SUSE Linux Enterprise Server 12 libgcrypt Cryptographic Module version 1.0

FIPS 140-2 Non-Proprietary Security Policy

Version 1.3

Last update: 2015-10-23

Prepared by:

atsec information security corporation

9130 Jollyville Road, Suite 260

Austin, TX 78759

www.atsec.com

Table of Contents

1 Introduction

This document is the non-proprietary Security Policy for the SUSE Linux Enterprise Server 12 libgcrypt Cryptographic Module version 1.0. 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 module.

2 Cryptographic Module Specification

2.1 Module Overview

The SUSE Linux Enterprise Server 12 libgcrypt Cryptographic Module (hereafter referred to as “the module”) is a software library implementing general purpose cryptographic algorithms. 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 implemented as a set of shared libraries / binary files; as shown in the diagram below, the shared library files and the integrity check file used to verify the module's integrity constitute the logical cryptographic boundary:

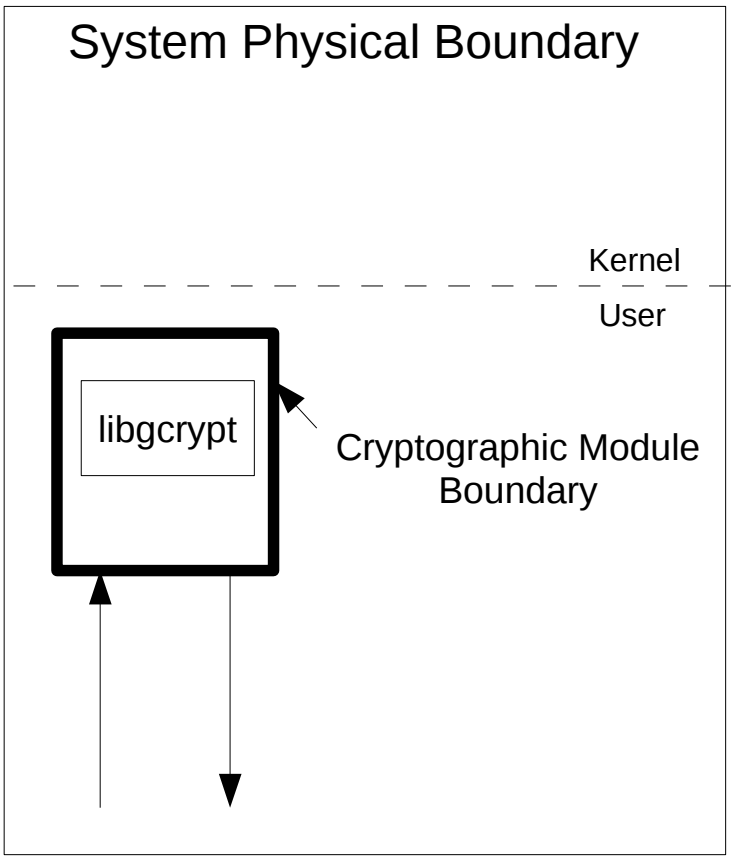

Figure 1: Software Block Diagram

The module is aimed to run in a general purpose computer; the physical boundary is the surface of the case of the target platform, as shown in the diagram below:

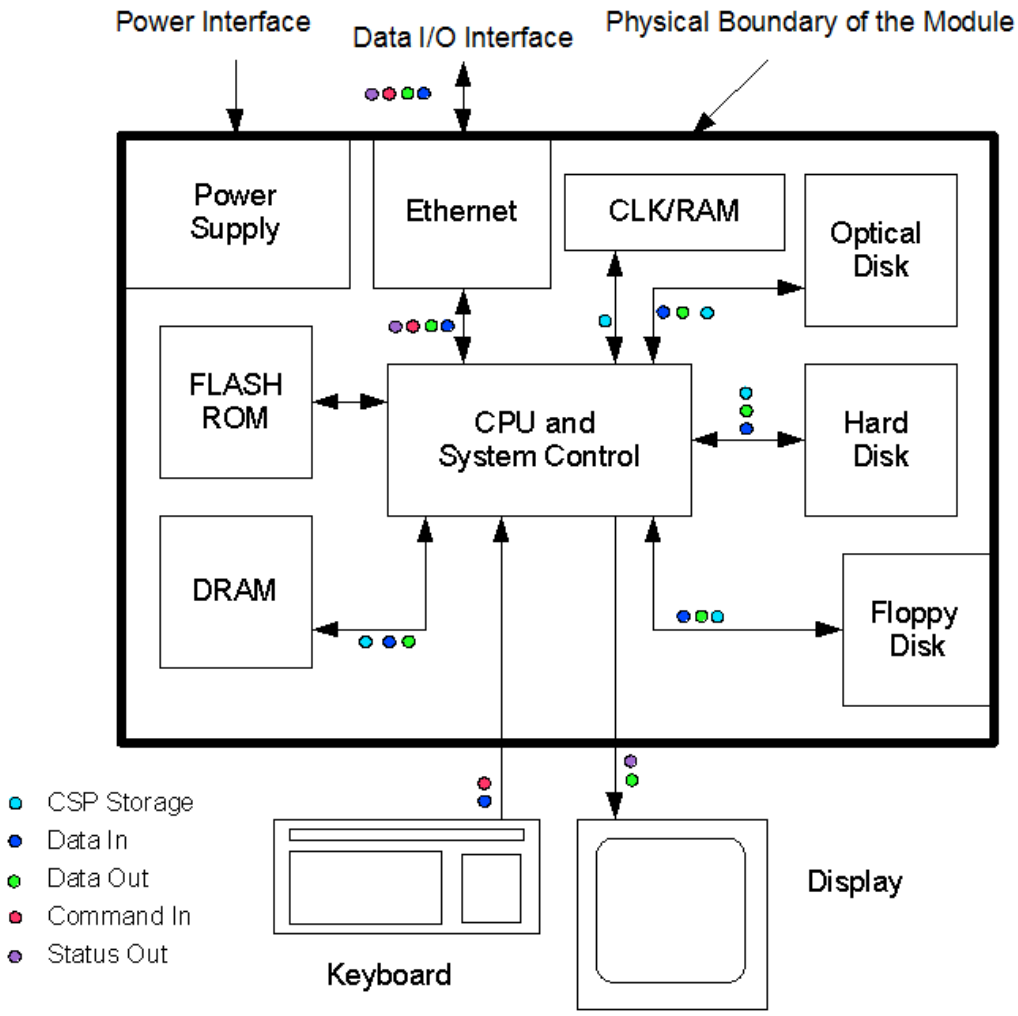

Figure 2: Cryptographic Module Physical Boundary

All components of the module will be in the libgcrypt RPM version 1.6.1-13.1. The following RPMs files are part of the module:

- dracut-fips-037-37.2.x86_64
- libgcrypt20-1.6.1-13.1.x86_64
- libgcrypt20-hmac-1.6.1-13.1.x86_64

When installed on the system, the module comprises the following files:

- /usr/lib64/libgcrypt.so.20.0.1
- /usr/lib64/.libgcrypt.so.20.hmac

2.2 FIPS 140-2 validation

For the purpose of the FIPS 140-2 validation, the module is a software-only, multi-chip standalone cryptographic module validated at 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

Table 1: Security Levels

The module has been tested on the following platforms:

Module Version	Hardware	Processor	Operating System
HP	ProLiant DL320e Gen8	x86_64	SUSE Linux Enterprise Server 12

Table 2: Tested Platforms

The physical boundary is the surface of the case of the target platform. The logical boundary is depicted in the software block diagram.

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

Processor	Processor Algorithm Acceleration (PAA) function	Cryptographic Module implementation
Intel x86	AESNI	AES
Intel x86	SSSE3, AVX, AVX2	SHS

Table 3: PAA function implementations

2.3 Modes of Operations

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

The module turns to FIPS approved mode after initialization and power-on self-tests succeed.

The mode of operation in which the module is operating can be determined by invoking a non FIPS-approved service: if the module is in the FIPS-mode, the service will fail. The user of the module can also check the flags /proc/sys/crypto/fips_enabled: if the value in this file is non-zero, the module will boot in FIPS mode.

The services available in FIPS mode can be found in section 4.2, Table 5.

The services available in non-FIPS mode can be found in section 4.2, Table 6.

3 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 application program interface (API) through which applications request services. The following table summarizes the four logical interfaces:

Logical interface	Description
Data input	API input parameters for data
Data output	API output parameters for data
Control input	API function calls, API input parameters, /proc/sys/crypto/fips_enabled control file
Status output	API return codes, API output parameters

Table 4: Logical Interfaces

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.

4 Roles, Services and Authentication

4.1 Roles

The module supports the following roles:

- **User role**: performs all services, except module installation and configuration.
- **Crypto Officer role**: performs module installation and configuration and some basic functions: get status function and performing self-tests.

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

4.2 Services

The module supports services available to users in the available roles. All services are described in detail in the user documentation.

The following table shows the available services, the roles allowed (“CO” stands for Crypto Officer role and “U” stands for User role), the Critical Security Parameters involved and how they are accessed in the FIPS mode:

Service	Algorithm	Key Length	Note / Mode	CAVS Cert.	Role		CSPs	Access
Symmetric encryption/ decryption	Triple-DES	168 bits	With all 3 keys independent; Modes: ECB, CBC, CFB, OFB, CTR	Cert. #1936	U		168 bits Triple-DES Key	R, W, EX

Service	Algorithm	Key Length	Note / Mode	CAVS Cert.	Role		CSPs	Access
	AES	128, 192 and 256 bits	AES-NI implementation; Modes: ECB, CBC, OFB, CFB, CTR	Cert. #3433	U		128/192/256 bits AES Key	R, W, EX
			Generic assembler implementation; Modes: ECB, CBC, OFB, CFB, CTR	Cert. #3434				
Get Key Length	N/A	N/A	cipher_get_keylen() function	N/A	U		N/A	R
Get Block Length	N/A	N/A	cipher_get_blocksize() function	N/A	U		N/A	R
Check availability of Algorithm	N/A	N/A	cipher_get_blocksize() function	N/A	U		N/A	R
Hash	SHA-1, SHA-224, SHA-256, SHA-384, SHA-512	N/A	Generic C implementation	Cert. #2831	U		N/A	R, W, EX
			SSSE3 assembler implementation	Cert. #2832				
	SHA-384, SHA-512	N/A	AVX assembler implementation	Cert. #2833				
			AVX2 assembler implementation	Cert. #2834				
HMAC	HMAC-SHA-1, HMAC-SHA-224, HMAC-SHA-256, HMAC-SHA-384, HMAC-SHA-512	At least 112 bits KS<BS, KS=BS, KS>BS	Generic C implementation	Cert. #2183	U		MAC-key	R, W, EX
			SSSE3 assembler implementation	Cert #2184				
	HMAC-SHA-384, HMAC-SHA-512		AVX assembler implementation	Cert. #2185				
			AVX2 assembler implementation	Cert. #2186				
RSA	Key pair generation, signature generation and verification	2048 and 3072 bits modulus	FIPS 186-4, PKCS #1.5	Cert. #1757	U		RSA private key	R, W, EX
	Key wrapping	2048 and 3072 bits modulus	Non-approved but allowed	N/A				
DSA	Key pair generation, signature generation and	L=2048, N=224; L=2048, N=256;	FIPS 186-4	Cert. #967	U		DSA private keys	R, W, EX

Service	Algorithm	Key Length	Note / Mode	CAVS Cert.	Role		CSPs	Access
	verification	L=3072, N=256;						
ECDSA	Key pair generation, public key verification, signature generation and verification	Key lengths according to the NIST curves P-224, P-256, P-384 and P-521	FIPS 186-4 ANSI X9.62	Cert. #689	U		Curves P-224, P-256, P-384, P-521	R, W, EX
Generate random numbers	DRBG: HMAC-SHA-1/256/384/512 DRBG: HASH-SHA-1/256/384/512 (with and without prediction resistance)	N/A	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	Certs. #831, #832, #833 and #834	U		Seed	W, EX
	DRBG: CTR with derivation function; AES 128/192/256 (ECB mode, with and without prediction resistance)			Certs. #831 and #832				
Initialize Module	N/A	N/A	Powering-up the module	N/A	U		N/A	EX
Selftests	N/A	N/A	Performs Known Answer Test (KAT) and integrity check	N/A	U	CO	N/A	EX
Zeroize secure memory	N/A	N/A	gcry_free() or gcry_xfree() functions	N/A	U		All CSPs stored in that secure memory	W, EX
Release all resources of context created by gcry_cipher_open()	N/A	N/A	Zeroises all sensitive information associated with this cipher handle	N/A	U		Cipher secret keys	W, EX
Release all resources of hash context created by	N/A	N/A	Zeroises all sensitive information associated with this cipher	N/A	U		N/A	W, EX

Service	Algorithm	Key Length	Note / Mode	CAVS Cert.	Role		CSPs	Access
gcry_md_open()			handle					
Release the S-expression objects SEXP	N/A	N/A	N/A	N/A	U		RSA/DSA asymmetric key pair	R, W, EX
Show Status	N/A	N/A	N/A	N/A	U	CO	N/A	R, EX
Installation and configuration of the module	N/A	N/A	N/A	N/A		CO	N/A	R, EX

Table 5: Available Cryptographic Module's Services in FIPS mode

The following table shows the available services, the roles allowed, the Critical Security Parameters involved and how they are accessed in the non-FIPS mode:

Service (involving algorithm)	Note / Mode	Role		Access
AES	GCM	U		R, W, EX
	Key wrapping	U		
ARC4	Encryption and decryption (stream cipher)	U		
Blowfish	Encryption and decryption	U		
Camellia	Encryption and decryption	U		
Cast5	Encryption and decryption	U		
CRC32	Cyclic redundancy code	U		
DES	Encryption and decryption (key size of 56 bits)	U		
ECDSA	Elliptic curve digital signature algorithm using Ed25519, Brainpool curves (P-160r1, P-192r1, P-224r1, P-256r1, P-320r1, P-384r1, P-512r1 curves) or NIST P-192 curve	U		
EC-Gost	Elliptic curve digital signature algorithm using NIST curves (P-192, P-224, P-256, P-384 and P-521), Ed25519 or Brainpool curves (P-160r1, P-192r1, P-224r1, P-256r1, P-320r1, P-384r1, P-512r1)	U		
EdDSA	Elliptic curves digital signature algorithm using NIST curves (P-192, P-224, P-256, P-384 and P-521), Ed25519 or Brainpool curves (P-160r1, P-192r1, P-224r1, P-256r1, P-320r1, P-384r1, P-512r1)	U		
El Gamal	Key pair generation, encryption and decryption, signature generation, signature verification	U		
Gost	28147 encryption	U		
	R 34.11-94 hash	U		
	R 34.11-2012 (Stribog) hash			
HMAC (SHA1, SHA224, SHA256, SHA384 and SHA512)	Key size < 112 bits	U		

Service (involving algorithm)	Note / Mode	Role		Access
IDEA	Encryption and decryption	U		
MD4	Hashing Digest size 128 bit	U		
MD5	Hashing Digest size 128 bit	U		
OpenPGP S2K Salted and Iterated/salted	Password based key derivation compliant with OpenPGP (RFC4880)	U		
RC2	Encryption and decryption based on RFC 2268	U		
RIPE-MD 160	Hashing	U		
RSA	RSA (key wrapping): 1024 bits (not allowed), 2048 bits (allowed), 3072 bits (allowed) and 4096 bits (not allowed)	U		
	RSA (signature generation, verification and key generation): 1024 and 4096 bits	U		
Salsa20	Encryption and decryption (stream cipher)	U		
SEED	Encryption and decryption	U		
Serpent	Encryption and decryption	U		
Scrypt	Password based key derivation	U		
Tiger	Hashing	U		
Twofish	Encryption and decryption	U		
Whirlpool	Hashing	U		
Services available in FIPS mode	The services available in FIPS mode can be used in non-FIPS mode, but CSP/key separation is enforced between both modes	U		

Table 6: Available Cryptographic Module's Services in non-FIPS mode

4.3 Authentication

The module is a Level 1 software-only cryptographic module and does not implement authentication. The role is implicitly assumed based on the service requested.

5 Physical Security

The module is comprised of software only and thus does not claim any physical security.

6 Operational Environment

6.1 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 section 2.2.

6.2 Policy

The operating system is restricted to a single operator (concurrent operators are explicitly excluded). The application that request 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(1)), and strace(1) shall be not used.

7 Cryptographic Key Management

The application that uses the module is responsible for appropriate destruction and zeroization of the key material. The library provides functions for key allocation and destruction, which overwrites the memory that is occupied by the key information with “zeros” before it is deallocated.

7.1 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/urandom of length 3/2 times the DRBG strength.

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/urandom also implements continuous tests.

Here are listed the CSPs/keys details concerning storage, input, output, generation and zeroization:

Keys/CSPs	Key Generation	Key Storage	Key Entry/Output	Key Zeroization
AES Keys	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	Automatic zeroized when freeing the cipher handler
Triple-DES Keys	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	Automatic zeroized when freeing the cipher handler
DSA private keys	Use of the module's SP 800-90A DRBG and the modules DSA key generation mechanism	Application's memory	API input/output parameters and return values within the physical boundaries of the module	Automatic zeroized when freeing the cipher handler
RSA private keys	Use of the module's SP 800-90A DRBG and the modules RSA key generation mechanism	Application's memory	API input/output parameters and return values within the physical boundaries of the module	Automatic zeroized when freeing the cipher handler
ECDSA private keys	Use of the module's SP 800-90A DRBG and the modules ECDSA key generation mechanism	Application's memory	API input/output parameters and return values within the physical boundaries of the module	Automatic zeroized when freeing the cipher handler

SP 800-90A DRBG Entropy string	The seed data obtained from hardware random number generator /dev/urandom	Application's memory	N/A	Automatic zeroized when freeing DRBG handler
SP 800-90A DRBG Seed and internal state values	Based on entropy string as defined in SP 800-90A	Application's memory	N/A	Automatic zeroized when freeing DRBG handler
HMAC Keys	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	Automatic zeroized when freeing the cipher handler

Table 7: Keys/CSPs

7.2 Key / Critical Security Parameter (CSP) Access

An authorized application as 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.

7.3 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.

7.4 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. 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.

8 Self Tests

8.1 Power-Up 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 selftests 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.

8.1.1 Integrity Tests

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

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) shown in the following table:

Algorithm	Tests
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-224	KAT
HMAC SHA-256	KAT
HMAC SHA-384	KAT
HMAC SHA-512	KAT
DRBG (Hash, HMAC and CTR-based)	KAT
RSA	KAT of signature generation/verification
DSA	KAT of signature generation/verification
ECDSA	KAT of signature generation/verification
Module Integrity test	HMAC SHA-256

Table 8: Self-tests

8.2 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.

8.3 Conditional Tests

The module performs conditional tests on the cryptographic algorithms shown in the following table:

Algorithm	Test
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 1st block generated per context is

Algorithm	Test
	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)
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())
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 successfull. 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())
ECDSA	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 (cipher/ecc.c:test_keys())

Table 9: Conditional Tests

9 Guidance

The following guidance items are to be used for assistance in maintaining the module's validated status while in use.

9.1 Crypto Officer Guidance

The Module is delivered as a binary object file packaged in an RPM. The integrity of the RPM is automatically verified during the installation and the Crypto Officer shall not install the RPM file if the RPM tool indicates an integrity error. The version of the RPM containing the validated module is stated in section 2.1 above.

The RPM package of the Module can be installed by standard tools recommended for the installation of RPM packages on a SUSE Linux system (for example, rpm, yast and yast online_update).

For proper operation of the in-Module integrity verification, the prelink has to be disabled. This can be done by setting PRELINKING=no in the /etc/sysconfig/prelink configuration file. If the libraries were already prelinked, the prelink should be undone on all the system files using the 'prelink -u -a' command.

To bring the Module into FIPS approved mode, perform the following:

1. Install the dracut-fips package:

```
# zypper install dracut-fips
```

2. Recreate the INITRAMFS image:

```
# dracut -f
```

After regenerating the initrd, the crypto officer has to append the following parameter in the /etc/default/grub configuration file in the GRUB_CMDLINE_LINUX_DEFAULT line:

```
fips=1
```

After editing the configuration file, please run the following command to change the setting in the boot loader:


```
grub2-mkconfig -o /boot/grub2/grub.cfg
```

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

```
$ df /boot
Filesystem      1K-blocks      Used      Available      Use%      Mounted on
/dev/sda1        233191        30454       190296       14%         /boot
```

The partition of /boot is located on /dev/sda1 in this example. Therefore, the following string needs to be appended to the kernel command line:

```
"boot=/dev/sda1"
```

Reboot to apply these settings.

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.

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.

9.2 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
```

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 ($y = x \cdot r^e \bmod n$) is decrypted and the unblinded value ($x' = y' \cdot r^{-1} \bmod n$) 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*/
  { 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*/ };
```


Appendix A Glossary and Abbreviations

AES	Advanced Encryption Standard
AES-NI	Advanced Encryption Standard New Instructions
CAVP	Cryptographic Algorithm Validation Program
CBC	Cipher Block Chaining
CCM	Counter with Cipher Block Chaining Message Authentication Code
CFB	Cipher Feedback
CMAC	Cipher-based Message Authentication Code
CMT	Cryptographic Module Testing
CMVP	Cryptographic Module Validation Program
CSP	Critical Security Parameter
CTR	Counter Mode
CVT	Component Verification Testing
DES	Data Encryption Standard
DFT	Derivation Function Test
DSA	Digital Signature Algorithm
DRBG	Deterministic Random Bit Generator
ECB	Electronic Code Book
ECC	Elliptic Curve Cryptography
FFC	Finite Field Cryptography
FIPS	Federal Information Processing Standards Publication
FSM	Finite State Model
GCM	Galois Counter Mode
HMAC	Hash Message Authentication Code
KAT	Known Answer Test
MAC	Message Authentication Code
NIST	National Institute of Science and Technology
NDRNG	Non-Deterministic Random Number Generator
OFB	Output Feedback
OS	Operating System
PAA	Processor Algorithm Acceleration
PR	Prediction Resistance
PSS	Probabilistic Signature Scheme
RNG	Random Number Generator

RSA	Rivest, Shamir, Addleman
SHA	Secure Hash Algorithm
SHS	Secure Hash Standard
SSH	Secure Shell
TDES	Triple DES
UI	User Interface
XTS	XEX-based Tweaked-codebook mode with ciphertext Stealing

Appendix B References

FIPS180-4 **Secure Hash Standard (SHS)**
March 2012
http://csrc.nist.gov/publications/fips/fips180-4/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

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

RFC3394 **Advanced Encryption Standard (AES) Key Wrap Algorithm**
September 2002
http://www.ietf.org/rfc/rfc3394.txt

RFC5649 **Advanced Encryption Standard (AES) Key Wrap with Padding Algorithm**
September 2009
http://www.ietf.org/rfc/rfc5649.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://csrc.nist.gov/publications/nistpubs/800-38C/SP800-38C_updated July20_2007.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

SP800-56A **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

SP800-56C **Recommendation for Key Derivation through Extraction-then-Expansion**
November 2011
http://csrc.nist.gov/publications/nistpubs/800-56C/SP-800-56C.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 - Recommendation for Random Number Generation Using Deterministic Random Bit Generators**
January 2012
http://csrc.nist.gov/publications/nistpubs/800-90A/SP800-90A.pdf

SP800-90B **NIST Draft Special Publication 800-90B - Recommendation for the Entropy Sources Used for Random Bit Generation**
August 2012
http://csrc.nist.gov/publications/drafts/800-90/draft-sp800-90b.pdf

SP800-108 **NIST Special Publication 800-108 - Recommendation for Key Derivation Using Pseudorandom Functions**
October 2009
http://csrc.nist.gov/publications/nistpubs/800-108/sp800-108.pdf

SP800-131A **NIST Special Publication 800-131A - Transitions: Recommendation for Transitioning the Use of Cryptographic Algorithms and Key Lengths**
January 2011
http://csrc.nist.gov/publications/nistpubs/800-131A/sp800-131A.pdf

