ubuntu 18.04 OpenSSH Client Cryptographic Module

version 2.1

FIPS 140-2 Non-Proprietary Security Policy

Version 2.6

Last update: 2020-08-21

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 2.1 of the Ubuntu 18.04 OpenSSH Client 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 PUB 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 18.04 OpenSSH Client Cryptographic Module (also referred to as "the module") is a client application implementing the Secure Shell (SSH) protocol in the Ubuntu Operating System user space. The module interacts with an SSH server via the SSH protocol. The module only supports SSHv2 protocol.

The module uses the Ubuntu 18.04 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 SSH sessions. The Ubuntu 18.04 OpenSSL Cryptographic Module is a FIPS validated module (certificate #3622).

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 SSH client application and its integrity verification file. The following table enumerates the files that comprise each module variant:

Component	Description
/usr/bin/ssh	SSH client application
/usr/bin/.ssh.hmac	Integrity verification file for SSH client application

Table 2 - Cryptographic Module Components

The software block diagram below shows the module, its interfaces with the bound OpenSSL module, the operational environment and the delimitation of its logical boundary, which is depicted in the blue box:

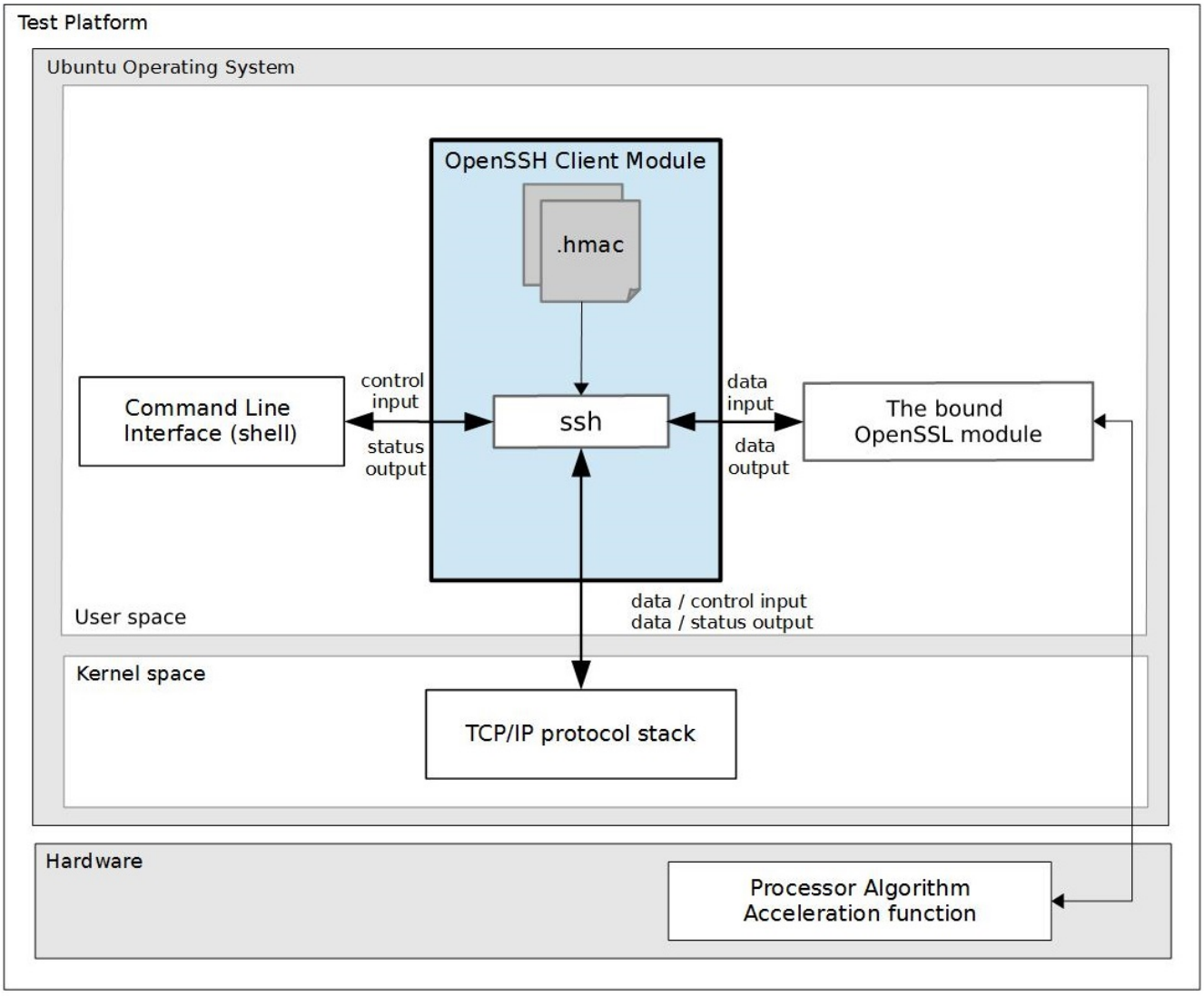

Figure 1 - Software Block Diagram

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:

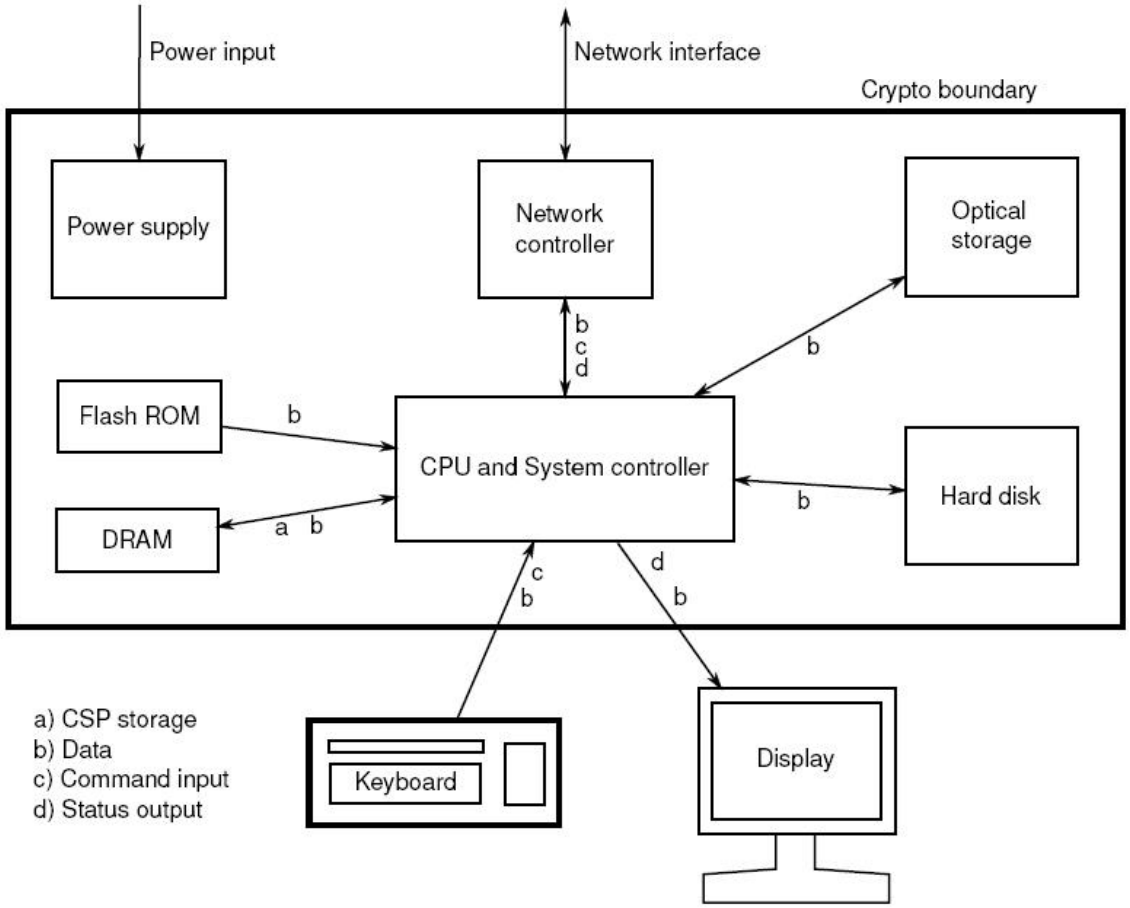

Figure 2 - Cryptographic Module Physical Boundary

The module has been tested on the platforms shown below:

Test Platform	Processor	Test Configuration
Supermicro SYS-5018R-WR	Intel® Xeon® CPU E5-2620v3	Ubuntu 18.04 LTS 64-bit with/without AES-NI (PAA)
IBM z/14	z14	Ubuntu 18.04 LTS 64-bit running on IBM z/VM with/without CPACF (PAI)

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 logical interfaces are the ssh command, the messages sent to and received from the SSH server, and the application program interface (API) provided by the bound OpenSSL module. The following table summarizes the four logical interfaces:

FIPS Interface	Physical Port	Logical Interface
Data Input	Keyboard, Ethernet port	Input parameters and data sent to the ssh command through the command line with the ~/.ssh/known_hosts file, /etc/ssh/ssh_known_hosts file, key files in ~/.ssh directory, input data received via the SSHv2 channel, input data received via local or remote port-forwarding port, input data received from the bound OpenSSL module via its API parameters.
Data Output	Display, Ethernet port	Output data returned by the ssh command, output data sent via the SSHv2 channel, output data sent via local or remote port-forwarding port, output data sent to the bound OpenSSL module via its API parameters.
Control Input	Keyboard, Ethernet port	Invocation of the ssh command on the command line, control parameters via the ssh command or via the /etc/ssh/ssh_config file and ~./.ssh/config file, SSHv2 protocol message requests received from SSH server.
Status Output	Display, Ethernet port	Status messages returned after execution of ssh command, status of processing SSHv2 protocol message requests.
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 SSH session, show status and self-tests.
- **Crypto Officer role**: performs services of module installation, configuration and terminate SSH client.

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 Table 6, 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: In Table 5, only the NIST SP800-135 SSH Key Derivation Function (KDF) algorithm is provided by the Ubuntu 18.04 OpenSSH Client Cryptographic Module and the Cryptographic Algorithm Validation System (CAVS) certificate numbers are listed in Table 7(A). All the other cryptographic algorithms listed in Table 5 are provided by the bound OpenSSL module, and the CAVS certificate numbers are listed in Table 7(B).

Service	Algorithms	Role	Access	Key / CSP
Establish SSH Session	Key exchange: • Diffie-Hellman with SHA-256 and key size between 2048-bit and 8192-bit[1]; • EC Diffie-Hellman with SHA-256/SHA-384/SHA-	User	Create	Client's Diffie-Hellman or EC Diffie-Hellman public and private keys; Server's Diffie-Hellman or EC Diffie-Hellman public and private keys; Diffie-Hellman or EC Diffie-Hellman

[1] Diffie-Hellman key agreement provides between 112 and 201 bits of encryption strength.

Service	Algorithms	Role	Access	Key / CSP
	512 and curve P-256/P-384/P-521[2]			shared secret
	Key derivation: • SP800-135 SSH KDF with SHA-1/SHA-256/SHA-384/SHA-512	User	Read	Diffie-Hellman or EC Diffie-Hellman shared secret
			Create	Session encryption keys (128/192/256-bit AES keys, 192-bit Triple-DES keys); Session data authentication keys (at least 112-bit HMAC keys)
Maintain SSH Session	Data Encryption and Decryption: • AES (CBC, CTR and GCM modes) • Triple-DES (CBC mode)	User	Read	Session encryption keys
	Data Integrity (MAC): • HMAC with SHA-1/SHA-256/SHA-512	User	Read	Session data authentication keys
Server Signature Verification	Signature verification used in SSH key-based authentication and certificate-based authentication: • DSA with SHA-1 and 1024-bit key size; • ECDSA with SHA-256/SHA-384/SHA-512 and curve P-256/P-384/P-521; • RSA with SHA-256/SHA-512 and key size between 2048-bit and 16384-bit	User	Read	Server's public keys
Client Signature Generation	Signature generation used in SSH key-based authentication and certificate-based authentication: • ECDSA with SHA-256/SHA-384/SHA-512 and curve P-	User	Read	Client's private keys; Client's public keys

[2] EC Diffie-Hellman key agreement provides between 128 and 256 bits of encryption strength.

Service	Algorithms	Role	Access	Key / CSP
	256/P-384/P-521; • RSA with SHA-256/SHA-512 and key size between 2048-bit and 16384-bit			
Close SSH session	N/A	User	Zeroize	All aforementioned CSPs
Terminate SSH client (ssh)	N/A	Crypto Officer	Zeroize	All aforementioned CSPs
Configure SSH client	N/A	Crypto Officer	N/A	None
Show status	N/A	User	N/A	None
Self-test (by restarting the module)	SP800-135 SSH KDF and HMAC	User	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
Server or Client Digital Signature using non-Approved algorithms or key sizes	Digital signature used in SSH key-based authentication and certificate-based authentication: • DSA signature generation with SHA-1 and 1024-bit key; • RSA signature generation with SHA-1 or key sizes smaller than 2048 bits, RSA signature verification with key sizes smaller than 1024 bits; •	User

Table 6 - Services in non-FIPS mode

3.3. Algorithms

Table 7(A) shows the Approved algorithm provided by the module in FIPS mode, which is tested and validated by CAVP.

Algorithm	Mode / Method	Use	Standard	CAVS Cert.
SP800-135 SSH KDF	SHA-1, SHA-256, SHA-384, SHA-512	Key derivation in the SSHv2 protocol.	[SP800-135]	CVL: #C740, #C741

Table 7(A) - Approved Algorithms provided by the OpenSSH Client Module

Table 7(B) shows the Approved and non-Approved but Allowed algorithms that are used by the module, but provided by the bound OpenSSL module in FIPS mode. The table includes the modes, methods and key lengths used specifically by the module. CAVS certificates for approved algorithms are also included; notice that not all algorithms validated in the CAVS certificates for the OpenSSL module are used by the OpenSSH module.

The OpenSSH and the bound OpenSSL module together provide the Diffie Hellman and EC Diffie Hellman key agreement algorithms. The OpenSSH 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 7(B).

Algorithm	Mode / Method	Key Lengths, Curves or Moduli (in bits)	Use	CAVS Cert.
AES	CBC, CTR	128, 192, 256	Data encryption and decryption	#C670, #C673, #C677, #C687, #C688
	GCM	128, 256	Data encryption and decryption	#C671, #C672, #C692, #C674, #C675, #C676, #C678, #C679, #C680, #C687, #C688, #C689, #C690, #C691
ECC CDH Primitive		P-256, P-384, P-521	Shared secret computation	CVL: #C682, #C683, #C684, #C685, #C687, #C688
DRBG	**CTR_DRBG:** AES-128, AES-192, AES-256 with/without DF, without PR		Random Number Generator	#C670, #C673, #C677, #C687, #C688

Algorithm	Mode / Method	Key Lengths, Curves or Moduli (in bits)	Use	CAVS Cert.
DSA	SHA-1	1024	Signature verification for SSH key–based authentication	#C682, #C683, #C684, #C685, #C687, #C688
ECDSA	SHA-256, SHA-384, SHA-512	P-256, P-384, P-521	Signature generation and verification	#C682, #C683, #C684, #C685, #C687, #C688
HMAC	SHA-1, SHA-256, SHA-512		Data Integrity	#C682, #C683, #C684, #C685, #C687, #C688
RSA	SHA-256, SHA-512	2048, 3072 and 4096	Signature generation	#C682, #C683, #C684, #C685, #C687, #C688
	SHA-256, SHA-512	greater than 4096	Signature generation	allowed per [SP800-131A]
	SHA-256, SHA-512	1024, 2048 and 3072	Signature verification	#C682, #C683, #C684, #C685, #C687, #C688
	SHA-256, SHA-512	greater than 3072	Signature verification	allowed per [SP800-131A]
SHS	SHA-1, SHA-256, SHA-384, SHA-512		Message Digest	#C682, #C683, #C684, #C685, #C687, #C688
Triple-DES	CBC	192	Data encryption and decryption	#C669, #C686
Partial Diffie-Hellman key agreement (shared secret computation)	SHA-256, SHA-512	2048, 4096, 8192	Key Agreement	CVL: #C682, #C683, #C684, #C685, #C687, #C688
Partial EC Diffie-Hellman key agreement (shared secret computation)	SHA-256, SHA-512	P-256, P-384, P-521	Key Agreement	CVL: #C682, #C683, #C684, #C685, #C687, #C688

Table 7(B) – Approved or Allowed Algorithms provided by the bound OpenSSL Module

Table 7(A) shows the non-Approved algorithm provided by the module in non-FIPS mode.

Table 9 shows the non-Approved algorithms that are used by the module, but provided by the bound OpenSSL module in non-FIPS mode (there are no non-Approved algorithms provided by the module).

Algorithm	Use
DSA signature generation with SHA-1 and 1024-bit key	Signature generation
RSA signature generation with SHA-1 or with key sizes smaller than 2048 bits, RSA signature verification with key sizes smaller than 1024 bits	Signature generation and verification

Table 8 - Non-Approved Algorithms provided by the bound OpenSSL Module

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 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
Client's private keys (ECDSA, RSA)	N/A	Entry: The keys are read from the key files. Output: The keys are output to the bound OpenSSL module via API parameters.	Zeroized automatically when closing SSH session or terminating SSH client.
Client's public keys (ECDSA, RSA)	N/A	Entry: The keys are read from the key files. Output: The keys are output during SSH handshake.	
Server's public keys (DSA, ECDSA, RSA)	N/A	Entry: The keys are entered during SSH handshake. Output: The keys are output to the bound OpenSSL module via API parameters.	
Client's Diffie-Hellman or EC Diffie-Hellman public keys	N/A (generated by the bound OpenSSL module)	Entry: The keys are entered from the bound OpenSSL module via API parameters. Output: The keys are output during SSH handshake.	
Client's Diffie-Hellman or EC Diffie-Hellman private keys	N/A (generated by the bound OpenSSL module)	Entry: The keys are entered from the bound OpenSSL module via API parameters. Output: The keys are output to the bound OpenSSL module via API parameters.	
Server's Diffie-Hellman or EC Diffie-Hellman public keys	N/A	Entry: The keys are entered during SSH handshake. Output: The keys are 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: The keys are entered from the bound OpenSSL module via API parameters. Output: N/A	

Session encryption keys (AES, Triple-DES)	N/A (derived from the shared secret via SP800-135 SSH KDF)	Entry: N/A Output: The keys are output to the bound OpenSSL module via API parameters.
Session data authentication keys (HMAC)	N/A (derived from the shared secret via SP800-135 SSH KDF)	Entry: N/A Output: The keys are output to the bound OpenSSL module via API parameters.

Table 9 - 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 service provided by the bound OpenSSL 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 Derivation

The module implements SP800-135 SSH KDF for the SSHv2 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.

The client's public and private keys are stored in the key files in ~/.ssh directory, which are within the module's physical boundary but outside its logical boundary.

The HMAC key used for the Integrity Test is stored in the module and relies on the operating system for protection.

6.6. Key / CSP Zeroization

Zeroization occurs when the "Close SSH session" and the "Terminate SSH client (ssh)" services are invoked.

The memory occupied by keys is allocated by regular memory allocation operating system calls. The module calls appropriate key zeroization functions provided by the bound OpenSSL module, which overwrite the memory occupied by keys with "zeros" and deallocate the memory with the regular memory deallocation

operating system call. The module also provides the key zeroization method to overwrite the memory occupied by keys with “zeros” when the module receives internal error codes or the module is terminated.

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 and enter the error state. In error state, the SSH client is terminated and thus no cryptographic operations or data output are possible.

Note: The bound OpenSSL module performs its own power-up tests automatically when it is loaded into memory. The Ubuntu 18.04 OpenSSH Client Cryptographic Module ensures that the bound OpenSSL module must complete its 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. The HMAC-SHA-256 algorithm is provided by the bound OpenSSL 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 SSH KDF	KAT KDF for SSH

Table 10 - Self-Tests

For the KAT, the module calculates the result and compares it with the known value. 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

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	openssh-client_1:7.9p1-10~ubuntu18.04.fips.0.2_amd64.deb openssh-client-hmac_1: 7.9p1-10~ubuntu18.04.fips.0.2_amd64.deb
z System	openssh-client_1: 7.9p1-10~ubuntu18.04.fips.0.2_s390x.deb openssh-client-hmac_1:7.9p1-4.fips.4~18.04.0.2_s390x.deb

Table 11- 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

To configure the operating environment to support FIPS, the following shall be performed with the root privilege:

(1) Install the following linux-fips and fips-initramfs Ubuntu packages depending on the target operational environment:

Processor Architecture	Ubuntu packages
x86_64	fips-initramfs_0.0.10_amd64.deb linux-fips_4.15.0.1011.10_amd64.deb
z System	fips-initramfs_0.0.10_s390x.deb linux-fips_4.15.0.1011.10_s390x.deb

(2) Add fips=1 to the kernel command line.

- For x86_64 systems, create the file /etc/default/grub.d/99-fips.cfg with the content: GRUB_CMDLINE_LINUX_DEFAULT=”$GRUB_CMDLINE_LINUX_DEFAULT fips=1”.
- For z systems, edit /etc/zipl.conf file and append the "fips=1" in the parameters line for the specified boot image.

(3) If /boot resides on a separate partition, the kernel parameter bootdev=UUID=<UUID of partition> must also be appended in the aforementioned grub or zipl.conf file. Please see the following **Note** for more details.

(4) Update the boot loader.

- For the x86_64 system, execute the update-grub command.
- For the z system, execute the zipl command.

(5) Execute reboot to reboot the system with the new settings.

Now, the operating environment is configured to support FIPS operation. The Crypto Officer should check the existence of the file, /proc/sys/crypto/fips_enabled, and that it contains "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.

Note: If /boot resides on a separate partition, the kernel parameter bootdev=UUID=<UUID of partition> must be supplied. The partition can be identified with the command df /boot. For example:

```
$ df /boot
Filesystem     1K-blocks   Used Available Use% Mounted on
/dev/sdb2        241965 127948    101525  56% /boot
```

The UUID of the /boot partition can be found by using the command grep /boot /etc/fstab . For example:

```
$ grep /boot /etc/fstab
# /boot was on /dev/sdb2 during installation
UUID=cec0abe7-14a6-4e72-83ba-b912468bbb38 /boot    ext2    defaults    0    2
```

Then, the following string needs to be appended to the /etc/default/grub file:

```
GRUB_CMDLINE_LINUX_DEFAULT="quiet bootdev=UUID=cec0abe7-14a6-4e72-83ba-b912468bbb38
fips=1"
```

Once the operating environment configuration is finished, the Crypto Officer can install the Ubuntu 18.04 OpenSSL Cryptographic Module following the instructions provided in the Guidance section of the FIPS 140-2 Non-Proprietary Security Policy [OPENSSL-SP].

9.1.2. Module Installation

Canonical distributes the module via Personal Package Archives (PPA), whose access is granted to users with a valid subscription. In order to obtain a subscription and download the FIPS validated version of the module, please email "sales@canonical.com" or contact a Canonical representative, https://www.ubuntu.com/contact-us. Canonical provides specific instructions to configure the system to get access to the corresponding PPA.

Once the operating environment is configured following the instructions provided in section 9.1.1, configuration to access the PPA is complete, and the OpenSSL module is installed and configured, the Crypto Officer can install the openssh-client and openssh-client-hmac Ubuntu packages listed in Table 11 using the Advanced Package Tool (APT) with the following command line:

```
$ sudo apt-get install openssh-client openssh-client-hmac
```

All the 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.

9.2. User Guidance

This module is designed to be used by connecting it with an SSH server. To use a FIPS validated SSH server, please see the Ubuntu OpenSSH Server Cryptographic Module FIPS 140-2 Non-Proprietary Security Policy.

When connecting to an unknown server, the user will be prompted to verify a fingerprint of the server's public key. This must be done by consulting a trusted source.

9.2.1. Starting an OpenSSH Client

To start the ssh client, use the following command:

```
$ ssh user@hostname
```

To stop the ssh program, use the exit command within the shell that is prompted after the login.

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

- Only the SSHv2 cipher suites listed in Appendix A are available to be used.
- Use of 1024-bit DSA keys for signature generation in SSH key-based or certificate-based authentication will result in the module entering non-FIPS mode implicitly. The DSA signature verification with 1024-bit key is only for legacy use.
- Use of less than 2048-bit RSA keys for signature generation or less than 1024-bit RSA keys for signature verification will result in the module entering non-FIPS mode implicitly.
- See the man pages of ssh command for more information about how to operate the module.

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 enters the error state. The following table shows the list of error messages when the module fails any self-test.

Error Events	Error Messages
When the Integrity Test fails at the power-up	“fips: mandatory checksum failed – aborting” “fips: mandatory checksum data missing –aborting”
When the KAT fails at the power-up	“ssh self test failed, aborting”

Table 12 - 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 OpenSSL module will prevent the module, “Ubuntu 18.04 OpenSSH Client Cryptographic Module”, from operating. See the Guidance section in the Ubuntu 18.04 OpenSSL Cryptographic Module FIPS 140-2 Non-Proprietary Security Policy [OPENSSL-SP] for instructions on handling OpenSSL self-test failures.

10. Mitigation of Other Attacks

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

Appendix A. SSHv2 Cipher Suites

In FIPS mode, the module only supports the following cipher suites for the different aspects of the SSHv2 protocol:

Encryption / Decryption	Description	Reference
3des-cbc	Three-key Triple-DES in CBC mode	RFC4253
aes128-cbc	AES in CBC mode with 128-bit key	RFC4253
aes192-cbc	AES in CBC mode with 192-bit key	RFC4253
aes256-cbc/rijndael-cbc@lysator.liu.se	AES in CBC mode with 256-bit key	RFC4253
aes128-ctr	AES in CTR mode with 128-bit key	RFC4344
aes192-ctr	AES in CTR mode with 192-bit key	RFC4344
aes256-ctr	AES in CTR mode with 256-bit key	RFC4344
aes128-gcm@openssh.com	AES in GCM mode with 128-bit key	RFC5647
aes256-gcm@openssh.com	AES in GCM mode with 256-bit key	RFC5647

Table 13 – Symmetric ciphers allowed in FIPS mode

Data Integrity (MAC)	Description	Reference
hmac-sha1/ hmac-sha1-etm@openssh.com	HMAC-SHA-1	RFC4253
hmac-sha2-256/ hmac-sha2-256-etm@openssh.com	HMAC-SHA-256	RFC6668
hmac-sha2-512/ hmac-sha2-512-etm@openssh.com	HMAC-SHA-512	RFC6668

Table 14 - Data Integrity algorithms (MAC) allowed in FIPS mode

Key Exchange	Description	Reference
diffie-hellman-group-exchange-sha256	Diffie-Hellman with SHA-256	RFC4419
diffie-hellman-group14-sha256	Diffie-Hellman with 2048-bit MODP Group and SHA-256	RFC8268
diffie-hellman-group16-sha512	Diffie-Hellman with 4096-bit MODP Group and SHA-512	RFC8268
diffie-hellman-group18-sha512	Diffie-Hellman with 8192-bit MODP Group and SHA-512	RFC8268

Key Exchange	Description	Reference
ecdh-sha2-nistp256	EC Diffie-Hellman with SHA-256 and NIST P-256 key	RFC5656
ecdh-sha2-nistp384	EC Diffie-Hellman with SHA-384 and NIST P-384 key	RFC5656
ecdh-sha2-nistp521	EC Diffie-Hellman with SHA-512 and NIST P-521 key	RFC5656

Table 15 - Key Exchange Methods allowed in FIPS mode

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
KAT	Known Answer Test
KDF	Key Derivation Function
MAC	Message Authentication Code
NDRNG	Non-Deterministic Random Number Generator
NIST	National Institute of Science and Technology
PAA	Processor Algorithm Acceleration
PAI	Processor Algorithm Implementation
PPA	Personal Package Archive
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**
February 5, 2019
http://csrc.nist.gov/groups/STM/cmvp/documents/fips140-2/FIPS1402IG.pdf

OPENSSL-SP **Ubuntu 18.04 OpenSSL Cryptographic Module FIPS 140-2 Non-Proprietary Security Policy**
https://csrc.nist.gov/CSRC/media/projects/cryptographic-module-validation-program/documents/security-policies/140sp3622.pdf

SP800-90A **NIST Special Publication 800-90A - Revision 1 - Recommendation for Random Number Generation Using Deterministic Random Bit Generators**
June 2015
http://nvlpubs.nist.gov/nistpubs/SpecialPublications/NIST.SP.800-90Ar1.pdf

SP800-131A **NIST Special Publication 800-131A Revision 2- 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