ubuntu 18.04 Libgcrypt Cryptographic Module

Module Version 1.0

FIPS 140-2 Non-Proprietary Security Policy

Document Version 1.2

Last update: November 19, 2020

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 Security Policy for version 1.0 of the Ubuntu 18.04 Libgcrypt 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 (Federal Information Processing Standards Publication 140-2) for a Security Level 1 software 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 Libgcrypt Cryptographic Module (hereafter referred to as "the module") is a set of software libraries implementing general purpose cryptographic algorithms. The module provides cryptographic services to applications running in the user space of the underlying Ubuntu operating system through a C language Application Program Interface (API). The module utilizes processor instructions to optimize and increase performance.

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	1
Overall Level		1

Table 1 - Security Levels

The cryptographic logical boundary consists of all shared libraries and the integrity check files used for Integrity Tests. The following table enumerates the files that comprise the module.

Component	Description
libgcrypt.so.20	Libgcrypt shared library
.libgcrypt.so.20.hmac	Libgcrypt shared library HMAC integrity file

Table 2 - Cryptographic Module Components

The software block diagram below shows the module, its interfaces with the operational environment and the delimitation of its logical boundary, comprised of all the components within the ORANGE box.

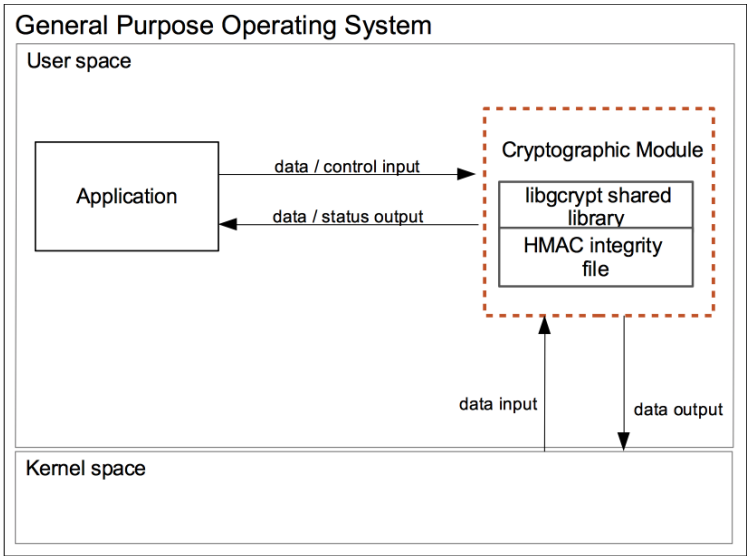

Figure 1 - Software Block Diagram

The module is aimed to run on a general-purpose computer (GPC); the physical boundary of the module is the tested platforms. Figure 2 shows the major components of a GPC.

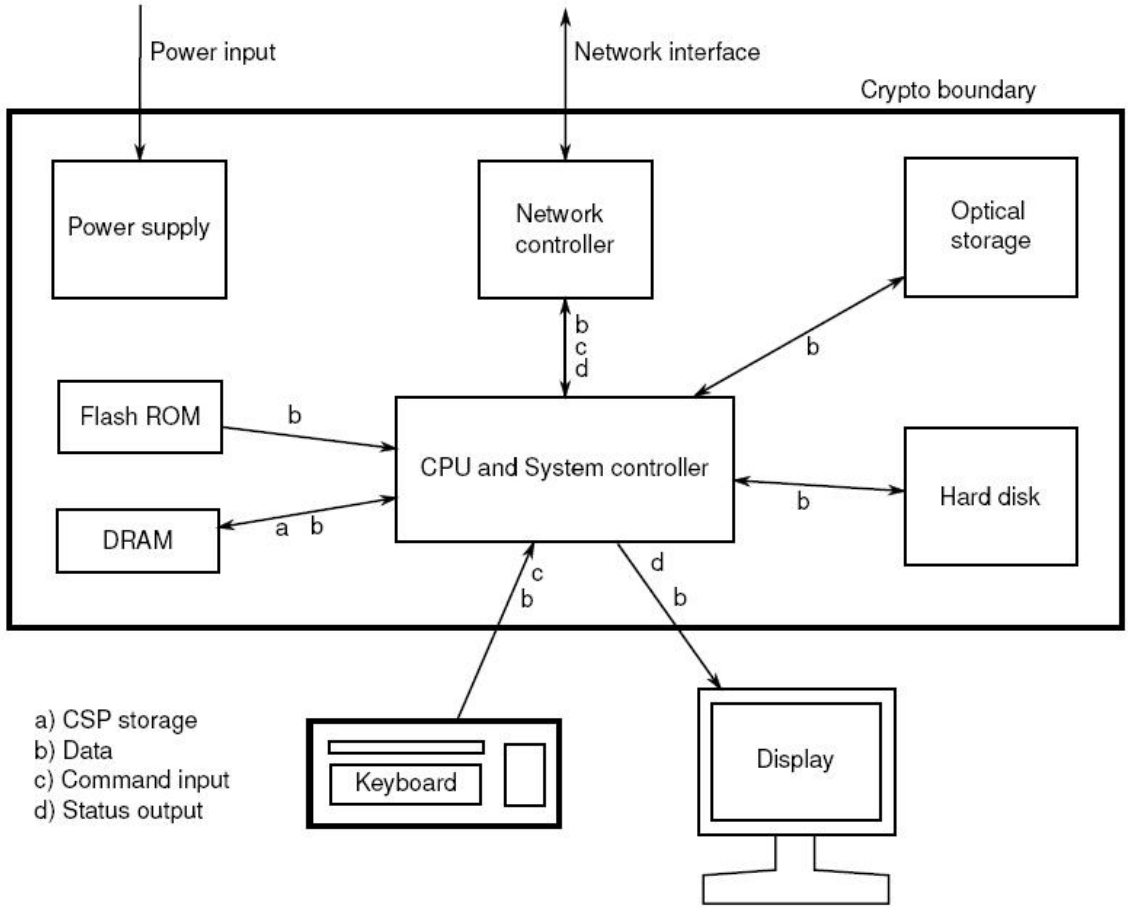

Figure 2 - Cryptographic Module Physical Boundary

The module has been tested on the test platforms shown below.

Test Platform	Processor	Test Configuration
Supermicro SYS-5018R-WR	Intel® Xeon® CPU E5-2620v3	Ubuntu 18.04 LTS 64-bit

Table 3 - Tested Platforms

Note: Per FIPS 140-2 IG G.5, the Cryptographic Module Validation Program (CMVP) makes no statement as to the correct operation of the module or the security strengths of the generated keys when this module is ported and executed in an operational environment not listed on the validation certificate.

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.

The module maintains separate contexts for each cryptographic operation. Therefore, Critical Security Parameters (CSPs) used and 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 API through which applications request services. The following table summarizes the four logical interfaces.

FIPS Interface	Logical Interface
Data Input	API input parameters
Data Output	API output parameters
Control Input	API function calls, API input parameters for control
Status Output	API return codes

Table 4 - Ports and Interfaces

3. Roles, Services and Authentication

3.1. Roles

The module supports the following roles:

- **User role**: performs cryptographic services (in both FIPS mode and non-FIPS mode), key zeroization, get status, and on-demand self-test.
- **Crypto Officer role**: performs module installation and initialization.

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 (i.e., man pages) referenced in section 9.1.

The table below shows the services available in FIPS mode. For each service, the associated cryptographic algorithms, the roles to perform the service, and the cryptographic keys or Critical Security Parameters and their access rights are listed. 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.

The complete list of cryptographic algorithms, modes and key lengths, and their corresponding Automated Cryptographic Validation Protocol (ACVP) certificate numbers can be found in Table 7 and Table 8 of this security policy. Notice that the algorithms mentioned in the Network Protocol Services correspond to the same implementation of the algorithms described in the Cryptographic Library Services.

Service	Algorithms	Role	Access	Keys/CSP
Cryptographic Library Services				
Symmetric Encryption and Decryption	AES	User	Read	AES key
	Three-key Triple-DES	User	Read	Triple-DES key
Symmetric Decryption	Two-key Triple-DES	User	Read	Triple-DES key
RSA key generation	RSA, DRBG	User	Create	RSA public-private key
RSA digital signature generation and verification	RSA	User	Read	RSA public-private key
DSA key generation	DSA, DRBG	User	Create	DSA public-private key

Service	Algorithms	Role	Access	Keys/CSP
DSA domain parameter generation and verification	DSA	User	n/a	n/a
DSA digital signature generation and verification	DSA	User	Read	DSA public-private key
ECDSA key generation	ECDSA, DRBG	User	Create	ECDSA public-private key
ECDSA public key validation	ECDSA	User	Read	ECDSA public key
ECDSA signature generation and verification	ECDSA	User	Read	ECDSA public and private keys
Random number generation	DRBG	User	Read, Update	Entropy input string, Internal state
Message digest	SHA-1, SHA-224, SHA-256, SHA-384, SHA-512, SHA3-224, SHA3-256, SHA3-384, SHA3-512, SHAKE-128, SHAKE-256	User	n/a	n/a
Message authentication code (MAC)	HMAC	User	Read	HMAC key
	CMAC	User	Read	AES or Triple-DES key
Key wrapping	AES	User	Read	AES key
Key encapsulation	RSA	User	Read	RSA public and private keys
Key Derivation	SP 800-132 PBKDF	User	Create, Read	Password, Derived key
Other FIPS-Related Services				
Show status	n/a	User	n/a	None
Zeroization	n/a	User	Zeroize	All CSPs
Self-Tests	AES, Triple-DES, SHS, HMAC, DSA, RSA, ECDSA, DRBG	User	n/a	None
Module installation	n/a	Crypto Officer	n/a	None
Module initialization	n/a	Crypto Officer	n/a	None

Table 5 - Services in FIPS mode of operation

The table below lists the services only available in non-FIPS mode of operation.

Service	Algorithms / Key sizes	Role	Access	Keys
Cryptographic Library Services				
Symmetric encryption and decryption	ARC4, Blowfish, Camellia, CAST5, DES, IDEA, RC2, SEED, Serpent, Twofish, GOST	User	Read	Symmetric key
Symmetric encryption	2-key Triple-DES	User	Read	2-key Triple-DES key
Asymmetric key generation	El Gamal; RSA/DSA keys in Table 9	User	Create	RSA, DSA or El Gamal public and private keys
Digital signature generation/verification	El Gamal; RSA/ DSA keys in Table 9; Signature generation using SHA-1	User	Read	RSA, DSA or El Gamal public and private keys
Asymmetric encryption and decryption	RSA keys in Table 9; El Gamal	User	Read	RSA, El Gamal public and private keys
Message digest	Tiger, MD4, MD5, Whirlpool, RIPE-MD 160, GOST	User	n/a	none
Message authentication code (MAC) using keys disallowed by [SP800-131A]	HMAC listed in Table 9; CMAC with 2-key Triple-DES	User	Read	HMAC key, 2-key Triple-DES key
Random Number Generation	CSPRNG	User	Read	none
Password Based Key Derivation Function	Password based KDF (RFC 4880)	User	Read	Password and derived key
Cyclic Redundancy Check	CRC32	User	Read	none

Table 6 – Services in non-FIPS mode of operation

3.3. Algorithms

The algorithms implemented in the module are tested and validated by CAVP for the following operating environment:

- Ubuntu 18.04 LTS 64-bit running on Intel® Xeon® CPU E5-2620v3 processor

3.3.1. Ubuntu 18.04 LTS 64-bit Running on Intel® Xeon® CPU E5-2620v3 Processor

The following table shows all algorithms with the associated CAVS certificates for the module.

Algorithm	Mode / Method	Key Lengths, Curves or Moduli (in bits)	Use	Standard	CAVP Certs
AES	ECB, CBC, OFB, CFB8, CFB128, CTR	128, 192, 256	Data Encryption and Decryption	[FIPS197], [SP800-38A]	A184
	CMAC	128, 192, 256	MAC Generation and Verification	[SP800-38B]	
	CCM	128, 192, 256	Data Encryption and Decryption	[SP800-38C]	
	XTS	128, 256	Data Encryption and Decryption for Data Storage	[SP800-38E]	
	KW	128, 192, 256	Key Wrapping and Unwrapping	[SP800-38F]	
DRBG	CTR_DRBG HMAC_DRBG HASH_DRBG (with/without PR)	• AES-128/192/256, with DF • SHA-1, SHA-256, SHA-512	Deterministic Random Bit Generation	[SP800-90A]	
DSA	N/A	L=2048, N=224 L=2048, N=256 L=3072, N=256	Key Pair Generation	[FIPS186-4]	A184
	SHA-224	L=2048, N=224	Domain Parameter Generation		
	SHA-256	L=2048, N=256 L=3072, N=256			
	SHA-224	L=2048, N=224	Digital Signature Generation		
	SHA-224 SHA-256	L=2048, N=256 L=3072, N=256			
	SHA-224	L=2048, N=224	Domain Parameter Verification		
	SHA-224 SHA-256	L=2048, N=256 L=3072, N=256			
	SHA-1	L=1024, N=160	Digital Signature Verification		
	SHA-224	L=2048, N=224			
	SHA-224 SHA-256	L=2048, N=256 L=3072, N=256			

Algorithm	Mode / Method	Key Lengths, Curves or Moduli (in bits)	Use	Standard	CAVP Certs
ECDSA	N/A	P-256, P-384, P-521	Key Pair Generation	[FIPS186-4]	A184
	SHA-224 SHA-256 SHA-384 SHA-512		Digital Signature Generation		
	SHA-1 SHA-224 SHA-256 SHA-384 SHA-512		Digital Signature Verification		
HMAC	SHA-1 SHA-224 SHA-256 SHA-384 SHA-512 SHA3-224 SHA3-256 SHA3-384 SHA3-512	112 or greater	Message Authentication Code	[FIPS198-1]	
RSA	X9.31	2048, 3072, 4096	Key Pair Generation	[FIPS186-4]	A184
	PKCS#1v1.5 and PSS with: SHA-224 SHA-256 SHA-384 SHA-512	2048, 3072, 4096	Digital Signature Generation		
			Digital Signature Verification		
SHS	SHA-1 SHA-224 SHA-256 SHA-384 SHA-512	N/A	Message Digest	[FIPS180-4]	
SHA-3	SHA3-224 SHA3-256 SHA3-384 SHA3-512 SHAKE-128 SHAKE-256			[FIPS202]	

Algorithm	Mode / Method	Key Lengths, Curves or Moduli (in bits)	Use	Standard	CAVP Certs
Triple-DES	ECB CBC CTR CFB8 CFB64 OFB	192 (two-key Triple-DES)	Data Decryption	[SP800-67], [SP800-38A]	
		192 (three-key Triple-DES)	Data Encryption and Decryption		
	CMAC	192	MAC Generation and Verification	[SP800-67], [SP800-38B]	A184
PBKDF (Vendor affirmed[1])	SHA-1 SHA-224 SHA-256 SHA-384 SHA-512 SHA3-224 SHA3-256 SHA3-384 SHA3-512	Iteration Count: 10-1000 Password Length: 8-128 Salt Length: 128-4096 Increment 8 Key Data Length: 128-4096 Increment 8	Password Based Key Derivation Function	[SP800-132]	

Table 7 - Cryptographic Algorithms for Intel® Xeon® CPU E5-2620v3 Processor

3.3.2. Allowed Algorithms

The following table describes the non-Approved but allowed algorithms in FIPS mode:

Algorithm	Caveat	Use
RSA Key encapsulation with Encryption and Decryption Primitives with keys equal or larger than 2048 bits up to 15360 or more.	Provides between 112 and 256 bits of encryption strength	Key Establishment; allowed per [FIPS140-2_IG] D.9
NDRNG	N/A	The module obtains the entropy data from NDRNG to seed the DRBG.

Table 8 – FIPS-Allowed Cryptographic Algorithms

[1] The PBKDF algorithm although CAVS tested is still vendor affirmed because KAT is not implemented.

3.3.3. Non-Approved Algorithms

The table below shows the non-Approved cryptographic algorithms implemented in the module that are only available in non-FIPS mode.

Algorithm	Use
ARC4	Encrypt/Decrypt
Blowfish	
Camellia	
CAST5	
DES	
IDEA	
RC2	
SEED	
Serpent	
Twofish	
GOST	
2-Key Triple-DES	Encryption; CMAC
RSA	Key generation, signature generation, key encapsulation with keys less than 2048 bits
RSA	Signature verification with keys smaller than 1024 bits modulus size
DSA	Parameter verification, Parameter/Key generation/Signature generation with keys and hash sizes not listed in Table 3
GOST	28147 Encryption, R 34.11-94 Hashing, R 34.11.2012 (Stribog) Hashing
HMAC	Using keys less than 112 bits.
CSPRNG	Generating random numbers
Tiger	Hashing
MD4	
MD5	
Whirlpool	

Algorithm	Use
RIPEMD 160	
GOST	
El Gamal	Key generation/encryption/decryption/signature generation and verification
CRC32	Cyclic redundancy check
OpenPGP Salted and Iterated/Salted	Password based KDF (RFC 4880)
SHA-1	Used for signature Generation

Table 9 - Non-Approved Cryptographic Algorithms

3.4. Operator Authentication

The module does not implement user 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 - Tested Platforms.

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 Critical Security Parameters (CSPs) that are used by the cryptographic services implemented in the module:

Name	Generation	Entry and Output	Zeroization
AES keys	Not Applicable. The key material is entered via API parameter.	The key is passed into the module via API input parameters in plaintext.	Automatically zeroized when freeing the cipher handler by calling gcry_cipher_close ()
Triple-DES keys			
HMAC keys			Automatically zeroized when freeing the cipher handler by calling gcry_md_close()
RSA private keys	The private keys are generated using FIPS 186-4 Key Generation method, and the random value used in the key generation is generated using SP800-90A DRBG.	The key is passed into the module via API input parameters in plaintext. The key is passed out of the module via API output parameters in plaintext.	Automatically zeroized when freeing the cipher handler by calling gcry_sexp_release ()
DSA private keys			
ECDSA private keys			
Entropy input string and seed	Obtained from the NDRNG.	None	Automatically zeroized when freeing DRBG handler by calling gcry_drbg_uninstantiate()
DRBG internal state (V, C, Key)	During DRBG initialization.		
Password	Not Applicable. The password is passed into the module via API input parameter.	The password is passed into the module via API input parameters in plaintext.	Automatically zeroized on module power off
Derived Key	Derived using SP 800-132 PBKDF	The key is passed out of the module via API output parameters in plaintext.	Automatically zeroized on module power off

Table 10 - Life cycle of Critical Security Parameters (CSP)

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

6.1. Random Number Generation

The module employs a Deterministic Random Bit Generator (DRBG) based on [SP800-90A] for the creation of seeds for asymmetric keys. In addition, the module provides a Random Number Generation service to calling applications.

The DRBG supports the CTR_DRBG, HMAC_DRBG and Hash_DRBG mechanism. The DRBG is initialized during module initialization; the module loads by default the DRBG using the CTR_DRBG mechanism with AES-256 and derivation function without prediction resistance. A different DRBG mechanism can be chosen through an API function call.

The module uses a Non-Deterministic Random Number Generator (NDRNG), getrandom() system call, as the entropy source for seeding the DRBG. The NDRNG is provided by the operational environment (i.e., Linux RNG), which is within the module's physical boundary but outside of the module's logical boundary. The NDRNG provides at least 256 bits of entropy to the DRBG during initialization (seed) and reseeding (reseed).

The Ubuntu Linux kernel performs conditional self-tests on the output of NDRNG to ensure that consecutive random numbers do not repeat. The module performs the DRBG health tests as defined in section 11.3 of [SP800-90A].

6.2. Key Generation

The Module provides an SP800-90A-compliant Deterministic Random Bit Generator (DRBG) for creation of key components of asymmetric keys, and random number generation.

The getrandom() system call from the Operational Environment is used as a source of random numbers for DRBG seeds and entropy input string.

The Key Generation methods implemented in the module for Approved services in FIPS mode is compliant with [SP800-133] (vendor affirmed).

For generating RSA, DSA and ECDSA keys the module implements asymmetric key generation services compliant with [FIPS186-4]. A seed (i.e. the random value) used in asymmetric key generation is directly obtained from the [SP800-90A] DRBG.

6.3. Key Transport / Key Derivation

The module provides key wrapping using the AES with KW mode and RSA key encapsulation using private key encryption and public key decryption primitives.

According to Table 2: Comparable strengths in [SP 800-57], the key sizes of AES and RSA provides the following security strength in FIPS mode of operation:

- AES key wrapping provides between 128 and 256 bits of encryption strength.
- RSA key wrapping[2] provides between 112 and 256 bits of encryption strength.

Note: As the module supports the size of RSA key pair greater than 2048 bits up to 15360 bits or more, the encryption strength 256 bits is claimed for RSA key encapsulations key agreement.

In addition, the module implements key derivation using the SP 800-132 PBKDF2 vendor affirmed algorithm as per D.6. The module supports option 1a from Section 5.4 of SP 800-132, whereby the Master Key (MK) is used

[2] "Key wrapping" is used instead of "key encapsulation" to show how the algorithm will appear in the certificate per IG G.13.

directly as the Data Protection Key (DPK). The keys derived from SP 800-132 KDF map to section 4.1 of SP 800-133 as indirect generation from DRBG.

6.4. Key Entry / Output

The module does not support manual key entry or intermediate key generation key output. The keys are provided to the module via API input parameters in plaintext form and output via API output parameters in plaintext form. This is allowed by [FIPS140-2_IG] IG 7.7, according to the “CM Software to/from App Software via GPC INT Path” entry on the Key Establishment Table.

6.5. Key / CSP Storage

Symmetric keys, HMAC keys, public and private keys are provided to the module by the calling application via API input parameters, and are destroyed by the module when invoking the appropriate API function calls.

The module does not perform persistent storage of keys. The keys and CSPs are stored as plaintext in the RAM. The only exception is the HMAC key used for the integrity test, which is stored in the module and relies on the operating system for protection.

6.6. Key / CSP Zeroization

The memory occupied by keys is allocated by regular memory allocation operating system calls. The application is responsible for calling the appropriate destruction functions. The destruction functions overwrite the memory occupied by keys with “zeros” and deallocates the memory with the regular memory deallocation operating system call. In case of abnormal termination, or swap in/out of a physical memory page of a process, the keys in physical memory are overwritten by the Linux kernel before the physical memory is allocated to another process.

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

The test platforms listed in Table 3 - Tested Platforms 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 devices are designed to provide reasonable protection against harmful interference when the devices are operated in a commercial environment. They shall be installed and used in accordance with the instruction manual.

8. Self-Tests

FIPS 140-2 requires that the module perform power-up tests to ensure the integrity of the module and the correctness of the cryptographic functionality at start up. In addition, some functions require continuous testing of the cryptographic functionality, such as the asymmetric key generation. If any self-test fails, the module returns an error code and enters the error state. No data output or cryptographic operations are allowed in error state.

8.1. Power-Up Tests

The module performs power-up tests when the module 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 by the calling application until the power-up tests are completed successfully.

The subsequent calls to the module will also fail - thus no further cryptographic operations are possible. If the power-up tests complete successfully, the module will return 1 in the return code and will accept cryptographic operation service requests.

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 software component of the 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 self-tests on all FIPS-Approved cryptographic algorithms supported in the Approved mode of operation, using the Known Answer Tests (KAT) and Pair-wise Consistency Tests (PCT) shown in the following table:

Algorithm	Power-Up Tests
AES	• AES-128: ECB, CFB, OFB • AES-192: ECB, CCM • AES-256: ECB, CMAC
Triple DES	• ECB, CBC, CTR, CFB, CMAC
HMAC	• SHA-1, SHA-224, SHA-256, SHA-384, SHA-512 • SHA3-224, SHA3-256, SHA3-384, SHA3-512
DSA	• PCT DSA with L=2048, N=256 and SHA-256
ECDSA	• PCT ECDSA with P-256 and SHA-256

Algorithm	Power-Up Tests
RSA	• KAT RSA with 2048-bit key, PKCS#1 v1.5 scheme and SHA-256, signature generation • KAT RSA with 2048-bit key, PKCS#1 v1.5 scheme and SHA-256, signature verification
DRBG	• SHA-256 with PR • HMAC-SHA-256 with PR • AES-128 with PR • SHA-256 no PR • HMAC-SHA256 no PR • AES-128 no PR • SHA1 no PR

Table 11- Self-Tests

For the KAT, the module calculates the result and compares it with the known value. If the answer does not match the known answer, the KAT is failed and the module enters the Error state.

For the PCT, if the signature generation or verification fails, the module enters the Error state. As described in section 3.3, only one AES or SHA implementation is available at run-time.

The KATs cover the different cryptographic implementations available in the operating environment.

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 performs conditional tests on the cryptographic algorithms, using the Pair-wise Consistency Tests (PCT) and Continuous Random Number Generator Test (CRNGT), shown in the following table:

Algorithm	Conditional Test
DSA key generation	• PCT using SHA-256, signature generation and verification.
ECDSA key generation	• PCT using SHA-256, signature generation and verification.
RSA key generation	• PCT using SHA-256, signature generation and verification. • PCT for encryption and decryption.
DRBG	• CRNGT is not required per IG 9.8

Table 12 - Conditional Tests

8.4. Error state

The Module enters Error state with error message, on failure of POST or conditional test. In Error state, all data output is inhibited and no cryptographic operation is allowed. The error can be recovered by calling gcry_control(GCRYCTL_SELFTEST) function that reruns the POST. The module enters Fatal Error state when random numbers are requested in error state or when requesting cipher operations on deallocated handle. In Fatal Error state the module is aborted and is not available for use. The module needs to be reloaded in order to recover from Fatal Error state.

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 Ubuntu packages contain the FIPS validated module:

Processor Architecture	Ubuntu packages
x86_64	libgcrypt20_1.8.1-4ubuntu1.fips.2.4_amd64.deb libgcrypt20-hmac-1.8.1-4ubuntu1.fips.2.4_amd64.deb

Table 13 – Ubuntu packages

The libgcrypt2-doc_1.8.1.ubuntu2.fips2.2.deb Ubuntu package contains the man pages for the module.

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

Table 14 – Prerequisite Ubuntu packages

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

- create the file /etc/default/grub.d/99-fips.cfg with the content: GRUB_CMDLINE_LINUX_DEFAULT="$GRUB_CMDLINE_LINUX_DEFAULT fips=1".

(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.

(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 UUID shall be added in the /etc/default/grub.d/99-fips.cfg. For example:

```
GRUB_CMDLINE_LINUX_DEFAULT="$GRUB_CMDLINE_LINUX_DEFAULT fips=1 bootdev=UUID=Insert boot UUID"
```

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, and configuration to access the PPA is complete, the Crypto Officer can install the Ubuntu packages containing the module listed in Table 13 using the Advanced Package Tool (APT) with the following command line:

```
$ sudo apt-get install libcgrypt20 libgcrypt20-hmac
```

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

9.2. User 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 section 3.2 Services). In addition, key sizes must comply with [SP800-131A].

Applications using libgcrypt need to call gcry_control(GCRYCTL_INITIALIZATION_FINISHED, 0) after initialization is done: that ensures that the DRBG is properly seeded, among others. gcry_control(GCRYCTL_TERM_SECMEM) needs to be called before the process is terminated. The function gcry_set_allocation_handler() may not be used.

The user must not call malloc/free to create/release space for keys, let libgcrypt manage space for keys, which will ensure that the key memory is overwritten before it is released. See the documentation file doc/gcrypt.texi within the source code tree for complete instructions for use.

The information pages are included within the developer package. The user can find the documentation at the following location after having installed the documentation package:

- /usr/share/info/gcrypt.info.gz.

9.2.1. AES XTS

The AES algorithm in XTS mode can be only used for the cryptographic protection of data on storage devices, as specified in [SP800-38E]. The length of a single data unit encrypted with the XTS-AES shall not exceed 2^{20} AES blocks that is 16MB of data.

To meet the requirement in [FIPS140-2_IG] A.9, the module implements a check to ensure that the two AES keys used in XTS-AES algorithm are not identical.

Note: AES-XTS shall be used with 128 and 256-bit keys only. AES-XTS with 192-bit keys is not an Approved service.

9.2.2. Triple-DES

[SP800-67] imposes a restriction on the number of 64-bit block encryptions performed under the same three-key Triple-DES key. The module cannot perform more than 2^{16} 64-bit data block encryptions. The user is responsible for ensuring the module's compliance with this requirement.

9.2.3. PBKDF

Keys derived from passwords or passphrases are only used for data at rest. The length of the salt should be at least 128 bits and the length of the password or passphrase should be at least 20 characters, which provides the probability of guessing this password or passphrase to be (1/10) ^ 20 assuming a scenario where all characters are digits. The caller shall observe all requirements and should consider all recommendations specified in SP800-132 with respect to the strength of the generated key, including the quality of the password and the quality of the salt.

10. Mitigation of Other Attacks

libgcrypt uses a blinding technique for RSA decryption to mitigate real world timing attacks over a network: Instead of using the RSA decryption directly, a blinded value (y = x·re mod n) is decrypted and the unblinded value (x' = y'·r-1 mod n) returned. The blinding value “r” is a random value with the size of the modulus “n” and generated with `GCRY_WEAK_RANDOM' random level.

Weak Triple-DES keys are detected as follows:

In DES there are 64 known keys which are weak because they produce only one, two, or four different subkeys in the subkey scheduling process. The keys in this table have all their parity bits cleared.

```
static byte weak_keys[64][8] =
{
    { 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00 }, /*w weak keys*/
    { 0x00, 0x00, 0x1e, 0x1e, 0x00, 0x00, 0x0e, 0x0e },
    { 0x00, 0x00, 0xe0, 0xe0, 0x00, 0x00, 0xf0, 0xf0 },
    { 0x00, 0x00, 0xfe, 0xfe, 0x00, 0x00, 0xfe, 0xfe },
    { 0x00, 0x1e, 0x00, 0x1e, 0x00, 0x0e, 0x00, 0x0e }, /*sw semi-weak keys*/
    { 0x00, 0x1e, 0x1e, 0x00, 0x00, 0x0e, 0x0e, 0x00 },
    { 0x00, 0x1e, 0xe0, 0xfe, 0x00, 0x0e, 0xf0, 0xfe },
    { 0x00, 0x1e, 0xfe, 0xe0, 0x00, 0x0e, 0xfe, 0xf0 },
    { 0x00, 0xe0, 0x00, 0xe0, 0x00, 0xf0, 0x00, 0xf0 }, /*sw*/
    { 0x00, 0xe0, 0x1e, 0xfe, 0x00, 0xf0, 0x0e, 0xfe },
    { 0x00, 0xe0, 0xe0, 0x00, 0x00, 0xf0, 0xf0, 0x00 },
    { 0x00, 0xe0, 0xfe, 0x1e, 0x00, 0xf0, 0xfe, 0x0e },
    { 0x00, 0xfe, 0x00, 0xfe, 0x00, 0xfe, 0x00, 0xfe }, /*sw*/
    { 0x00, 0xfe, 0x1e, 0xe0, 0x00, 0xfe, 0x0e, 0xf0 },
    { 0x00, 0xfe, 0xe0, 0x1e, 0x00, 0xfe, 0xf0, 0x0e },
    { 0x00, 0xfe, 0xfe, 0x00, 0x00, 0xfe, 0xfe, 0x00 },
    { 0x1e, 0x00, 0x00, 0x1e, 0x0e, 0x00, 0x00, 0x0e },
    { 0x1e, 0x00, 0x1e, 0x00, 0x0e, 0x00, 0x0e, 0x00 }, /*sw*/
    { 0x1e, 0x00, 0xe0, 0xfe, 0x0e, 0x00, 0xf0, 0xfe },
    { 0x1e, 0x00, 0xfe, 0xe0, 0x0e, 0x00, 0xfe, 0xf0 },
    { 0x1e, 0x1e, 0x00, 0x00, 0x0e, 0x0e, 0x00, 0x00 },
    { 0x1e, 0x1e, 0x1e, 0x1e, 0x0e, 0x0e, 0x0e, 0x0e }, /*w*/
    { 0x1e, 0x1e, 0xe0, 0xe0, 0x0e, 0x0e, 0xf0, 0xf0 },
    { 0x1e, 0x1e, 0xfe, 0xfe, 0x0e, 0x0e, 0xfe, 0xfe },
```

{ 0x1e, 0xe0, 0x00, 0xfe, 0x0e, 0xf0, 0x00, 0xfe },
{ 0x1e, 0xe0, 0x1e, 0xe0, 0x0e, 0xf0, 0x0e, 0xf0 }, /***sw***/
{ 0x1e, 0xe0, 0xe0, 0x1e, 0x0e, 0xf0, 0xf0, 0x0e },
{ 0x1e, 0xe0, 0xfe, 0x00, 0x0e, 0xf0, 0xfe, 0x00 },
{ 0x1e, 0xfe, 0x00, 0xe0, 0x0e, 0xfe, 0x00, 0xf0 },
{ 0x1e, 0xfe, 0x1e, 0xfe, 0x0e, 0xfe, 0x0e, 0xfe }, /***sw***/
{ 0x1e, 0xfe, 0xe0, 0x00, 0x0e, 0xfe, 0xf0, 0x00 },
{ 0x1e, 0xfe, 0xfe, 0x1e, 0x0e, 0xfe, 0xfe, 0x0e },
{ 0xe0, 0x00, 0x00, 0xe0, 0xf0, 0x00, 0x00, 0xf0 },
{ 0xe0, 0x00, 0x1e, 0xfe, 0xf0, 0x00, 0x0e, 0xfe },
{ 0xe0, 0x00, 0xe0, 0x00, 0xf0, 0x00, 0xf0, 0x00 }, /***sw***/
{ 0xe0, 0x00, 0xfe, 0x1e, 0xf0, 0x00, 0xfe, 0x0e },
{ 0xe0, 0x1e, 0x00, 0xfe, 0xf0, 0x0e, 0x00, 0xfe },
{ 0xe0, 0x1e, 0x1e, 0xe0, 0xf0, 0x0e, 0x0e, 0xf0 },
{ 0xe0, 0x1e, 0xe0, 0x1e, 0xf0, 0x0e, 0xf0, 0x0e }, /***sw***/
{ 0xe0, 0x1e, 0xfe, 0x00, 0xf0, 0x0e, 0xfe, 0x00 },
{ 0xe0, 0xe0, 0x00, 0x00, 0xf0, 0xf0, 0x00, 0x00 },
{ 0xe0, 0xe0, 0x1e, 0x1e, 0xf0, 0xf0, 0x0e, 0x0e },
{ 0xe0, 0xe0, 0xe0, 0xe0, 0xf0, 0xf0, 0xf0, 0xf0 }, /***w***/
{ 0xe0, 0xe0, 0xfe, 0xfe, 0xf0, 0xf0, 0xfe, 0xfe },
{ 0xe0, 0xfe, 0x00, 0x1e, 0xf0, 0xfe, 0x00, 0x0e },
{ 0xe0, 0xfe, 0x1e, 0x00, 0xf0, 0xfe, 0x0e, 0x00 },
{ 0xe0, 0xfe, 0xe0, 0xfe, 0xf0, 0xfe, 0xf0, 0xfe }, /***sw***/
{ 0xe0, 0xfe, 0xfe, 0xe0, 0xf0, 0xfe, 0xfe, 0xf0 },
{ 0xfe, 0x00, 0x00, 0xfe, 0xfe, 0x00, 0x00, 0xfe },
{ 0xfe, 0x00, 0x1e, 0xe0, 0xfe, 0x00, 0x0e, 0xf0 },
{ 0xfe, 0x00, 0xe0, 0x1e, 0xfe, 0x00, 0xf0, 0x0e },
{ 0xfe, 0x00, 0xfe, 0x00, 0xfe, 0x00, 0xfe, 0x00 }, /***sw***/
{ 0xfe, 0x1e, 0x00, 0xe0, 0xfe, 0x0e, 0x00, 0xf0 },
{ 0xfe, 0x1e, 0x1e, 0xfe, 0xfe, 0x0e, 0x0e, 0xfe },
{ 0xfe, 0x1e, 0xe0, 0x00, 0xfe, 0x0e, 0xf0, 0x00 },
{ 0xfe, 0x1e, 0xfe, 0x1e, 0xfe, 0x0e, 0xfe, 0x0e }, /***sw***/
{ 0xfe, 0xe0, 0x00, 0x1e, 0xfe, 0xf0, 0x00, 0x0e },

```
{ 0xfe, 0xe0, 0x1e, 0x00, 0xfe, 0xf0, 0x0e, 0x00 },
{ 0xfe, 0xe0, 0xe0, 0xfe, 0xfe, 0xf0, 0xf0, 0xfe },
{ 0xfe, 0xe0, 0xfe, 0xe0, 0xfe, 0xf0, 0xfe, 0xf0 }, /*sw*/
{ 0xfe, 0xfe, 0x00, 0x00, 0xfe, 0xfe, 0x00, 0x00 },
{ 0xfe, 0xfe, 0x1e, 0x1e, 0xfe, 0xfe, 0x0e, 0x0e },
{ 0xfe, 0xfe, 0xe0, 0xe0, 0xfe, 0xfe, 0xf0, 0xf0 },
{ 0xfe, 0xfe, 0xfe, 0xfe, 0xfe, 0xfe, 0xfe, 0xfe } /*w*/ };
```

Appendix A. Glossary and Abbreviations

AES	Advanced Encryption Standard
AES-NI	Advanced Encryption Standard New Instructions
API	Application Program Interface
APT	Advanced Package Tool
CAVP	Cryptographic Algorithm Validation Program
CAVS	Cryptographic Algorithm Validation System
CBC	Cipher Block Chaining
CCM	Counter with Cipher Block Chaining-Message Authentication Code
CFB	Cipher Feedback
CLMUL	Carry-less Multiplication
CMAC	Cipher-based Message Authentication Code
CMVP	Cryptographic Module Validation Program
CPACF	CP Assist for Cryptographic Function
CRNGT	Continuous Random Number Generator Test
CSP	Critical Security Parameter
CTR	Counter Mode
DES	Data Encryption Standard
DF	Derivation Function
DSA	Digital Signature Algorithm
DTLS	Datagram Transport Layer Security
DRBG	Deterministic Random Bit Generator
ECB	Electronic Code Book
ECC	Elliptic Curve Cryptography
EMI/EMC	Electromagnetic Interference/Electromagnetic Compatibility
FCC	Federal Communications Commission
FFC	Finite Field Cryptography
FIPS	Federal Information Processing Standards Publication
GCM	Galois Counter Mode
GPC	General Purpose Computer
HMAC	Hash Message Authentication Code
IG	Implementation Guidance

KAS	Key Agreement Schema
KAT	Known Answer Test
KDF	Key Derivation Function
KW	Key Wrap
LPAR	Logical Partitions
MAC	Message Authentication Code
NIST	National Institute of Science and Technology
NDRNG	Non-Deterministic Random Number Generator
OFB	Output Feedback
PCT	Pair-wise Consistency Test
PPA	Personal Package Archive
PR	Prediction Resistance
PRNG	Pseudo-Random Number Generator
PSS	Probabilistic Signature Scheme
RSA	Rivest, Shamir, Addleman
SHA	Secure Hash Algorithm
SHS	Secure Hash Standard
SSSE3	Supplemental Streaming SIMD Extensions 3
TLS	Transport Layer Security
XTS	XEX-based Tweaked-codebook mode with ciphertext Stealing

Appendix B. 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

FIPS180-4 **Secure Hash Standard (SHS)**
March 2012
http://nvlpubs.nist.gov/nistpubs/FIPS/NIST.FIPS.180-4.pdf

FIPS186-4 **Digital Signature Standard (DSS)**
July 2013
http://nvlpubs.nist.gov/nistpubs/FIPS/NIST.FIPS.186-4.pdf

FIPS197 **Advanced Encryption Standard**
November 2001
http://csrc.nist.gov/publications/fips/fips197/fips-197.pdf

FIPS198-1 **The Keyed Hash Message Authentication Code (HMAC)**
July 2008
http://csrc.nist.gov/publications/fips/fips198-1/FIPS-198-1_final.pdf

FIPS202 **SHA-3 Standard: Permutation-Based Hash and Extendable-Output Functions**
August 2015
https://nvlpubs.nist.gov/nistpubs/FIPS/NIST.FIPS.202.pdf

LMAN **Linux Man Pages**
http://man7.org/linux/man-pages/

PKCS#1 **Public Key Cryptography Standards (PKCS) #1: RSA Cryptography Specifications Version 2.1**
February 2003
http://www.ietf.org/rfc/rfc3447.txt

RFC2246 **The TLS Protocol Version 1.0**
January 1999
https://www.ietf.org/rfc/rfc2246.txt

RFC3268 **Advanced Encryption Standard (AES) Ciphersuites for Transport Layer Security (TLS)**
June 2002
https://www.ietf.org/rfc/rfc3268.txt

RFC4279 **Pre-Shared Key Ciphersuites for Transport Layer Security (TLS)**
December 2005
https://www.ietf.org/rfc/rfc4279.txt

RFC4346 **The Transport Layer Security (TLS) Protocol Version 1.1**
April 2006
https://www.ietf.org/rfc/rfc4346.txt

RFC4492 **Elliptic Curve Cryptography (ECC) Cipher Suites for Transport Layer Security (TLS)**
May 2006
https://www.ietf.org/rfc/rfc4492.txt

RFC5116 **An Interface and Algorithms for Authenticated Encryption**
January 2008
https://www.ietf.org/rfc/rfc5116.txt

RFC5246 **The Transport Layer Security (TLS) Protocol Version 1.2**
August 2008
https://tools.ietf.org/html/rfc5246.txt

RFC5288 **AES Galois Counter Mode (GCM) Cipher Suites for TLS**
August 2008
https://tools.ietf.org/html/rfc5288.txt

RFC5487 **Pre-Shared Key Cipher Suites for TLS with SHA-256/384 and AES Galois Counter Mode**
March 2009
https://tools.ietf.org/html/rfc5487.txt

RFC5489 **ECDHE_PSK Cipher Suites for Transport Layer Security (TLS)**
March 2009
https://tools.ietf.org/html/rfc5489.txt

RFC6655 **AES-CCM Cipher Suites for Transport Layer Security (TLS)**
July 2012
https://tools.ietf.org/html/rfc6655.txt

RFC7251 **AES-CCM Elliptic Curve Cryptography (ECC) Cipher Suites for TLS**
June 2014
https://tools.ietf.org/html/rfc7251.txt

SP800-38A **NIST Special Publication 800-38A - Recommendation for Block Cipher Modes of Operation Methods and Techniques**
December 2001
http://csrc.nist.gov/publications/nistpubs/800-38a/sp800-38a.pdf

SP800-38B **NIST Special Publication 800-38B - Recommendation for Block Cipher Modes of Operation: The CMAC Mode for Authentication**
May 2005
http://csrc.nist.gov/publications/nistpubs/800-38B/SP_800-38B.pdf

SP800-38C **NIST Special Publication 800-38C - Recommendation for Block Cipher Modes of Operation: the CCM Mode for Authentication and Confidentiality**
May 2004
http://nvlpubs.nist.gov/nistpubs/Legacy/SP/nistspecialpublication800-38c.pdf

SP800-38D **NIST Special Publication 800-38D - Recommendation for Block Cipher Modes of Operation: Galois/Counter Mode (GCM) and GMAC**
November 2007
http://csrc.nist.gov/publications/nistpubs/800-38D/SP-800-38D.pdf

SP800-38E **NIST Special Publication 800-38E - Recommendation for Block Cipher Modes of Operation: The XTS AES Mode for Confidentiality on Storage Devices**
January 2010
http://csrc.nist.gov/publications/nistpubs/800-38E/nist-sp-800-38E.pdf

SP800-38F **NIST Special Publication 800-38F - Recommendation for Block Cipher Modes of Operation: Methods for Key Wrapping**
December 2012
http://nvlpubs.nist.gov/nistpubs/SpecialPublications/NIST.SP.800-38F.pdf

SP800-52 **NIST Special Publication 800-52 Revision 1 - Guidelines for the Selection, Configuration, and Use of Transport Layer Security (TLS) Implementations**
April 2014
http://nvlpubs.nist.gov/nistpubs/SpecialPublications/NIST.SP.800-52r1.pdf

SP800-56A **NIST Special Publication 800-56A Revision 2 - Recommendation for Pair Wise Key Establishment Schemes Using Discrete Logarithm Cryptography**
May 2013
http://nvlpubs.nist.gov/nistpubs/SpecialPublications/NIST.SP.800 56Ar2.pdf

SP800-56B **NIST Special Publication 800-56B Revision 1 - Recommendation for Pair-Wise Key-Establishment Schemes Using Integer Factorization Cryptography**
September 2014
https://nvlpubs.nist.gov/nistpubs/SpecialPublications/NIST.SP.800-56Br1.pdf

SP800-57 **NIST Special Publication 800-57 Part 1 Revision 4 - Recommendation for Key Management Part 1: General**
January 2016
http://nvlpubs.nist.gov/nistpubs/SpecialPublications/NIST.SP.800-57pt1r4.pdf

SP800-67 **NIST Special Publication 800-67 Revision 1 - Recommendation for the Triple Data Encryption Algorithm (TDEA) Block Cipher**
January 2012
http://csrc.nist.gov/publications/nistpubs/800-67-Rev1/SP-800-67-Rev1.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