FIPS 140-2 Non-Proprietary Security Policy

CryptoComply for Libgcrypt

Software Version 5.0

Document Version 1.0

December 1, 2021

SafeLogic Inc.
530 Lytton Ave., Suite 200
Palo Alto, CA 94301
www.safelogic.com

Overview

This document provides a non-proprietary FIPS 140-2 Security Policy for CryptoComply for Libgcrypt.

SafeLogic's CryptoComply for Libgcrypt is designed to provide FIPS 140-2 validated cryptographic functionality and is available for licensing. For more information, visit www.safelogic.com/software.

Table of Contents

List of Tables

List of Figures

1 Introduction

1.1 About FIPS 140

Federal Information Processing Standards Publication 140-2 — Security Requirements for Cryptographic Modules specifies requirements for cryptographic modules to be deployed in a Sensitive but Unclassified environment. The National Institute of Standards and Technology (NIST) and Canadian Centre for Cyber Security (CCCS) Cryptographic Module Validation Program (CMVP) run the FIPS 140 program. The National Voluntary Laboratory Accreditation Program (NVLAP) accredits independent testing labs to perform FIPS 140 testing; the CMVP validates modules meeting FIPS 140 validation. *Validated* is the term given to a module that is documented and tested against the FIPS 140 criteria.

More information is available on the CMVP website at https://csrc.nist.gov/projects/cryptographic-module-validation-program.

1.2 About this Document

This non-proprietary Cryptographic Module Security Policy for CryptoComply for Libgcrypt from SafeLogic Inc. ("SafeLogic") provides an overview of the product and a high-level description of how it meets the overall Level 1 security requirements of FIPS 140-2.

CryptoComply for Libgcrypt may also be referred to as the "module" in this document.

1.3 External Resources

The SafeLogic website (www.safelogic.com) contains information on SafeLogic services and products. The Cryptographic Module Validation Program website contains links to the FIPS 140-2 certificate and SafeLogic contact information.

1.4 Notices

2 CryptoComply for Libgcrypt

2.1 Cryptographic Module Specification

CryptoComply for Libgcrypt is a software library designed to provide FIPS 140-2 validated cryptographic functionality and is available for licensing.

The module's software version is 5.0. The module provides cryptographic services to applications running in the user space of the underlying operating system through an application program interface (API).

The module is a software module that relies on the physical characteristics of the host platform. The module's physical boundary is defined by the enclosure of the host platform, which is the General Purpose Device that the module is installed on. For the purposes of FIPS 140-2 validation, the module's embodiment type is defined as multi-chip standalone.

The module's logical cryptographic boundary is the shared library file and its integrity check file as listed below:

- `libgcrypt.so.11`
- `libgcrypt.so.11.hmac`

2.1.1 Validation Level Detail

The following table lists the module's level of validation for each area in FIPS 140-2:

Table 1 - Validation Level by FIPS 140-2 Section

FIPS 140-2 Section Title	Validation Level
Cryptographic Module Specification	1
Cryptographic Module Ports and Interfaces	1
Roles, Services, and Authentication	1
Finite State Model	1
Physical Security	N/A
Operational Environment	1
Cryptographic Key Management	1
Electromagnetic Interference/Electromagnetic Compatibility	1
Self-Tests	1
Design Assurance	3
Mitigation of Other Attacks	1

2.1.2 Modes of Operation

The module supports two modes of operation: FIPS Approved mode and non-Approved mode.

The mode of operation in which the module is operating can be determined by:

- If the file /proc/sys/crypto/fips_enabled exists and contains a numeric value other than 0, the module is put into FIPS Approved mode at initialization time.
- If the file /etc/gcrypt/fips_enabled exists, the module is put into FIPS Approved mode at initialization time. This filename does not depend on any configuration options.

The module turns to the FIPS Approved mode after the initialization and the power-on self-tests have completed successfully.

Using a non-Approved algorithm specified in Table 4 will result in the module implicitly entering the non-Approved mode of operation.

The services available in FIPS Approved mode can be found in Section 2.1.3, Table 2. The non-Approved but allowed services can be found in Section 2.1.4, Table 3. The services available in non-Approved mode can be found in Section 2.1.5, Table 4.

2.1.3 Approved Cryptographic Algorithms

The module's cryptographic algorithm implementations have received the following certificate numbers from the Cryptographic Algorithm Validation Program (CAVP).

Table 2 - FIPS Approved Algorithm Certificates

CAVP Cert.	Algorithm	Standard	Mode/Method	Key Lengths, Curves or Moduli	Use
C709	**AES**	FIPS 197 SP 800-38A	CBC, CFB128, CTR, ECB, OFB	128, 192, 256	Encryption, Decryption
Vendor affirmed	**CKG**	SP 800-133	Asymmetric seeds are the unmodified output of the SP 800-90A DRBG (ref. SP section 2.2.3) per IG D.12		Key Generation
C709	**DRBG**	SP 800-90A	CTR DRBG using AES 128/192/256 with derivation function (with and without prediction resistance) Hash DRBG using SHA-1/256/384/512 (with and without predication resistance) HMAC DRBG using HMAC-SHA-1/256/384/512 (with and without predication resistance)	112, 128, 192, 256	Random Bit Generation

CAVP Cert.	Algorithm	Standard	Mode/Method	Key Lengths, Curves or Moduli	Use
C709	**DSA**	FIPS 186-4	Key Generation: (2048, 224), (2048, 256), (3072, 256) PQGGen, Signature Generation: (2048, 224) (SHA-224) (2048, 256) (SHA-256) (3072, 256) (SHA-256) Signature Verification: (1024, 160) (SHA-1) (2048, 224) (SHA-224) (2048, 256) (SHA-256) (3072, 256) (SHA-256)		Digital Signature Services
C709	**HMAC**	FIPS 198-1	HMAC-SHA-1 HMAC-SHA-224 HMAC-SHA-256 HMAC-SHA-384 HMAC-SHA-512	KS<BS, KS=BS, KS>BS	HMAC Generation, Authentication
C709	**RSA**	FIPS 186-4 PKCS #1 v2.1 (PSS and PKCS1.5)	Key Generation: 2048, 3072 bits (primality per Table C.3) Signature Generation (PKCS #1 v1.5 and PKCSPSS): 2048, 3072 bits (SHA-224, SHA-256, SHA-384, SHA-512) Signature Verification (PKCS #1 v1.5 and PKCSPSS): 1024, 2048, 3072 bits (SHA-1, SHA-224, SHA-256, SHA-384, SHA-512)		Digital Signature Services

CAVP Cert.	Algorithm	Standard	Mode/Method	Key Lengths, Curves or Moduli	Use
C709	**SHS**	FIPS 180-4	SHA-1 SHA-224 SHA-256 SHA-384 SHA-512		Digital Signature Services, non-Digital Signature Applications
C709	**Triple-DES**	SP 800-67	CBC, CFB64, CTR, ECB, OFB	2-key[1], 3-key[2]	Encryption, Decryption

[1] Two-key Triple-DES only for legacy decryption
[2] Each 3-key Triple-DES key shall not be used to encrypt more than 2^{16} 64-bit blocks of data, refer to Security Policy Section 3.2

2.1.4 Non-Approved but Allowed Cryptographic Algorithms

The module supports the following FIPS 140-2 non-Approved but allowed algorithms that may be used in the FIPS Approved mode of operation.

Table 3 - Non-Approved but Allowed Cryptographic Algorithms

Algorithm	Use
NDRNG	Generation of random numbers
RSA key wrapping	[IG D.9] RSA may be used by a calling application as part of a key encapsulation scheme. Key establishment methodology provides between 112 and 256 bits of encryption strength. Key sizes ≥ 2048 bits

2.1.5 Non-Approved Mode of Operation

The module supports a non-Approved mode of operation. The algorithms listed in this section are not to be used by the operator in the FIPS Approved mode of operation. Use of these algorithms will put the module in the non-Approved mode of operation implicitly.

Table 4 - Non-Approved Cryptographic Functions for Use in Non-Approved mode Only

Algorithm	Use
ARC4	Encryption and decryption (stream cipher)
Blowfish	Encryption and decryption
Camellia	Encryption and decryption
CAST5	Encryption and decryption
CRC32	Cyclic redundancy code
CSPRNG	Cryptographically Secure Pseudorandom Number Generator
DES	Encryption and decryption (key size of 56 bits)
DSA	Parameter verification, Parameter/Key generation/Signature generation with keys not listed in Table 2
El Gamal	Key pair generation, encryption and decryption, signature generation, signature verification
Gost	28147 encryption
	R 34.11-94 hash
	R 34.11-2012 (Stribog) hash
IDEA	Encryption and decryption
MD4	Hashing
MD5	Hashing
OpenPGP S2K Salted and Iterated/salted	Password based key derivation compliant with OpenPGP (RFC 4880)
RC2	Encryption and decryption based on RFC 2268
RIPEMD 160	Hashing

RSA	Key generation, signature generation, key wrapping: keys < 2048 bits Signature verification: keys < 1024 bits
SHA-1	Used for signature generation
SEED	Encryption and decryption
Serpent	Encryption and decryption
Tiger	Hashing
Twofish	Encryption and decryption
2-key Triple-DES	Encryption
Whirlpool	Hashing

2.2 Critical Security Parameters and Public Keys

2.2.1 Critical Security Parameters

The table below provides a complete list of Critical Security Parameters (CSPs) used within the module.

Table 5 - Critical Security Parameters

CSP	Generation	Storage	Entry/Output	Zeroization
AES Keys[19]	N/A The key is passed into the module via API input parameter	Application memory	API input/output parameters and return values within the physical boundaries of the module	Automatically zeroized when freeing the cipher handler by calling gcry_cipher_close ()
Triple-DES Keys	N/A The key is passed into the module via API input parameter	Application memory	API input/output parameters and return values within the physical boundaries of the module	Automatically zeroized when freeing the cipher handler by calling gcry_cipher_close ()
DSA Private Keys	Use of the module's SP 800-90A DRBG and the module's 186-4 DSA key generation mechanism	Application memory	API input/output parameters and return values within the physical boundaries of the module	Automatically zeroized when freeing the cipher handler by calling gcry_sexp_release ()
RSA Private Keys	Use of the module's SP 800-90A DRBG and the module's 186-4 RSA key generation mechanism	Application memory	API input/output parameters and return values within the physical boundaries of the module	Automatically zeroized when freeing the cipher handler by calling gcry_sexp_release ()

[19] The AES-GCM key and IV are generated randomly per IG A.5, and the Initialization Vector (IV) is a minimum of 96 bits. In the event module power is lost and restored, the consuming application must ensure that any of its AES-GCM keys used for encryption or decryption are re-distributed.

CSP	Generation	Storage	Entry/Output	Zeroization
SP 800-90A DRBG Entropy string	N/A obtained from NDRNG	Application memory	N/A	Automatically zeroized when freeing the cipher handler by calling gcry_drbg_uninstantiate()
SP 800-90A DRBG Seed and internal state values (C and V values)	Based on entropy string as defined in SP 800-90A	Application memory	N/A	Automatically zeroized when freeing the cipher handler by calling gcry_drbg_uninstantiate()
HMAC Keys	N/A The key is passed into the module via API input parameter	Application memory	API input/output parameters and return values within the physical boundaries of the module	Automatically zeroized when freeing the cipher handler by calling gcry_md_close()

2.2.2 Random Number Generation

The module employs a Deterministic Random Bit Generator (DRBG) based on [SP800-90A] for the creation of key components of asymmetric keys and for random number generation. The DRBG is initialized during module initialization. The module loads by default the DRBG using HMAC_DRBG with SHA-256 and derivation function tests without prediction resistance.

The Module uses NDRNG from /dev/urandom as a source of entropy 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 130 bits of entropy to the DRBG. The module generates cryptographic keys whose strengths are modified by available entropy.

The module performs continuous tests on the output of the DRBG to ensure that consecutive random numbers do not repeat. The underlying operating system performs the continuous test on the NDRNG which is outside the module's logical boundary but inside the physical boundary.

2.2.3 Key Generation

The Key Generation methods implemented in the module for Approved services in FIPS mode are compliant with [SP 800-133].

For generating RSA and DSA keys the module implements asymmetric key generation services compliant with [FIPS186-4] and [SP800-90A]. A seed (i.e. the random value) used in asymmetric key generation is obtained from [SP800-90A] DRBG using unmodified output from the Approved DRBG. The module does not offer a dedicated service for generating keys for symmetric algorithms or for HMAC. In accordance with FIPS 140-2 IG D.12, the cryptographic module performs Cryptographic Key Generation (CKG) for asymmetric keys as per SP800-133 (vendor affirmed).

The module does not support manual key entry or intermediate key generation output. In addition, the module does not output keys outside its physical boundary. The keys can be entered or output from the module in plaintext form via API parameters, to and from the calling application only.

2.2.4 Key/CSP Storage

Public and private keys are provided to the module by the calling process, and are destroyed when released by the appropriate API function calls. The module does not perform persistent storage of any keys or CSPs.

2.2.5 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 provided in the module's API listed in Table 8. 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.

2.3 Module Interfaces

The figure below shows the module's physical and logical block diagram:

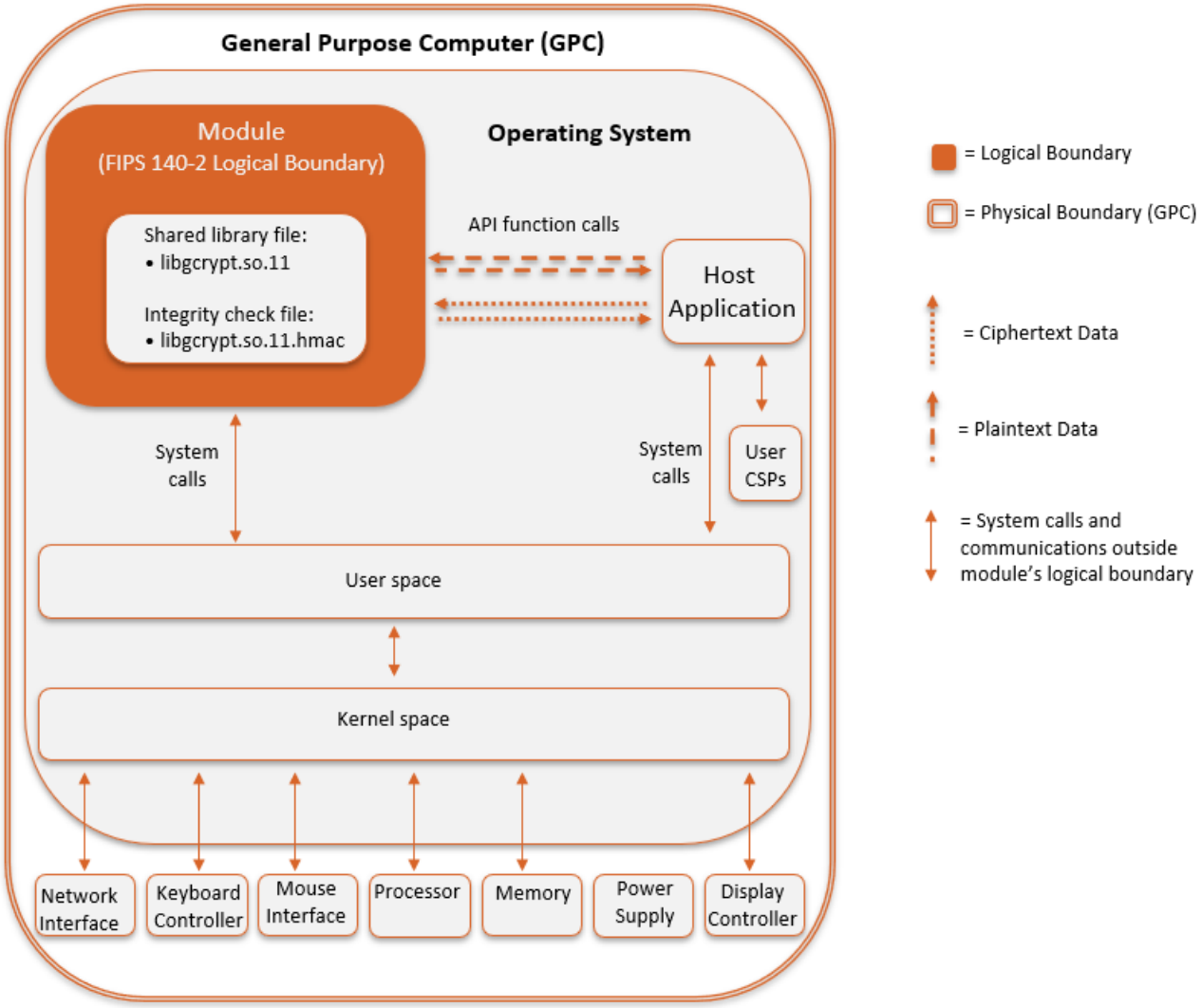

Figure 1 - Module Boundary and Interfaces Diagram

The module's physical boundary is the boundary of the General Purpose Computer (GPC) that the module is installed on, which includes a processor and memory. The interfaces (ports) for the physical boundary include the computer's network port, keyboard port, mouse port, power plug and display. When operational, the module does not transmit any information across these physical ports because it is a software cryptographic module. Therefore, the module's interfaces are purely logical.

The module's logical interfaces are provided through an Application Programming Interface (API) that a calling daemon can operate. The API itself defines the module's logical boundary, i.e. all access to the module is through this API. The API provides functions that may be called by an application (see Section

2.4.2 - Services for the list of available functions). The module distinguishes between logical interfaces by logically separating the information according to the defined API.

The API provided by the module is mapped onto the FIPS 140-2 logical interfaces, which relate to the module's callable interface as follows.

Table 6 - Logical Interface/Physical Interface Mapping

FIPS 140-2 Interface	Module Logical Interface	GPC Physical Interface
Data Input	API input parameters for data	Network Interface
Data Output	API output parameters for data	Network Interface
Control Input	API function calls, API input parameters, /proc/sys/crypto/fips_enabled control file, /etc/gcrypt/fips_enabled configuration file	Network Interface, Keyboard Interface, Mouse Interface
Status Output	API return codes, API output parameters for status	Display Controller, Network Interface
Power	None	Power Supply

As shown in

Figure 1 - Module Boundary and Interfaces Diagram and Table 8 - Services in FIPS Approved Mode, the output data path is provided by the data interfaces and is logically disconnected from processes performing key generation or zeroization. No key information will be output through the data output interface when the module zeroizes keys.

2.4 Roles, Services, and Authentication

2.4.1 Roles

The module supports two distinct operator roles, User and Crypto Officer (CO). The role is implicitly assumed based on the service requested. A user is considered the owner of the thread that instantiates the module and, therefore, only one concurrent user is allowed.

The module does not support a Maintenance role or bypass capability. The module does not support authentication.

Table 7 - Description of Roles

Role	Role Description	Authentication Type
CO	Performs module installation	N/A – Authentication is not a requirement for Level 1
User	Performs all services, except module installation	N/A – Authentication is not a requirement for Level 1

2.4.2 Services

2.4.2.1 FIPS Approved Mode Services

All services implemented by the module in FIPS Approved mode are listed in Table 8 - Services in FIPS Approved Mode. The table includes a description of each service, service availability to the Crypto Officer and User, and CSP access. Modes of CSP access shown in the table are defined as:

- **G** = Generate: The module generates the CSP.
- **R** = Read: The module reads the CSP. The read access is typically performed before the module uses the CSP.
- **E** = Execute: The module executes using the CSP.
- **W** = Write: The module writes the CSP. The write access is typically performed after a CSP is imported into the module, when the module generates a CSP, or when the module overwrites an existing CSP.
- **Z** = Zeroize: The module zeroizes the CSP.

Table 8 - Services in FIPS Approved Mode

Service	Description	CO	User	Key, CSP Access
Symmetric Encryption/ Decryption	Encrypts or decrypts a block of data using Triple-DES and/or AES		X	• R, W, E: AES, TDES Key
Get Key Length	cipher_get_keylen() function		X	• N/A
Get Block Length	cipher_get_blocksize() function		X	• N/A
Check Availability of Algorithm	cipher_get_blocksize() function		X	• N/A

Service	Description	CO	User	Key, CSP Access
Hash	Secure Hash Algorithm (SHS) hashes a block of data.		X	• N/A
Keyed Hash (HMAC)	HMAC function		X	• R, W, E: HMAC Key
Digital Signature Generation and Verification	Sign and verify operation		X	• R, W, E: RSA, DSA Key
Asymmetric Key Generation	Key pair generation for RSA or DSA		X	• R, W, E: RSA, DSA Key
Asymmetric Encryption/ Decryption	Encrypts or decrypts using Approved RSA key size		X	• R, W, E: RSA Key pair
Random Number Generation	Generate random numbers based on the SP 800-90A DRBG		X	• W, E: Entropy input string, seed and internal state
Self-tests	Performs power-up self-test on demand		X	• N/A
Zeroization	Zeroization performed by functions gcry_cipher_close(), gcry_sexp_release(), gcry_drbg_uninstantiate(), gcry_md_close()		X	• W: all CSPs
Show Status	Show status of the module state		X	• N/A
Module Installation	Install and configure the module	X		• N/A

2.4.2.2 Non-Approved Mode Services

In non-Approved mode, the module may use algorithms specified in Table 4. CSP/key separation is enforced between both modes. The services available in FIPS Approved mode can also be used in non-Approved mode. The module also provides the additional non-Approved services:

- Cyclic Redundancy Code (CRC 32)
- Password based KDF (OpenPGP Salted and Itereated/Salted, RFC 4880)

2.5 Physical Security

The module is a software-only module and does not have physical security mechanisms.

2.6 Operational Environment

The module operates in a modifiable operational environment under the FIPS 140-2 definitions.

The module runs on a GPC running one of the operating systems specified in this section, i.e. the approved operational environments.

The module was tested on the following platforms:

Table 9 - FIPS Tested Configurations

Operating System	Hardware	Processor
Oracle Linux 7.6 64-bit	Oracle Server X7-2	AMD® EPYC® 7551
Oracle Linux 7.6 64-bit	Oracle Server X7-2	Intel® Xeon® Silver 4114

FIPS 140-2 validation compliance is maintained for other compatible operating systems (in single user mode) where the module source code is unmodified, and the requirements outlined in NIST IG G.5 are met. No claim can be made as to the correct operation of the module or the security strengths of the generated keys when ported to an operational environment that is not listed on the validation certificate.

The GPC(s) used during testing met Federal Communications Commission (FCC) FCC Electromagnetic Interference (EMI) and Electromagnetic Compatibility (EMC) requirements for business use as defined by 47 Code of Federal Regulations, Part15, Subpart B.

2.6.1 Operational Environment 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, even when the application is serving multiple clients. In operational mode, the ptrace(2) system call, the debugger (gdb(1)) and strace(1) shall be not used.

2.7 Self-Tests

The module performs power-up tests at module initialization to ensure that the module is not corrupted and that the cryptographic algorithms work as expected. The self-tests are performed without any user intervention.

2.7.1 Power-Up Self-Tests

The module performs power-on self-tests (POST) automatically during loading of the module by making use of default entry point (DEP) without any user intervention. While the power-up tests are in progress, services are not available and input or output is not possible: the module is single-threaded and will not return to the calling application until the self-tests are completed successfully. If any of the self-test fails module enters Error state. If all the self-tests are successful, then the module is operational and cryptographic functionality is available for use.

The integrity of the module is verified by comparing the HMAC-SHA-256 value calculated at run time with the HMAC value stored in the HMAC file that was computed at build time.

Table 10 - Power-Up Self-Tests

Algorithm	Test
Module Integrity Test	HMAC-SHA-256 Integrity Test (Cert. #C709
AES (128, 192, 256)	KATs, encryption and decryption tested separately
DRBG	KATs for Hash, HMAC, and CTR The module supports health tests for the DRBG Instantiate, Generate, and Reseed Functions
DSA	PCT for signature generation/verification
HMAC	KATs for HMAC-SHA-1, HMAC-SHA-224, HMAC-SHA-256, HMAC-SHA-384, HMAC-SHA-512
RSA	KATs for signature generation/verification
SHS	KATs for SHA-1, SHA-224, SHA-256, SHA-384, SHA-512
Triple-DES	KATs, encryption and decryption tested separately

2.7.2 On Demand Self-Tests

The module provides the Self-Test service to perform self-tests on demand. This service performs the same cryptographic algorithm tests executed during power-up. To invoke the on-demand self-tests, the user can make a call to the gcry_control (GCRYCTL_SELFTEST) function. During the execution of the on-demand self-tests, services are not available and no data output or input is possible.

2.7.3 Conditional Self-Tests

If any of the conditional test fails, the module enters the Error state. The module implements the following conditional self-tests upon random number generation or key generation.

Table 11 - Conditional Self-Tests

Test Target	Description
DRBG	Continuous Random Number Generator Test (random/drbg.c:cdrbg_fips_continuous_test) The module also implements the SP 800-90A Section 11.3 Health Tests
DSA	Pairwise Consistency Test: signature generation and verification (cipher/dsa.c:test_keys())
RSA	Pairwise Consistency Test: signature generation and verification (cipher/rsa.c:test_keys())

2.7.4 Error State

The Module enters the Error state with an error message, on the failure of any POST or conditional test. In the Error state, all data output is inhibited and no cryptographic operations are 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.

2.8 Mitigation of Other Attacks

2.8.1 RSA blinding

The module 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.

2.8.2 Weak Triple-DES key detection

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*/ };
```

3 Security Rules and Guidance

3.1 Crypto Officer Guidance

The module is provided directly to solution developers and is not available for direct download to the general public. Only the compiled module is provided to solution developers. The module and its host application shall be installed on an operating system specified in Section 2.6 or one where portability is maintained.

Using a non-Approved algorithm specified in Table 4 will result in the module implicitly entering the non-Approved mode of operation.

The module needs to be put into FIPS Approved mode explicitly. If an application that uses the module for its cryptography is put into a chroot environment, the Crypto Officer must ensure one of the below methods is available to the module from within the chroot environment to ensure entry into FIPS Approved mode. Failure to do so will not allow the application to properly enter FIPS Approved mode.

If the file /proc/sys/crypto/fips_enabled exists and contains a numeric value other than 0, the module is put into FIPS Approved mode at initialization time. This is the mechanism recommended for ordinary use, activated by using the fips=1 option in the boot loader.

An application may request FIPS Approved mode using the control command gcry_control (GCRYCTL_FORCE_FIPS_MODE). This must be done prior to any initialization (i.e. before the gcry_check_version() function).

If the file /etc/gcrypt/fips_enabled exists, the module is put into FIPS mode at initialization time. This filename does not depend on any configuration options.

Once the module has been put into FIPS Approved mode, it is not possible to switch back to non-Approved mode without terminating the process first.

If the logging verbosity level of the module has been set to at least 2, the state transitions and the self-tests are logged.

3.2 User Guidance

Applications using the module need to call gcry_control(GCRYCTL_INITIALIZATION_FINISHED, O) 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 the module 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 locations after having installed the developer package:

/usr/share/info/gcrypt.info-1.gz

/usr/share/info/gcrypt.info-2.gz

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

3.2.1 Three-key Triple-DES

It is the calling application's responsibility to make sure that the three keys k1, k2 and k3 are independent. Two-key Triple-DES usage will bring the module into the non-Approved mode of operation implicitly.

Data encryption using the same three-key Triple-DES key shall not exceed 2^{16} Triple-DES (64-bit) blocks, in accordance to [SP800-67] and IG A.13 in [FIPS140-2-IG]. The user of the module is responsible for ensuring the module's compliance with this requirement.

4 References and Acronyms

4.1 References

Table 12 - References

Abbreviation	Full Specification Name
FIPS 140-2	*Security Requirements for Cryptographic modules, May 25, 2001*
FIPS 180-4	*Secure Hash Standard (SHS)*
FIPS 186-4	*Digital Signature Standard (DSS)*
FIPS 197	*Advanced Encryption Standard*
FIPS 198-1	*The Keyed-Hash Message Authentication Code (HMAC)*
IG	*Implementation Guidance for FIPS PUB 140-2 and the Cryptographic Module Validation Program*
PKCS#1 v2.1	*RSA Cryptography Standard*
SP 800-38A	*Recommendation for Block Cipher Modes of Operation: Three Variants of Ciphertext Stealing for CBC Mode*
SP 800-56B	*Recommendation for Pair-Wise Key Establishment Schemes Using Integer Factorization Cryptography*
SP 800-67	*Recommendation for the Triple Data Encryption Algorithm (TDEA) Block Cipher*
SP 800-89	*Recommendation for Obtaining Assurances for Digital Signature Applications*
SP 800-90A	*Recommendation for Random Number Generation Using Deterministic Random Bit Generators*

4.2 Acronyms

The following table defines acronyms found in this document:

Table 13 - Acronyms and Terms

Acronym	Term
AES	Advanced Encryption Standard
API	Application Programming Interface
CAVP	Cryptographic Algorithm Validation Program
CBC	Cipher-Block Chaining
CCCS	Canadian Centre for Cyber Security
CFB	Cipher Feedback Mode
CMAC	Cipher-based Message Authentication Code
CMVP	Cryptographic Module Validation Program
CO	Crypto Officer
CSP	Critical Security Parameter
CTR	Counter Mode
DES	Data Encryption Standard
DRAM	Dynamic Random Access Memory
DRBG	Deterministic Random Bit Generator
DSA	Digital Signature Algorithm
ECB	Electronic Code Book
EMC	Electromagnetic Compatibility
EMI	Electromagnetic Interference
FCC	Federal Communications Commission
FIPS	Federal Information Processing Standard
GCM	Galois/Counter Mode
GPC	General Purpose Computer
HMAC	(Keyed-) Hash Message Authentication Code
IG	Implementation Guidance
KAT	Known Answer Test
KDF	Key Derivation Function
MAC	Message Authentication Code
MD5	Message Digest Algorithm MD5
N/A	Not Applicable
NDRNG	Non Deterministic Random Number Generator
NIST	National Institute of Science and Technology
OFB	Output Feedback
OS	Operating System
PKCS	Public-Key Cryptography Standards
PSS	Probabilistic Signature Scheme
RIPEMD	RACE Integrity Primitives Evaluation Message Digest
RSA	Rivest, Shamir, and Adleman
SHA	Secure Hash Algorithm
SHS	Secure Hash Standard

Acronym	Term
TCBC	TDEA Cipher-Block Chaining
TCFB	TDEA Cipher Feedback Mode
TDES	Triple Data Encryption Standard
TECB	TDEA Electronic Codebook
TOFB	TDEA Output Feedback
USB	Universal Serial Bus