		Security Level
1	Cryptographic Module Specification	1
2	Cryptographic Module Ports and Interfaces	1
3	Roles, Services and Authentication	1
4	Finite State Model	1
5	Physical Security	N/A
6	Operational Environment	1
7	Cryptographic Key Management	1
8	EMI/EMC	1
9	Self-Tests	1
10	Design Assurance	1
11	Mitigation of Other Attacks	N/A
Overall Level		1

Table 1 - Security Levels

The cryptographic logical boundary consists of the following components, including their location in the target platform.

Description	Components
Module programs	/usr/sbin/ipsec /usr/lib/ipsec/stroke /usr/lib/ipsec/starter /usr/lib/ipsec/charon /usr/lib/ipsec/pool /usr/lib/ipsec/_updown
Self-test programs	/usr/lib/ipsec/_fipscheck /usr/lib/ipsec/ikev2-kdf-selftest
Integrity HMAC files for module binaries	/usr/sbin/.ipsec.hmac /usr/lib/ipsec/.stroke.hmac /usr/lib/ipsec/.starter.hmac /usr/lib/ipsec/.charon.hmac /usr/lib/ipsec/._fipscheck.hmac /usr/lib/ipsec/.ikev2-kdf-selftest.hmac /usr/lib/ipsec/._updown.hmac /usr/lib/ipsec/.pool.hmac
Standard base libs and openssl plugin	/usr/lib/ipsec/libstrongswan.so.0.0.0 /usr/lib/ipsec/plugins/libstrongswan-openssl.so /usr/lib/ipsec/libcharon.so.0.0.0
Integrity HMAC files for standard base libs and openssl plugin	/usr/lib/ipsec/.libstrongswan.so.0.0.0.hmac /usr/lib/ipsec/plugins/.libstrongswan-openssl.so.hmac /usr/lib/ipsec/.libcharon.so.0.0.0.hmac
Default plugins	/usr/lib/ipsec/plugins/libstrongswan-fips-prf.so /usr/lib/ipsec/plugins/libstrongswan-nonce.so /usr/lib/ipsec/plugins/libstrongswan-dnskey.so /usr/lib/ipsec/plugins/libstrongswan-pem.so /usr/lib/ipsec/plugins/libstrongswan-pgp.so /usr/lib/ipsec/plugins/libstrongswan-pkcs1.so /usr/lib/ipsec/plugins/libstrongswan-pkcs7.so /usr/lib/ipsec/plugins/libstrongswan-pkcs8.so /usr/lib/ipsec/plugins/libstrongswan-pkcs12.so /usr/lib/ipsec/plugins/libstrongswan-pubkey.so /usr/lib/ipsec/plugins/libstrongswan-sshkey.so /usr/lib/ipsec/plugins/libstrongswan-x509.so /usr/lib/ipsec/plugins/libstrongswan-constraints.so

Description	Components
	/usr/lib/ipsec/plugins/libstrongswan-revocation.so /usr/lib/ipsec/plugins/libstrongswan-kernel-netlink.so /usr/lib/ipsec/plugins/libstrongswan-socket-default.so /usr/lib/ipsec/plugins/libstrongswan-stroke.so /usr/lib/ipsec/plugins/libstrongswan-attr.so /usr/lib/ipsec/plugins/libstrongswan-resolve.so /usr/lib/ipsec/plugins/libstrongswan-updown.so
Integrity HMAC files for default plugins	/usr/lib/ipsec/plugins/.libstrongswan-fips-prf.so.hmac /usr/lib/ipsec/plugins/.libstrongswan-nonce.so.hmac /usr/lib/ipsec/plugins/.libstrongswan-dnskey.so.hmac /usr/lib/ipsec/plugins/.libstrongswan-pem.so.hmac /usr/lib/ipsec/plugins/.libstrongswan-pgp.so.hmac /usr/lib/ipsec/plugins/.libstrongswan-pkcs1.so.hmac /usr/lib/ipsec/plugins/.libstrongswan-pkcs7.so.hmac /usr/lib/ipsec/plugins/.libstrongswan-pkcs8.so.hmac /usr/lib/ipsec/plugins/.libstrongswan-pkcs12.so.hmac /usr/lib/ipsec/plugins/.libstrongswan-pubkey.so.hmac /usr/lib/ipsec/plugins/.libstrongswan-sshkey.so.hmac /usr/lib/ipsec/plugins/.libstrongswan-x509.so.hmac /usr/lib/ipsec/plugins/.libstrongswan-constraints.so.hmac /usr/lib/ipsec/plugins/.libstrongswan-revocation.so.hmac /usr/lib/ipsec/plugins/.libstrongswan-kernel-netlink.so.hmac /usr/lib/ipsec/plugins/.libstrongswan-socket-default.so.hmac /usr/lib/ipsec/plugins/.libstrongswan-stroke.so.hmac /usr/lib/ipsec/plugins/.libstrongswan-attr.so.hmac /usr/lib/ipsec/plugins/.libstrongswan-resolve.so.hmac /usr/lib/ipsec/plugins/.libstrongswan-updown.so.hmac

Table 2- Cryptographic Module Components

The software block diagram below shows the module with the delimitation of its logical boundary (depicted in the blue box), the module interfaces with the bound OpenSSL and Kernel Crypto API modules, and the operational environment. Data flow is represented by solid lines, while control flow is represented by dotted lines.

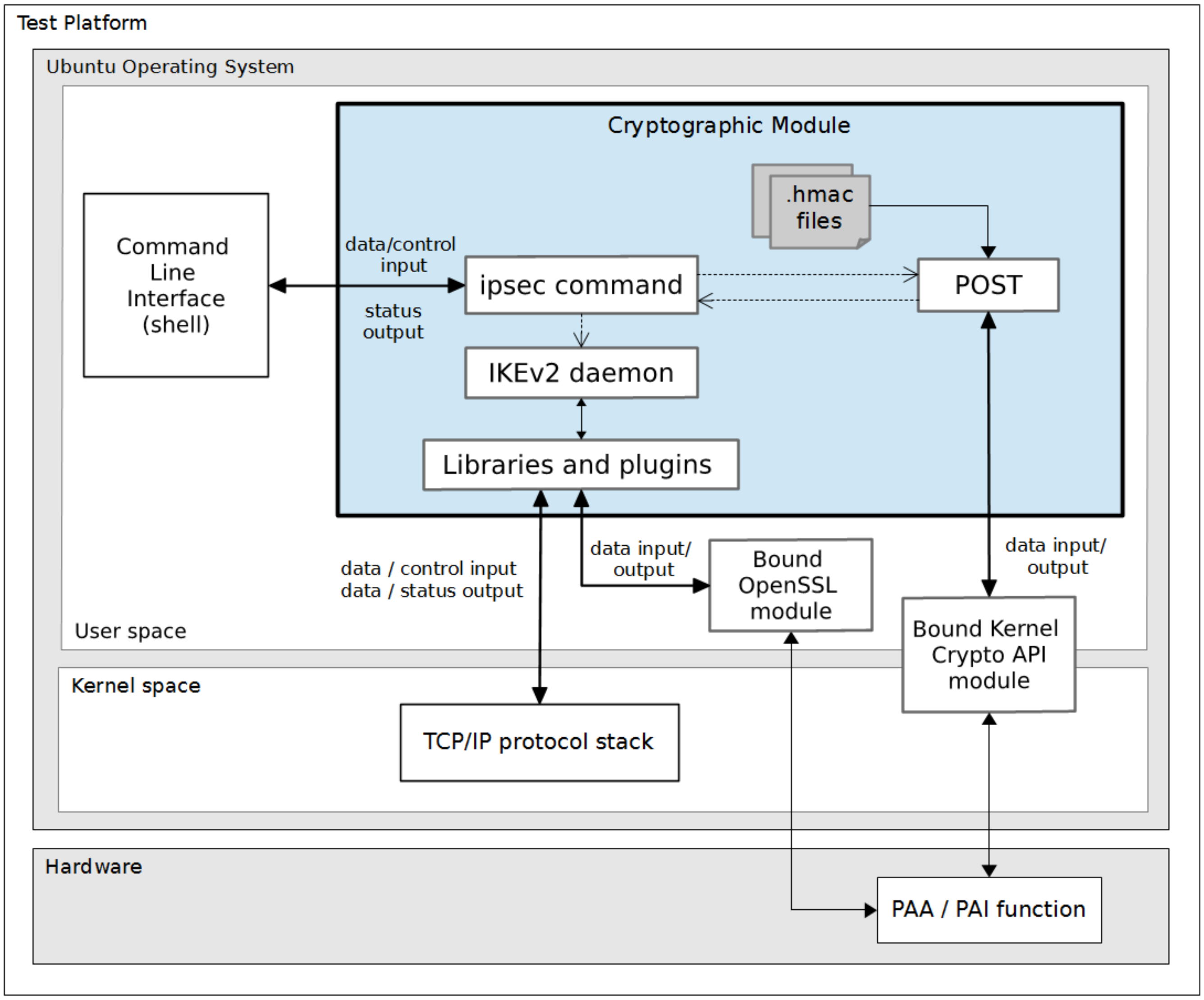

Figure 1 - Software Block Diagram

The module is aimed to run on a general-purpose computer (GPC); the physical boundary is the surface of the case of the tested platforms, as shown in the diagram below:

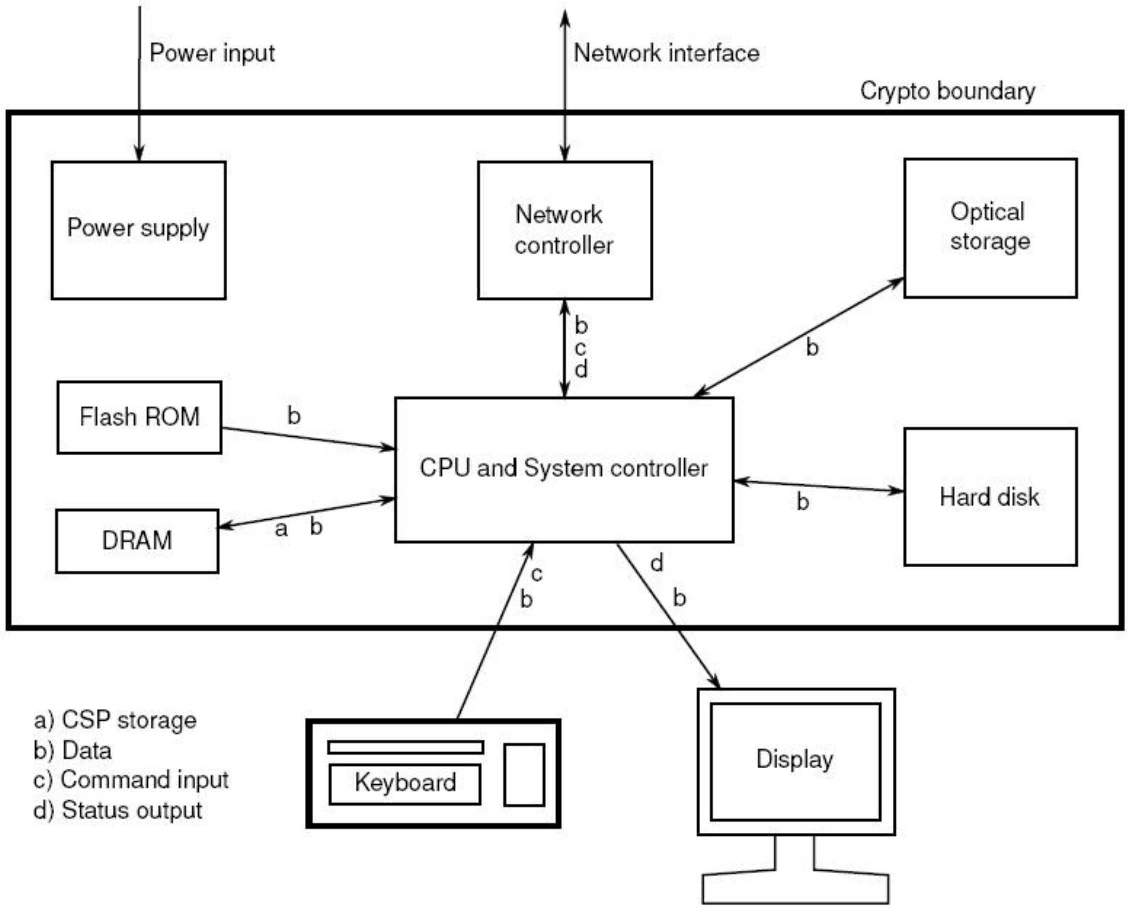

Figure 2 - Cryptographic Module Physical Boundary

The module has been tested on the platforms shown below:

Test Platform	Processor	Test Configuration
Supermicro SYS-1019P-WTR	Intel® Xeon® Gold 6226	Ubuntu 20.04 LTS 64-bit with/without AES-NI (PAA)

Table 3 - Tested Platforms

1.2. Modes of Operation

The module supports two modes of operation:

- **FIPS mode** (the Approved mode of operation): only approved or allowed security functions with sufficient security strength can be used.
- **non-FIPS mode** (the non-Approved mode of operation): only non-approved security functions can be used.

The module enters FIPS mode after power-up tests succeed. Once the module is operational, the mode of operation is implicitly assumed depending on the security function invoked and the security strength of the cryptographic keys.

Critical security parameters used or stored in FIPS mode are not used in non-FIPS mode and vice versa.

2. Cryptographic Module Ports and Interfaces

As a software-only module, the module does not have physical ports. For the purpose of the FIPS 140-2 validation, the physical ports are interpreted to be the physical ports of the hardware platform on which it runs.

The following table summarizes the four logical interfaces provided by the module:

FIPS Interface	Physical Port	Logical Interface
Data Input	Keyboard, Ethernet port	/etc/ipsec.secrets file, private key file, certificate files under the /etc/ipsec.d directory, input data received from the network (IKEv2 protocol), input data received from the bound OpenSSL module via its API parameters.
Data Output	Display, Ethernet port	Output data sent through the network (IKEv2 protocol), output data sent to the bound OpenSSL module via its API parameters.
Control Input	Keyboard, Ethernet port	Invocation of the ipsec command on the command line, control parameters via the ipsec command and the /etc/ipsec.conf file, IKEv2 protocol message requests received from the network.
Status Output	Display, Ethernet port	Status messages returned after execution of the ipsec command, status of processing IKEv2 protocol message requests sent through the network.
Power Input	PC Power Supply Port	N/A

Table 4 - Ports and Interfaces

3. Roles, Services and Authentication

3.1. Roles

The module supports the following roles:

- **User role**: performs services of establish, maintain and close IKEv2 sessions.
- **Crypto Officer role**: performs services of module installation and configuration, manage the IKEv2 daemon (start, stop, etc), show status and self-tests.

The User and Crypto Officer roles are implicitly assumed by the entity accessing the module services.

3.2. Services

The module provides services to users that assume one of the available roles. All services are shown in Table 5, and described in detail in the user documentation.

Table 5 shows the Approved services in FIPS mode, the cryptographic algorithms supported for the service, the roles to perform the service, the cryptographic keys or Critical Security Parameters (CSPs) involved and how they are accessed. The following convention is used to specify access rights to a CSP:

- **Create**: the calling application can create a new CSP.
- **Read**: the calling application can read the CSP.
- **Update**: the calling application can write a new value to the CSP.
- **Zeroize**: the calling application can zeroize the CSP.
- **n/a**: the calling application does not access any CSP or key during its operation.

See also Appendix A of this document for the complete list of supported cipher suites by the module in FIPS mode.

Note: Only the NIST SP800-135 IKEv2 Key Derivation Function (KDF) algorithm is provided by the Ubuntu 20.04 Strongswan Cryptographic Module. The HMAC with SHA-256 algorithm used for integrity test is provided by the bound Kernel Crypto API module, while the rest of the cryptographic algorithms listed in Table 8 are provided by the bound OpenSSL module.

Service	Algorithms	Role	Access	Key / CSP
Start IKEv2 daemon	N/A	Crypto Officer	N/A	None
Configure IKEv2 daemon	N/A	Crypto Officer	Read	Pre-shared Key or Post-Quantum Pre-shared Key

Service	Algorithms	Role	Access	Key / CSP
IKE_SA_INIT Exchange	Key exchange: • Diffie-Hellman with key size between 2048 and 8192 bits[1]; • EC Diffie-Hellman with NIST curves P-224, P-256, P-384 and P-521.	User	Create, Read	Diffie-Hellman or EC Diffie-Hellman public and private keys Diffie-Hellman or EC Diffie-Hellman shared secret
	Key derivation: • SP800-135 IKEv2 KDF using HMAC with SHA-1, SHA-256, SHA-384 and SHA-512	User	Create	Derivation key (SK_d) Encryption keys (SK_ei, SK_er) Authentication keys (SK_ai, SK_ar) Authentication payload keys (SK_pi, SK_pr) Diffie-Hellman or EC Diffie-Hellman shared secret
IKE_AUTH Exchange	Signature generation and verification: • ECDSA with SHA-256, SHA-384, SHA-512 and curves P-256, P-384, P-521; • RSA with SHA-224, SHA-256, SHA-384, SHA-512 and key size between 2048 and 16384 bits	User	Read	RSA or ECDSA private/public key Peer's RSA or ECDSA public key
	Data encryption and decryption: • AES (CBC, GCM), Triple-DES (CBC)	User	Read	Encryption keys (SK_ei, SK_er)
	Data integrity (MAC): • HMAC with SHA-1, SHA-256, SHA-384, SHA-512	User	Read	Authentication keys (SK_ai, SK_ar)
CREATE_CHILD_SA Exchange	Data encryption and decryption: • AES (CBC, GCM), Triple-DES	User	Read	Encryption keys (SK_ei, SK_er)

[1] Diffie-Hellman key agreement provides between 112 and 200 bits of encryption strength.

Service	Algorithms	Role	Access	Key / CSP
	(CBC)			
	Data integrity (MAC): • HMAC with SHA-1, SHA-256, SHA-384, SHA-512	User	Read	Authentication keys (SK_ai, SK_ar)
	Key exchange: • Diffie-Hellman with key size between 2048 and 8192 bits[2]; • EC Diffie-Hellman with NIST curves P-224, P-256, P-384 and P-521.	User	Create	Diffie-Hellman or EC Diffie-Hellman public and private keys Diffie-Hellman or EC Diffie-Hellman shared secret
	Key derivation: • SP800-135 IKEv2 KDF using HMAC with SHA-1, SHA-256, SHA-384 and SHA-512	User	Read	Derivation key (SK_d)
		User	Create	New derivation key (SK_d) New encryption keys (SK_ei, SK_er) New authentication keys (SK_ai, SK_ar) New authentication payload keys (SK_pi, SK_pr)
INFORMATIONAL Exchange	Data encryption and decryption: • AES (CBC, GCM), Triple-DES (CBC)	User	Read	Encryption key (SK_ei, SK_er)
	Data integrity (MAC): • HMAC with SHA-1, SHA-256, SHA-384, SHA-512	User	Read	Authentication key (SK_ai, SK_ar)
Close Security Association	N/A	User	Zeroize	All aforementioned CSPs
Terminate IKEv2 daemon	N/A	Crypto Officer	Zeroize	All aforementioned CSPs
Show status	N/A	Crypto Officer	N/A	None

[2] Diffie-Hellman key agreement provides between 112 and 200 bits of encryption strength.

Service	Algorithms	Role	Access	Key / CSP
Self-test	Integrity tests: • HMAC with SHA-256 Known Answer Tests: • SP800-135 IKEv2 KDF	Crypto Officer	N/A	None
Module installation	N/A	Crypto Officer	N/A	None

Table 5 - Services in FIPS mode

Table 6 lists the service that uses the non-Approved algorithms or non-compliant key sizes, which cause the module to transition to the non-FIPS mode implicitly.

Service	Algorithms / Key sizes	Role
IKE_SA_INIT Exchange	Key exchange: • EC Diffie-Hellman with non-NIST curves	User
IKE_AUTH Exchange	Signature generation: • ECDSA with SHA-1 • RSA with SHA-1 or key size less than 2048 bits Signature generation or verification: • ECDSA with non-NIST curves Authenticated Data Encryption and Decryption • ChaCha20 and Poly1305	User
CREATE_CHILD_SA Exchange	Key exchange: • EC Diffie-Hellman with non-NIST curves Authenticated Data Encryption and Decryption • ChaCha20 and Poly1305	User
INFORMATIONAL Exchange	Authenticated Data Encryption and Decryption • ChaCha20 and Poly1305	User

Table 6 - Services in non-FIPS mode

3.3. Algorithms

The following table shows the CAVP certificates and their associated information of the cryptographic implementation in FIPS mode.

Notice that for the Internet Key Exchange version 2 protocol (IKEv2), no parts of this protocol, other than the key derivation function (KDF), have been tested by the CAVP.

Algorithm	CAVP Cert.	Standard	Use
SP800-135 IKEv2 KDF using HMAC with SHA-1, SHA-256, SHA-384, SHA-512	CVL #A660	[SP800-135]	Key derivation in the IKEv2 protocol.

Table 7 – Approved Algorithms implemented by the module

The table below shows the Approved and non-Approved, but Allowed algorithms that are used by the module, but provided by the bound Ubuntu 20.04 OpenSSL Cryptographic Module in FIPS mode. The CAVP certificates are listed if applicable. Please refer to Table 8 for detailed information about the modes, methods, or key sizes of the algorithms used by the module.

The Strongswan and the bound OpenSSL module together provide the Diffie Hellman and EC Diffie Hellman key agreement algorithms. The Strongswan module only implements the KDF portion of the key agreement as stated in the above table and the bound OpenSSL module provides the shared secret computation as stated in Table 8.

The HMAC algorithm used for SP800-135 IKEv2 key derivation is also provided by the OpenSSL bound module.

Algorithm	Modes / Key sizes	CAVP Cert.	Use
Approved Algorithms			
AES	CBC mode with 128, 192 and 256 bit keys	#A522, #A523, #A524	Data encryption and decryption
	GCM mode with 128, 192 and 256 bit keys	#A526, #A527, #A528, #A533, #A534, #A535, #A536, #A537, #A538	Data encryption and decryption
DRBG	CTR_DRBG	#A522, #A523, #A524	Random number generation
ECDSA	P-224, P-256, P-384, P-521	#A529, #A530, #A531, #A532	Key generation
	SHA-256, SHA-384, SHA-512 with curves P-256, P-384, P-521	#A529, #A530, #A531, #A532	Signature generation and verification
HMAC	HMAC-SHA-1, HMAC-SHA-256, HMAC-SHA-384, HMAC-SHA-512	#A529, #A530, #A531, #A532	Message Authentication Code
RSA	SHA-224, SHA-256, SHA-384, SHA-512 with key size between 2048 and 16384 bits	#A529, #A530, #A531, #A532	Signature generation and verification
SHS	SHA-1, SHA-256, SHA-384, SHA-512	#A529, #A530, #A531, #A532	Message digest

Algorithm	Modes / Key sizes	CAVP Cert.	Use
Triple-DES	CBC mode with 192-bit key	#A525	Data encryption and decryption
KAS ECC-SSC	ECC Ephemeral Unified scheme P-224, P-256, P-384, P-521 K-233, K-283, K-409, K-571, B-233, B-283, B-409, B-571	#A529, #A530, #A531, #A532	EC Diffie-Hellman shared secret computation
KAS FFC-SSC	FCC dhEphem scheme; MODP-2048, MODP-3072, MODP-4096, MODP-6144, MODP-8192	#A539	Diffie-Hellman shared secret computation
Safe prime key generation and verification	MODP-2048, MODP-3072, MODP-4096, MODP-6144, MODP-8192	#A539	Key generation
Allowed Algorithms			
NDRNG		non-approved but allowed	Entropy source for the DRBG

Table 8 – Approved and Allowed Algorithms provided by the bound OpenSSL module

The table below shows the Approved algorithms that are used by the module, but provided by the bound Ubuntu 20.04 Kernel Crypto API Cryptographic Module in FIPS mode, specifically for supporting integrity tests. CAVP certificates are also provided.

Algorithm	Modes / Key sizes	CAVP Cert.
Approved Algorithms		
HMAC	HMAC-SHA-256	#A616, #A617, #A618, #A644, #A645, #A646
SHS	SHA-256	#A616, #A617, #A618, #A644, #A645, #A646
Allowed Algorithms		
N/A		

Table 9 – Approved and Allowed Algorithms provided by the bound Kernel Crypto API module

Table 10 shows the non-Approved algorithm implemented in the bound module, which is only available in non-FIPS mode.

Algorithm	Use
ECDSA signature generation with SHA-1 or	Signature generation

Algorithm	Use
RSA signature generation with SHA-1 or key size less than 2048 bits	Signature generation
ECDSA with non-NIST curves	Digital Signature Generation or verification
ECDH with non-NIST curves	Key exchange
ChaCha20 and Poly1305	Authenticated Data Encryption and Decryption

Table 10 - Non-Approved Algorithms provided by the bound OpenSSL module

The Strongswan and the bound OpenSSL module together provide the Diffie Hellman and EC Diffie Hellman key agreement. The Strongswan module only implements the IKEv2KDF part of the key agreement using the HMAC portion of the key agreement as stated in the above table and the bound OpenSSL module provides the shared secret computation:

- KAS (KAS-SSC Cert. #A539, CVL Cert. #A660); Diffie-Hellman provides between 112 and 200 bits of encryption strength.
- KAS (KAS-SSC Certs. #A529, #A530, #A531 and #A532, CVL Cert. #A660); EC Diffie-Hellman provides between 112 and 256 bits of encryption strength.

3.4. Operator Authentication

The module does not implement operator authentication. The role of the user is implicitly assumed based on the service requested.

4. Physical Security

The module is comprised of software only and therefore, this security policy does not make any claims on physical security.

5. Operational Environment

5.1. Applicability

The module operates in a modifiable operational environment per FIPS 140-2 Security Level 1 specifications. The module runs on a commercially available general-purpose operating system executing on the hardware specified in Table 3.

5.2. Policy

The operating system is restricted to a single operator (concurrent operators are explicitly excluded). The application that requests cryptographic services is the single user of the module.

6. Cryptographic Key Management

The following table summarizes the cryptographic keys and CSPs that are used by the cryptographic services implemented in the module:

Name	Generation	Entry and Output	Zeroization
Private keys (ECDSA, RSA)	N/A	Entry: read from the private key files. Output: to the bound OpenSSL module via API parameters.	Immediately after use.
Public keys (ECDSA, RSA)	N/A	Entry: read from the host key files. Output: to the network peer via IKE_AUTH exchange message.	
Peer's public keys (ECDSA, RSA)	N/A	Entry: from the network peer via IKE_AUTH exchange message. Output: to the bound OpenSSL module via API parameters.	
Diffie-Hellman or EC Diffie-Hellman public key	N/A (generated by the bound OpenSSL module)	Entry: from the bound OpenSSL module via API parameters. Output: to the network peer via IKE_SA_INIT or CREATE_CHILD_SA exchange messages.	Close of the IKEv2 Security Association (SA) Rekeying of the SA. Termination of the IKEv2 daemon.
Diffie-Hellman or EC Diffie-Hellman private key	N/A (generated by the bound OpenSSL module)	Entry: from the bound OpenSSL module via API parameters. Output: to the bound OpenSSL module via API parameters.	
Peer's Diffie-Hellman public or EC Diffie-Hellman public key	N/A	Entry: from the network peer IKE_SA_INIT or CREATE_CHILD_SA exchange messages. Output: to the bound OpenSSL module via API parameters.	
Diffie-Hellman or EC Diffie-Hellman shared secret	N/A (generated by the bound OpenSSL module)	Entry: from the bound OpenSSL module via API parameters. Output: N/A	
Derivation key (SK_d) from IKE_SA	Key derivation from shared secret using SP800-135 IKEv2 KDF	Entry: N/A Output: N/A	Close of the IKEv2 Security Association (SA) Rekeying of the SA. Termination of the IKEv2 daemon.
Encryption keys from IKE_SA (SK_ei, SK_er) (AES, Triple-DES)	Key derivation from shared secret using SP800-135 IKEv2 KDF	Entry: N/A Output: to the bound OpenSSL module via API parameters.	
Authentication keys from IKE_SA (SK_ai,	Key derivation from shared secret using	Entry: N/A	

Name	Generation	Entry and Output	Zeroization
SK_ar) (HMAC)	SP800-135 IKEv2 KDF	Output: to the bound OpenSSL module via API parameters.	
Authentication payload keys from IKE_SA (SK_pi, SK_pr) (HMAC)	Key derivation from shared secret using SP800-135 IKEv2 KDF	Entry: N/A Output: to the bound OpenSSL module via API parameters.	
New Derivation Key from CHILD_SA (SK_d) (HMAC)	Key derivation from shared secret using SP800-135 IKEv2 KDF	Entry: N/A Output: N/A	
New Encryption keys from CHILD_SA (SK_ei, SK_er) (HMAC)	Key derivation from shared secret using SP800-135 IKEv2 KDF	Entry: N/A Output: to the calling application via API parameters.	
New Authentication keys from CHILD_SA (SK_ai, SK_ar) (HMAC)	Key derivation from shared secret using SP800-135 IKEv2 KDF	Entry: N/A Output: to the calling application via API parameters.	
New Authentication payload keys from CHILD_SA (SK_pi, SK_pr) (HMAC)	Key derivation from shared secret using SP800-135 IKEv2 KDF	Entry: N/A Output: to the calling application via API parameters.	
Pre-shared Key or Post-Quantum Pre-shared Key	N/A	Entry: read from the private key files. Output: N/A	Zeroized on Power off.

Table 11 - Life cycle of Keys/CSPs

The following sections describe how CSPs, in particular cryptographic keys, are managed during their life cycle.

6.1. Random Number Generation

The module does not implement any random number generator. Instead, it uses the Random Number Generation (RNG) service provided by the bound Ubuntu 20.04 OpenSSL Cryptographic Module, which implements a Deterministic Random Bit Generator (DRBG) based on [SP800-90A].

6.2. Key Generation

The module does not implement key generation.

6.3. Key Agreement / Key Derivation

The module implements SP800-135 KDF for the IKEv2 protocol.

6.4. Key Entry / Output

The module does not support manual key entry or intermediate key generation key output. The keys are entered from or outputted to the module electronically.

6.5. Key / CSP Storage

The module does not perform persistent storage of keys. The keys and CSPs are temporarily stored as plaintext in the RAM.

Public and private keys for IKEv2 authentication are stored in the /etc/ipsec.d/certs and /etc/ipsec.d/private directories, which are within the module's physical boundary, but outside its logical boundary.

6.6. Key / CSP Zeroization

The memory occupied by keys and CSPs is allocated by regular memory allocation operating system calls. The module calls appropriate key zeroization functions provided by the bound OpenSSL module, and calls its own appropriate key zeroization functions. In both cases, these functions overwrite the memory with zeroes and deallocate the memory with the regular memory deallocation operating system call.

7. Electromagnetic Interference / Electromagnetic Compatibility (EMI/EMC)

The test platforms listed in Table 3 have been tested and found to conform to the EMI/EMC requirements specified by 47 Code of Federal Regulations, FCC PART 15, Subpart B, Unintentional Radiators, Digital Devices, Class A (i.e., Business use). These limits are designed to provide reasonable protection against harmful interference when the equipment is operated in a commercial environment. Each of the test platforms shall be installed and used in accordance with its instruction manual.

8. Self-Tests

8.1. Power-Up Tests

The module performs power-up tests when it is loaded into memory without operator intervention. Power-up tests ensure that the module is not corrupted and that the cryptographic algorithms work as expected.

While the module is executing the power-up tests, services are not available, and input and output are inhibited. The module is not available for use until the power-up tests complete successfully.

If any power-up test fails, the module will return the error message listed in section 9.2.2, enter the error state and terminate. Therefore, no cryptographic operations or data output are possible.

Note: The bound Ubuntu 20.04 OpenSSL Cryptographic Module and the Ubuntu 20.04 Kernel Crypto API Cryptographic Module perform their own power-up tests automatically when they are loaded into memory. The Ubuntu 20.04 Strongswan Cryptographic Module ensures that both bound modules complete their power-up tests successfully.

8.1.1. Integrity Tests

The integrity of the module is verified by comparing an HMAC-SHA-256 value calculated at run time with the HMAC value stored in the .hmac file that was computed at build time, for each of the components that comprise the module. The HMAC-SHA-256 algorithm and the HMAC key for integrity test are provided by the bound Ubuntu 20.04 Kernel Crypto API Cryptographic Module. If the HMAC values do not match, the test fails and the module enters the error state.

8.1.2. Cryptographic Algorithm Tests

The module performs the self-test on the following FIPS-Approved cryptographic algorithm supported in FIPS mode using the Known Answer Test (KAT) shown below:

Algorithm	Test
SP800-135 IKEv2 KDF	KAT KDF for IKEv2 using HMAC-SHA-1

Table 12- Self-Tests

For the KAT, the module calculates the result and compares it with the known answer. If the calculated value does not match the known answer, the KAT fails and the module enters the error state.

8.2. On-Demand Self-Tests

On-demand self-tests can be invoked by powering off and reloading the module, which cause the module to run the power-up tests again. During the execution of the on-demand self-tests, services are not available and no data output or input is possible.

8.3. Conditional Tests

The module does not perform conditional tests.

9. Guidance

In order to run in FIPS mode, the module must be operated using the FIPS Approved services, with their corresponding FIPS Approved and FIPS allowed cryptographic algorithms provided in this Security Policy (see sections 3.2 and 3.3). In addition, key sizes must comply with [SP800-131A].

9.1. Crypto Officer Guidance

The binaries of the module are contained in the Ubuntu packages for delivery. The Crypto Officer shall follow this Security Policy to configure the operational environment and install the module to be operated as a FIPS 140-2 validated module.

The following table lists the Ubuntu packages containing the FIPS validated module:

Processor Architecture	Ubuntu packages
x86_64	• strongswan_5.8.2-1ubuntu3.fips.3.1.2_all.deb • strongswan-charon_5.8.2-1ubuntu3.fips.3.1.2_amd64.deb • strongswan-hmac_5.8.2-1ubuntu3.fips.3.1.2_amd64.deb • strongswan-libcharon_5.8.2-1ubuntu3.fips.3.1.2_amd64.deb • strongswan-starter_5.8.2-1ubuntu3.fips.3.1.2_amd64.deb • libstrongswan_5.8.2-1ubuntu3.fips.3.1.2_amd64.deb • libstrongswan-standard-plugins_5.8.2-1ubuntu3.fips.3.1.2_amd64.deb

Table 13 - Ubuntu packages

Note: The prelink is not installed on Ubuntu by default. For proper operation of the in-module integrity verification, the prelink should be disabled.

9.1.1. Operating Environment Configurations

In order to configure the operating environment, the following bound modules must be installed:

- Ubuntu 20.04 Kernel Crypto API Cryptographic Module
- Ubuntu 20.04 OpenSSL Cryptographic Module

Please follow the instructions provided in the security policies ([KCAPI_SP] and [OPENSSL_SP]) to install and configure both modules in FIPS mode of operation.

Once these modules are installed and configured properly, the operating environment is configured to support FIPS operation. The Crypto Officer should check the existence of the file/proc/sys/crypto/fips_enabled, which content should be the character “1”. If the file does not exist or does not contain “1”, the operating environment is not configured to support FIPS and the module will not operate as a FIPS validated module properly.

9.1.2. Module Installation

Once the operating environment configuration is finished, the Crypto Officer can install the strongswan and strongswan-hmac Ubuntu packages using a normal packaging tool such as the Advanced Package Tool (APT), for example:

```
$ sudo apt-get install strongswan strongswan-hmac
```

This command will install the specified packages and all the dependent packages listed in Table 13.

To download the FIPS validated version of the module, please email "sales@canonical.com" or contact a Canonical representative, https://www.ubuntu.com/contact-us.

All Ubuntu packages are associated with hashes for integrity check. The integrity of the Ubuntu package is automatically verified by the packaging tool during the module installation. The Crypto Officer shall not install the Ubuntu package if the integrity of the Ubuntu package fails.

9.1.3. Module Configuration

For the module, the mode of operation is implicitly assumed depending on the services/security functions invoked as stated in section 3.2 and the successive sections lists the available ciphers from the module. Any use of non-approved ciphers or non-Approved key sizes will result in the module entering the non-FIPS mode of operation. With the operational environment setup as stated in the above section, the following restrictions are applicable. No more cipher addition is possible by configuration or command line options.

- Configure Charon as specified in ipsec.conf(5), and ipsec.secrets(5) man pages
- To start and stop the module, use the *ipsec* command.
- IKEv2 must be used In order to run the module in FIPS mode of operation, the following setting must be included in the ipsec.conf file:

  keyexchange = ikev2
- ikelifetime should not be larger than 1 hour:

  ikelifetime = 1h
- salifetime should not be larger than 1 hour:

  salifetime = 1h
- Galois Counter Mode (GCM) should be used with their full tag lengths
- Aggressive mode should not be used
- Stopping the module will zeroize the ephemeral CSPs and keys
- To check module status, read the Charon debug data using the ipsec statusall command line and the logs in /var/log/charon.log
- Only the FIPS 140-2 approved and allowed ciphers listed in section 3.2 shall be used in configuring the Charon daemon. Use of non-approved cipher will put the module in the non-FIPS mode implicitly.

NOTE: Encryption and decryption of data is done implicitly when the kernel triggers Charon to set up a new Security Association.

9.2. User Guidance

9.2.1. Managing the IKEv2 daemon

To start the IKEv2 daemon, use the following command:

```
# ipsec start
```

To stop the IKEv2 daemon, use the following command:

```
# ipsec stop
```

To start the IKEv2 daemon automatically at the system boot time, use the following command:

```
# systemctl enable strongswan
```

To prevent the IKEv2 daemon from automatically starting, use the following command:

```
# systemctl disable strongswan
```

See the ipsec(8), ipsec.conf(5) and ipsec.secrets(5) man pages for more information about how to operate the module.

To operate the module in FIPS mode, please consider the following restrictions:

- Only the IKEv2 cipher suites listed in Appendix A are available for use in the module (this is enforced by the module with the settings required in section 9.1.3).
- Use of RSA keys of less than 2048 for IKEv2 authentication will result in the module entering non-FIPS mode.
- The following ipsec subcommands must not be used:
  - ipsec _updown

9.2.2. Handling Self-Test Errors

When the module fails any self-test, it will return an error message to indicate the error and then enter the error state. The following table shows the list of error messages when the module fails any self-test.

Error Events	Error Messages
Integrity test script (/usr/lib/ipsec/_fipscheck) is missing	/usr/sbin/ipsec: 55: /usr/lib/ipsec/_fipscheck: not found ipsec: strongswan fips file integrity check failed
Integrity test program (/usr/bin/fipscheck) is missing	ipsec: strongswan fips file integrity check failed
HMAC file missing	ipsec: please install strongswan-hmac package required in fips mode ipsec: strongswan fips file integrity check failed
Integrity Test failure	ipsec: strongswan fips file integrity check failed
KAT failure	ipsec: strongswan fips ikev2 kdf self test failed
Self-test failure in Ubuntu 20.04 OpenSSL Cryptographic Module	fips.c(139): OpenSSL internal error, assertion failed: FATAL FIPS SELFTEST FAILURE Aborted (core dumped) ipsec: strongswan fips ikev2 kdf self test failed

Table 14 - Self-Tests

To recover from the error state, the module must be restarted and perform power-up tests again. If the failure persists, the module must be reinstalled.

Note: Self-test failures in the bound Ubuntu 20.04 OpenSSL Cryptographic Module will prevent the Ubuntu 20.04 Strongswan Cryptographic Module from operating. Refer to the Guidance section in the corresponding Non-Proprietary Security Policy [OPENSSL-SP] for instructions on handling self-test failures in the bound module.

9.2.3. AES-GCM IV

The Ubuntu 20.04 Strongswan Cryptographic Module is bound to the Ubuntu 20.04 OpenSSL Cryptographic Module which implements AES-GCM. This module, Strongswan, generates the IV for AES-GCM in OpenSSL through the IKEv2 key establishment protocol which is compliant with IG A.5 provision 1), "IPSec protocol IV generation".

The AES-GCM IV generation is in compliance with [RFC5282]. The module uses the [RFC7296] compliant IKEv2 protocol to establish the shared secret SKEYSEED from which the AES-GCM encryption keys are derived.

By the virtue of the lifetime limit (see above section 9.1.3), the IV is renegotiated before reaching 2^{64}. The IV does not get stored permanently. In case or normal or abnormal termination of the IKE/IPsec connection, the SA has to be renegotiated by the module.

10. Mitigation of Other Attacks

The module does not implement security mechanisms to mitigate other attacks.

Appendix A. IKEv2 Cipher Suites

The module supports the following cipher suites for the IKEv2 protocol in FIPS mode of operation. The module does not allow any other cipher suite.

Encryption algorithms

Keyword	Description
3des	Triple-DES in CBC mode with 168 bit key
aes or aes128	AES in CBC mode with 128 bit key
aes192	AES in CBC mode with 192 bit key
aes256	AES in CBC mode with 256 bit key
aes128gcm8 or aes128gcm64	AES-GCM mode with 128 bit key and 64 bit ICV
aes192gcm8 or aes192gcm64	AES-GCM mode with 192 bit key and 64 bit ICV
aes256gcm8 or aes256gcm64	AES-GCM mode with 256 bit and 64 bit ICV
aes128gcm12 or aes128gcm96	AES-GCM mode with 128 bit key and 96 bit ICV
aes192gcm12 or aes192gcm96	AES-GCM mode with 192 bit key and 96 bit ICV
aes256gcm12 or aes256gcm96	AES-GCM mode with 256 bit and 96 bit ICV
aes128gcm16 or aes128gcm128	AES-GCM mode with 128 bit key and 128 bit ICV
aes192gcm16 or aes192gcm128	AES-GCM mode with 192 bit key and 128 bit ICV
aes256gcm16 or aes256gcm128	AES-GCM mode with 256 bit and 128 bit ICV
chacha20poly1305	256 bit ChaCha20/Poly1305 with 128 bit ICV

Integrity algorithms

Keyword	Description
sha1 or sha	HMAC SHA-1 with 160 bits key and 96 bits truncation
sha1_160	HMAC SHA-1 with 160 bits key
sha256 or sha2_256	HMAC SHA-256 with 256 bits key
sha384 or sha2_384	HMAC SHA-384 with 384 bits key
sha512 or sha2_512	HMAC SHA-512 with 512 bits key
sha256_96 or sha2_256_96	HMAC SHA-256 with 256 bits key and 96 bits truncation

Pseudo-random functions (PRF) used in KDF

Keyword	Description
prfsha1	PRF using HMAC SHA-1
prfsha256	PRF using HMAC SHA-256
prfsha384	PRF using HMAC SHA-384
prfsha512	PRF using HMAC SHA-512

Diffie-Hellman groups

Keyword	Description
modp2048	Regular group 14 (2048-bit modulus)
modp3072	Regular group 15 (3072-bit modulus)
modp4096	Regular group 16 (4096-bit modulus)
modp8192	Regular group 18 (8192-bit modulus)
modp2048s224	Modulo Prime Group 23 (2048-bit modulus, 224-bit subgroup)
modp2048s256	Modulo Prime Group 24 (2048-bit modulus, 256-bit subgroup)
ecp224	NIST Elliptic Curve Group 26
ecp256	NIST Elliptic Curve Group 19
ecp384	NIST Elliptic Curve Group 20
ecp521	NIST Elliptic Curve Group 21

Appendix B. Glossary and Abbreviations

AES	Advanced Encryption Standard
AES-NI	Advanced Encryption Standard New Instructions
CAVP	Cryptographic Algorithm Validation Program
CAVS	Cryptographic Algorithm Validation System
CBC	Cipher Block Chaining
CMVP	Cryptographic Module Validation Program
CSP	Critical Security Parameter
CTR	Counter Mode
DES	Data Encryption Standard
DSA	Digital Signature Algorithm
DRBG	Deterministic Random Bit Generator
EC	Elliptic Curve
FCC	Federal Communications Commission
FIPS	Federal Information Processing Standards Publication
HMAC	Hash Message Authentication Code
IKE	Internet Key Exchange
KAT	Known Answer Test
KDF	Key Derivation Function
MAC	Message Authentication Code
NIST	National Institute of Science and Technology
PAA	Processor Algorithm Acceleration
PAI	Processor Algorithm Implementation
RSA	Rivest, Shamir, Addleman
SHA	Secure Hash Algorithm
SSH	Secure Shell

Appendix C. References

FIPS140-2	FIPS PUB 140-2 - Security Requirements For Cryptographic Modules May 2001 http://csrc.nist.gov/publications/fips/fips140-2/fips1402.pdf
FIPS140-2_IG	Implementation Guidance for FIPS PUB 140-2 and the Cryptographic Module Validation Program Ausgut 28, 2020 http://csrc.nist.gov/groups/STM/cmvp/documents/fips140-2/FIPS1402IG.pdf
OPENSSL-SP	Ubuntu 20.04 OpenSSL Cryptographic Module version 3.0 FIPS 140-2 Non-Proprietary Security Policy version 1.0 https://csrc.nist.gov/Projects/Cryptographic-Module-Validation-Program/Certificate/3966
KCAPI-SP	Ubuntu 20.04 Kernel Crypto API Cryptographic Module version 3.0 FIPS 140-2 Non-Proprietary Security Policy version 1.0 https://csrc.nist.gov/Projects/Cryptographic-Module-Validation-Program/Certificate/3928
RFC5282	Using Authenticated Encryption Algorithm with the Encrypted Payload of the Internet Key Exchange version 2 (IKEv2) Protocol August 2008 https://tools.ietf.org/html/rfc5282
RFC7296	Internet Key Exchange Protocol Version 2 (IKEv2) October 2014 https://tools.ietf.org/html/rfc7296
SP800-131A	NIST Special Publication 800-131A Revision 1- Transitions: Recommendation for Transitioning the Use of Cryptographic Algorithms and Key Lengths March 2019 https://nvlpubs.nist.gov/nistpubs/SpecialPublications/NIST.SP.800-131Ar2.pdf
SP800-135	NIST Special Publication 800-135 Revision 1 - Recommendation for Existing Application-Specific Key Derivation Functions December 2011 http://nvlpubs.nist.gov/nistpubs/Legacy/SP/nistspecialpublication800-135r1.pdf