FIPS 140-2 Non-Proprietary Security Policy

CryptoComply for Libgcrypt

Software Version 4.0

Document Version 1.8

March 12, 2018

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

Abstract

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 https://www.safelogic.com/cryptocomply-for-libgcrypt/.

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 Communications Security Establishment Canada (CSE) Cryptographic Module Validation Program (CMVP) run the FIPS 140 program. The 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 http://csrc.nist.gov/groups/STM/cmvp/index.html.

1.2 About this Document

This non-proprietary Cryptographic Module Security Policy for CryptoComply for Libgcrypt from 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 (https://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

This document may be freely reproduced and distributed in its entirety without modification.

2 CryptoComply for Libgcrypt

2.1 Cryptographic Module Specification

CryptoComply for Libgcrypt (hereafter referred to as "the module") is a software library implementing general purpose cryptographic algorithms. The software version is 4.0.

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'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 level of validation for each area in FIPS 140-2:

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	1
Mitigation of Other Attacks	1

Table 1 – Validation Level by FIPS 140-2 Section

2.1.2 Modes of Operation

The module supports two modes of operation: FIPS approved and non-approved modes.
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, Libgcrypt is put into FIPS mode at initialization time
- If the file /etc/gcrypt/fips_enabled exists, Libgcrypt is put into FIPS mode at initialization time. Note that this filename is hardwired and 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.

When Libgcrypt is in the FIPS mode of operation, the request of services involving non-FIPS approved algorithms will be denied. However, the module does not check for approved key sizes or approved mode of algorithms.

The services available in FIPS 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-FIPS mode can be found in Section 2.1.5, Table 4.

Note: Using a non-Approved key sizes, algorithms or block chaining mode specified in Table 4 will result in the module implicitly entering the non-FIPS mode of operation.

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 Cert.	Algorithm	Standard	Mode/Method	Key Lengths, Curves or Moduli	Use
3643, 3644, 3645, 3646	**AES**	FIPS 197 SP 800-38A	ECB, CBC, OFB, CFB128, CTR	128, 192, 256	Encryption, Decryption
972, 973, 974, 975 ----- 979, 980	**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
1020, 1021	**DSA**[1]	FIPS 186-4	Key Pair Generation, Signature Generation, Signature Verification	1024, 2048, 3072 bits (1024 only for SigVer)	Digital Signature Services

[1] DSA signature generation with SHA-1 is only for use with protocols.

CAVP Cert.	Algorithm	Standard	Mode/Method	Key Lengths, Curves or Moduli	Use
2398, 2399	**HMAC**	FIPS 198-1	SHA-1, SHA-224, SHA-256, SHA-384, SHA-512	At least 112 bits KS<BS, KS=BS, KS>BS	Generation, Authentication
1882, 1883	**RSA**	FIPS 186-4 PKCS #1 v2.1 (PSS and PKCS1.5)		1024, 2048, 3072, and 4096 bits (1024 only for SigVer)	Key Pair Generation, Signature Generation, Signature Verification, Component Test
3065, 3066	**SHA**	FIPS 180-4	SHA-1, SHA-224, SHA-256, SHA-384, SHA-512		Digital Signature Generation, Digital Signature Verification, non-Digital Signature Applications
2033, 2034	**Triple-DES**	SP 800-67	TECB, TCBC, TCFB64, TOFB, CTR	2-key, 3-key	Encryption, Decryption

Table 2 – FIPS-Approved Algorithm Certificates

2.1.4 Non-Approved Cryptographic Algorithms

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

Algorithm	Use
RSA Key Encrypt/Decryption	[IG D.9] RSA may be used by a calling application as part of a key encapsulation scheme. Key sizes: 2048 and 3072 bits
NDRNG	Generation of random numbers

Table 3 – Non-Approved but Allowed Cryptographic Algorithms

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.

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)
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
HMAC (SHA1, SHA224, SHA256, SHA384 and SHA512)	Key size < 112 bits
IDEA	Encryption and decryption
MD4	Hashing Digest size 128 bit

Algorithm	Use
MD5	Hashing Digest size 128 bit
OpenPGP S2K Salted and Iterated/salted	Password based key derivation compliant with OpenPGP (RFC4880)
RC2	Encryption and decryption based on RFC 2268
RIPEMD 160	Hashing
RSA	Encryption/decryption: 1024 bits
	Signature generation, key generation: 1024 bits
SEED	Encryption and decryption
Serpent	Encryption and decryption
Tiger	Hashing
Twofish	Encryption and decryption
2-key Triple-DES	Encryption
Whirlpool	Hashing
Services available in FIPS mode	The services available in FIPS mode can be used in non-FIPS mode CSPs/keys separation is enforced between both modes

Table 4 – Non-Approved Cryptographic Functions for use in non-FIPS mode only

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 used within the module:

CSP	Description / Usage	Key Generation	Key Storage	Key Entry/Output	Key Zeroization
AES Keys	[FIPS-197, Addendum to SP 800-38A] AES (128/192/256) encrypt key[19] Encryption and decryption	Use of the module's SP 800-90A DRBG	Application's 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_free()
Triple-DES Keys	Encryption and decryption	Use of the module's SP 800-90A DRBG	Application's 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_free()
DSA Private Keys	Signature generation	Use of the module's SP 800-90A DRBG and the module's DSA key generation mechanism	Application's 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_free()
RSA Private Keys	Signature generation	Use of the module's SP 800-90A DRBG and the module's RSA key generation mechanism	Application's 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_free()

[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	Description / Usage	Key Generation	Key Storage	Key Entry/Output	Key Zeroization
SP 800-90A DRBG Entropy string	Seeding material	The seed data obtained from hardware random number generator /dev/random	Application's memory	N/A	Automatically zeroized when freeing the cipher handler by calling gcry_free()
SP 800-90A DRBG Seed and internal state values (C and V values)	DRBG state	Based on entropy string as defined in SP 800-90A	Application's memory	N/A	Automatically zeroized when freeing the cipher handler by calling gcry_free()
HMAC Keys	Keyed hashing	Use of the module's SP 800-90A DRBG	Application's 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_free()

Table 5 – Critical Security Parameters

2.2.2 Random Number Generation

The module employs a Deterministic Random Bit Generator (DRBG) based on [SP800-90A] for the creation of asymmetric and symmetric keys.

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 DRBG is seeded during initialization with a seed obtained from /dev/random of the appropriate length depending on the instantiated type (see Section 10 of [SP800-90A]).

The module performs continuous tests on the output of the DRBG to ensure that consecutive random numbers do not repeat. The noise source of /dev/random also implements continuous tests.

2.2.3 Key / Critical Security Parameter (CSP) Access

An authorized application user (the User role) has access to all key data generated during the operation of the module. Moreover, the module does not support the output of intermediate key generation values during the key generation process.

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 keys.

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 by using the API function gcry_free(). 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:

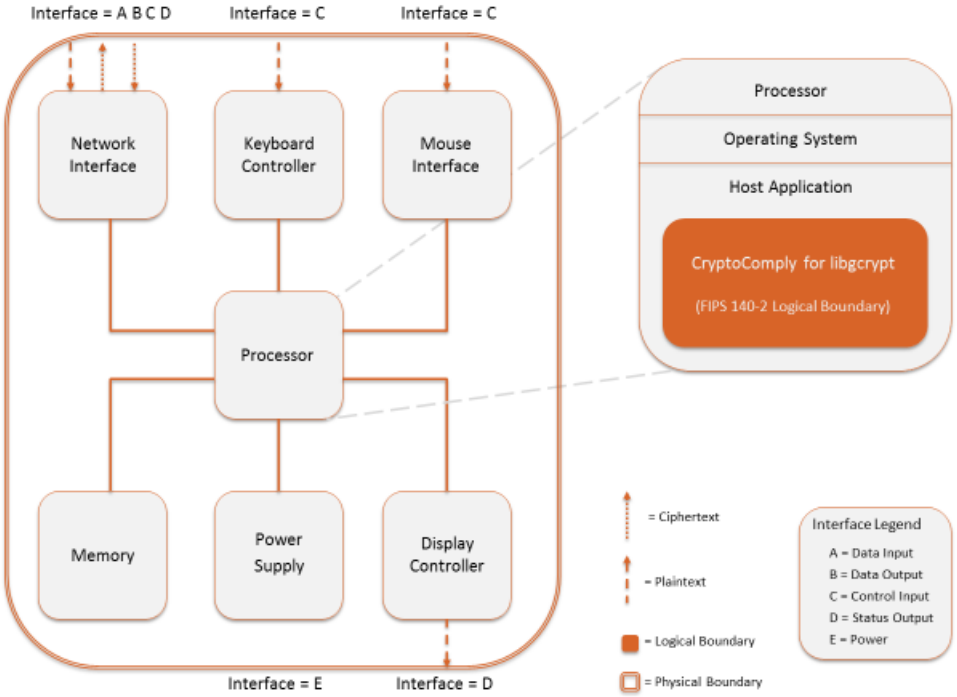

Figure 1 – Module Boundary and Interfaces Diagram

The interfaces (ports) for the physical boundary include the computer keyboard port, mouse port, network port, USB ports, display and power plug. 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 and are provided through the Application Programming Interface (API) that a calling daemon can operate. The logical interfaces expose services that applications directly call, and the API provides functions that may be called by a referencing application (see Section 2.4 –

Roles, Services, and Authentication 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: data input, data output, control input, and status output. Each of the FIPS 140- 2 logical interfaces relates to the module's callable interface, as follows:

FIPS 140-2 Interface	Logical Interface	Module 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	Keyboard Interface, Mouse Interface
Status Output	API return codes, API output parameters	Display Controller, Network Interface
Power	None	Power Supply

Table 6 – Logical Interface / Physical Interface Mapping

The Data Input interface consists of the input parameters of the API functions. The Data Output interface consists of the output parameters of the API functions. The Control Input interface consists of the API function calls and the input parameters used to control the behavior of the module. The Status Output interface includes the return values of the API functions and status sent through output parameters.

As shown in Figure 1 – Module Boundary and Interfaces Diagram and Table 8 – Cryptographic Module's Approved Services, 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 Assumption of Roles

The module supports two distinct operator roles, User and Crypto Officer (CO). The cryptographic module implicitly maps the two roles to the services. 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.

Role	Role Description	Authentication Type
CO	Performs module installation and configuration and some basic functions: get status function and performing self-tests.	N/A – Authentication is not a requirement for Level 1
User	Performs all services, except module installation and configuration.	N/A – Authentication is not a requirement for Level 1

Table 7 – Description of Roles

2.4.2 Services

All services implemented by the module are listed in Table 8 – Cryptographic Module's Approved Services. The second column provides a description of each service and availability to the Crypto Officer and User, in columns 3 and 4, respectively.

Service	Description	CO	User
Symmetric Encryption/Decryption	AES and Triple-DES encryption and decryption		X
Get Key Length	cipher_get_keylen() function		X
Get Block Length	Cipher_get_blocksize() funciton		X
Check Availability of Algorithm	Cipher_get_blocksize() function		X
Secure Hash Algorithm (SHS)	SHA function		X
HMAC	HMAC function		X
RSA	FIPS 186-4 RSASSA-PKCS #1.5 and RSASSA-PSS function		X

Service	Description	CO	User
DSA	DSA FIPS 186-4 function		X
Generate Random Numbers	Fill buffer with length random bytes, function to allocate a memory block consisting of nbytes of random bytes, function to allocate a memory block consisting of nbytes fresh random bytes using a random quality as defined by level. This function differs from gcry_randomize() in that the returned buffer is allocated in a "secure" area of the memory		X
Initialize Module	Powering-up the module		X
Selftests	Performs Known Answer Tests (KAT) and Integrity check	X	X
Zeroize Secure Memory	Gcry_free() or gcry_xfree() functions		X
Release all Resources of Context Created By gcry_cipher_open()	Zeroizes all sensitive information associated with this cipher handle		X
Release all Resources of Hash Context Created by gcry_md_open()	Zeroizes all sensitive information associated with this cipher handle		X
Release the S-expression Objects SEXP	N/A		X
Show Status	N/A	X	X
Installation and Configuration of the Module	N/A	X	

Table 8 – Cryptographic Module's Approved Services

Table 9 – CSP Access Rights within Services defines the relationship between access to CSPs and the different module services. The modes of access shown in the table are defined as:

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.

Service	AES Keys	Triple-DES Keys	DSA Private Keys	RSA Private Keys	SP 800-90A DRBG Entropy String	DRBG Seed and internal state values (C and V values)	HMAC Keys
Symmetric Encryption/Decryption	RWE	RWE	-	-	-	-	-
Get Key Length	-	-	-	-	-	-	-
Get Block Length	-	-	-	-	-	-	-
Check Availability of Algorithm	-	-	-	-	-	-	-
Secure Hash Algorithm (SHS)	-	-	-	-	-	-	-
HMAC	-	-	-	-	-	-	RWE
RSA	-	-	-	RWE	-	-	-
DSA	-	-	RWE	-	-	-	-
Generate Random Numbers	-	-	-	-	-	WE	-
Initialize Module	-	-	-	-	-	-	-
Selftests	-	-	-	-	-	-	-

Service	AES Keys	Triple-DES Keys	DSA Private Keys	RSA Private Keys	SP 800-90A DRBG Entropy String	DRBG Seed and internal state values (C and V values)	HMAC Keys
Zeroize Secure Memory	Z	Z	Z	Z	Z	Z	Z
Release all resources of connect created by gcry_cipher_open()	WE	WE	-	-	-	-	-
Release all resources of hash context created by gcry_md_oopen()	-	-	-	-	-	-	-
Release the S-expression objects SEXP	-	-	RWE	RWE	-	-	-
Show Status	-	-	-	-	-	-	-
Installation and Configuration on the Module	-	-	-	-	-	-	-

Table 9 – CSP Access Rights within Services

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 Level 1 definitions. The module runs on a commercially available general-purpose operating system executing on the hardware specified below.

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 FIPS Approved mode, the ptrace(2) system call, the debugger (gdb(l)). and strace(l) shall be not used.

The module was tested on the following platforms:

Hardware	Processor	Operating System	w/AES-NI	Without AES-NI
HP Proliant DL380p Gen8	Intel® Xeon® E5-2600 v3 product family	Red Hat Enterprise Linux 7.1	Yes	Yes

Table 10 – FIPS Tested Configurations

The module also includes algorithm implementations using Processor Algorithm Acceleration (PAA) functions provided by the different processors supported, as shown in the following:

Processor	Processor Algorithm Acceleration (PAA) Function	Cryptographic Module Implementation
Intel x86	AES-NI	AES

Table 11 – PAA Function Implementations

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.

While the module is performing the power-up tests, 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.

2.7.1 Power-Up Self-Tests

Algorithm	Test
Triple-DES	KAT, encryption and decryption tested separately
AES 128	KAT, encryption and decryption tested separately
AES 192	KAT, encryption and decryption tested separately
AES 256	KAT, encryption and decryption tested separately
SHA-1	KAT
SHA-224	KAT
SHA-256	KAT
SHA-384	KAT
SHA-512	KAT
HMAC SHA-1	KAT
HMAC SHA-256	KAT
HMAC SHA-384	KAT
HMAC SHA-512	KAT
DRBG (Hash, HMAC and CTR-based)	KAT

Algorithm	Test
RSA	KAT of signature generation/verification
DSA	PCT of signature generation/verification
Module Integrity Test	HMAC SHA-256

Table 12 – Power-Up Self-Tests

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, plus some extended self-tests, such as testing additional block chaining modes. During the execution of the on-demand self-tests, services are not available and no data output or input is possible. To invoke the on-demand self-tests, the user can invoke the gcry_control(GCRYCTL_SELFTEST) command.

2.7.3 Conditional Self-Tests

The module implements the following conditional self-tests upon key generation, or random number generation (respectively):

Test Target	Description
DRBG	The continuous random number test is only used in FIPS mode. The RNG generates random numbers per block size depending on the underlying DRBG type (CTR; HMAC or Hash); the first block generated per context is saved in the context and another block is generated to be returned to the caller. Each block is compared against the saved block and then stored in the context. If a duplicated block is detected, an error is signaled and the library is put into the "Fatal-Error" state. (random/drbg.c:cdrbg_fips_continuous_test)
DSA	The test uses a random number of the size of the q parameter to create a signature and then checks that the signature verification is successful. As a second signing test, the data is modified by incrementing its value and then is verified against the signature with the expected result that the verification fails. (cipher/dsa.c:test_keys())
RSA	The test creates a random number of the size of p-64 bits and encrypts this value with the public key. Then the test checks that the encrypted value does not match the plaintext value. The test decrypts the ciphertext value and checks that it matches the original plaintext. The test will then generate another random plaintext, sign it, modify the signature by incrementing its value by 1, and verify that the signature verification fails. (cipher/rsa.c:test_keys())

Table 13 – Conditional Self-Tests

2.8 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·r" mod is decrypted and the unblinded value (x' = y" ·r1 mod n) returned. The blinding value "r" is 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*/
{ 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*/
{ 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. The module and its host application are to be installed on an operating system specified in Section 2.6 or one where portability is maintained.

Because FIPS 140-2 has certain restrictions on the use of cryptography which are not always wanted, the Module needs to be put into FIPS Approved mode explicitly: 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.

If an application that uses the Module for its cryptography is put into a chroot environment, the Crypto Officer must ensure one of the above 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.

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

Because FIPS 140-2 has certain restrictions on the use of cryptography which are not always wanted, Libgcrypt needs to be put into FIPS mode explicitly. To switch Libgcrypt into this mode, the file /proc/sys/crypto/fips_enabled must contain a numeric value other than 0. If the application requests FIPS mode, use 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).

Once Libgcrypt has been put into FIPS mode, it is not possible to switch back to standard mode without terminating the process first. If the logging verbosity level of Libgcrypt has been set to at least 2, the state transitions and the self-tests are logged.

3.2 User Guidance

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 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.

4 References and Acronyms

4.1 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*

Table 14 – References

4.2 Acronyms

The following table defines acronyms found in this document:

Acronym	Term
AES	Advanced Encryption Standard
API	Application Programming Interface
CAVP	Cryptographic Algorithm Validation Program
CBC	Cipher-Block Chaining
CFB	Cipher Feedback Mode
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

Acronym	Term
ECB	Electronic Code Book
EMC	Electromagnetic Compatibility
EMI	Electromagnetic Interference
FCC	Federal Communications Commission
FIPS	Federal Information Processing Standard
GPC	General Purpose Computer
HMAC	(Keyed-) Hash Message Authentication Code
IG	Implementation Guidance
KAT	Known Answer Test
MAC	Message Authentication Code
N/A	Non 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
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

Table 15 – Acronyms and Terms