Red Hat Enterprise Linux 8 libgcrypt Cryptographic Module version rhel8.20190624

FIPS 140-2 Non-Proprietary Security Policy

Document Version 1.2
Last update: 2020-12-18

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 Red Hat Enterprise Linux 8 libgcrypt Cryptographic Module version rhel8.20190624. 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 Red Hat Enterprise Linux 8 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:

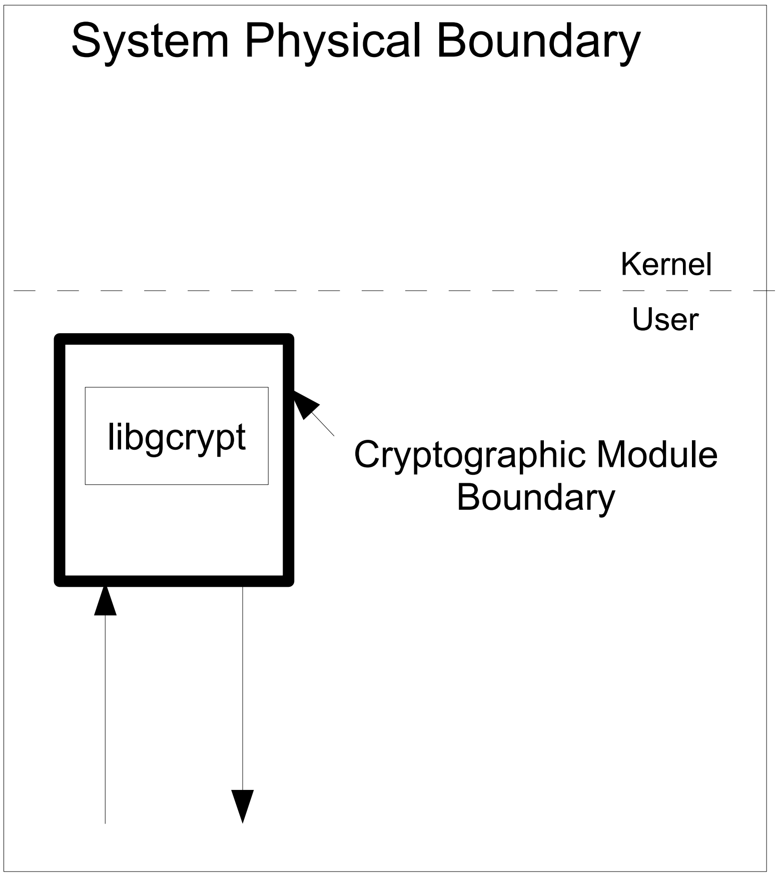

Figure 1: Software Block Diagram

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:

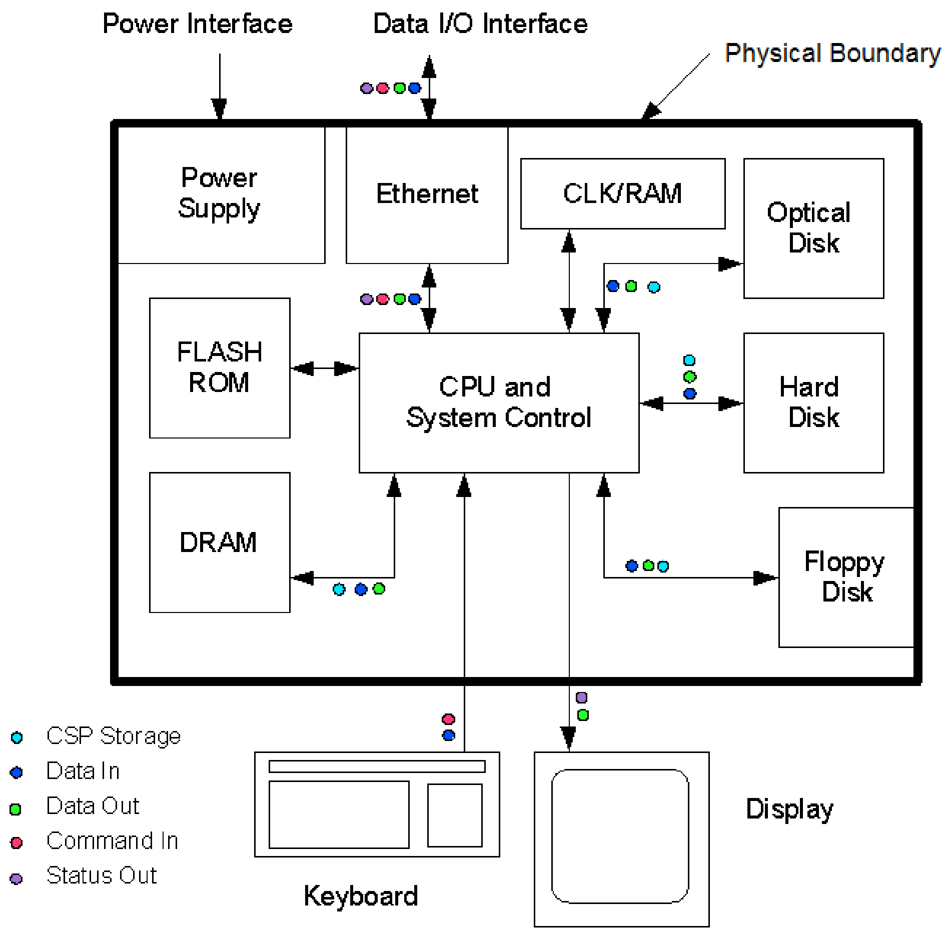

Figure 2: Hardware Block Diagram

All components of the module will be in the libgcrypt RPM. The following RPMs files constitute the module:

- Red Hat Enterprise Linux 8 libgcrypt Cryptographic Module: libgcrypt-1.8.3-4.el8.rpm

When installed on the system, the module comprises the following files[1]:

- /usr/lib64/libgcrypt.so.20.2.3
- /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

1 *Note: the file /usr/lib64/libgcrypt.so.20 is a symlink to /usr/lib64/libgcrypt.so.20.2.3. This symlink file is not part of the module.*

FIPS 140-2 Section		Security Level
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 Red Hat Enterprise Linux 8 libgcrypt Cryptographic Module rhel8.20190624 module has been tested on the following platform(s):

Manufacturer	Model	Processor	Operating System
Dell	PowerEdge R430	Intel(R) Xeon(R) E5	Red Hat Enterprise Linux 8

Table 2: Tested Platform(s) for the Red Hat Enterprise Linux 8 libgcrypt Cryptographic Module

The Module has been tested for the following configurations:

- 64-bit library, x86_64.

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

2.3 Modes of Operations

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

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 Approved services available in FIPS mode can be found in section 4.2, Table 4.

The non-Approved but allowed services available in FIPS mode can be found in section 4.2, Table 5.

The non-Approved and not allowed[2] services available in non-FIPS mode can be found in section 4.2, Table 6.

2 *Note: Using a non-Approved key sizes, algorithms or block chaining mode specified in Table 6 will result in the module implicitly entering the non-FIPS mode of operation.*

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, /etc/gcrypt/fips_enabled configuration file
Status output	API return codes, API output parameters

Table 3: 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	ACVP Cert.	Role		CSPs	Access
Symmetric encryption/ decryption	Triple-DES	168 bits	Modes: ECB, CBC, CFB8,CFB64, OFB, CTR, CMAC 3-key Triple-DES encryption/ decryption 2-key Triple-DES decryption only	Cert. #A175	U		168 bits Triple-DES Key	R, EX
	AES	128, 192 and 256 bits	Modes: ECB, CBC, CFB8, CFB128, OFB, CTR, KW, CCM, CMAC	Cert. #A175	U		128/192/256 bits AES Key	R, EX
		128 and 256 bits	Mode: XTS	Cert. #A175	U		128/256 bits AES Key	R, EX
Get Key Length	N/A	N/A	cipher_get_keylen() function	N/A	U		N/A	N/A
Get Block Length	N/A	N/A	cipher_get_blocksize() function	N/A	U		N/A	N/A
Check algorithm availability	N/A	N/A	cipher_get_blocksize() function	N/A	U		N/A	N/A
Secure Hash	SHA-1,	N/A	N/A	Cert.	U		N/A	N/A

Service	Algorithm	Key Length	Note / Mode	ACVP Cert.	Role		CSPs	Access
Algorithm (SHS)	SHA-224, SHA-256, SHA-384, SHA-512, SHA3-224, SHA3-256, SHA3-384, SHA3-512, SHAKE-128, SHAKE-256			#A175				
HMAC	HMAC-SHA-1, HMAC-SHA-224, HMAC-SHA-256, HMAC-SHA-384, HMAC-SHA-512, HMAC-SHA3-224, HMAC-SHA3-256, HMAC-SHA3-384, HMAC-SHA3-512	At least 112 bits KS<BS, KS=BS, KS>BS	N/A	Cert. #A175	U		MAC key	R, EX
Key pair generation Signature generation	RSA	2048,3072, 4096 bits modulus SHA-224, SHA-256, SHA- 384, SHA-512	FIPS 186-4 RSASSA-PKCS #1.5 RSASSA-PSS	Cert. #A175	U		RSA private key	R, W, EX
	ECDSA	P-256, P-384, P-521 SHA-224, SHA-256, SHA- 384, SHA-512	FIPS 186-4	Cert. #A175	U		ECDSA private key	R, W, EX
	DSA	L=2048, N=224; L=2048, N=256; L=3072, N=256;	FIPS 186-4	Cert. #A175	U		DSA private key	R, W, EX
Signature verification	RSA	2048, 3072, 4096 bits modulus	FIPS 186-4 RSASSA-PKCS #1.5 RSASSA-PSS	Cert. #A175	U		RSA private key	R, EX

Service	Algorithm	Key Length	Note / Mode	ACVP Cert.	Role		CSPs	Access
		SHA-224, SHA-256, SHA- 384, SHA-512						
	ECDSA	P-256, P-384, P-521 SHA-1, SHA-224, SHA-256, SHA- 384, SHA-512	FIPS 186-4	Cert. #A175	U		ECDSA private key	R, EX

	DSA	L=1024, N=160, SHA-1;
		L=2048, N=224, SHA-224;
		L=2048, N=256, SHA-224, SHA-256;
		L=3072, N=256, SHA-224, SHA-256;
			FIPS 186-4	Cert. #A175	U		DSA private key	R, EX

Domain Parameter Generation	DSA PQGGen	L=2048, N=224, SHA-224;
		L=2048, N=256, SHA-256;
		L=3072, N=256, SHA-256
			FIPS 186-4	Cert. #A175	U		N/A	N/A

Domain Parameter Verification	DSA PQGVer	L=2048, N=224, SHA-224;
		L=2048, N=256, SHA-256;
		L=3072, N=256, SHA-256
			FIPS 186-4	Cert. #A175	U		N/A	N/A

Key Pair verification	ECDSA	P-256, P-384, P-521	FIPS 186-4	Cert. #A175	U		ECDSA private key	R, EX
Key derivation	PBKDF (Vendor Affirmed[3])	SHA-1, SHA2-224, SHA2-256, SHA2-384,	SP 800-132	See footnote	U		PBKDF Derived key PBKDF Password	R, W, EX

3 The vendor claims compliance to the SP 800-132 PBKDF2, which was tested through ACVP obtaining cert #A175. However, the module does not implement a KAT for this algorithm and hence it is claimed as vendor affirmed. This vendor affirmed algorithm was tested through ACVP Cert.

Service	Algorithm	Key Length	Note / Mode	ACVP Cert.	Role		CSPs	Access
		SHA2-512, SHA3-224, SHA3-256, SHA3-384, SHA3-512						
Random number generation	SP 800-90A DRBG: HMAC_DRBG with SHA-1/256/512 HASH_DRBG with SHA-1/256/512 (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	Cert. #A175	U		Seed, internal state values and entropy input	W, EX
	SP 800-90A DRBG: CTR_DRBG with derivation function AES 128/192/256 (with and without prediction resistance)			Cert. #A175				
Module initialization	N/A	N/A	Powering-up the module	N/A	U		N/A	N/A
Self-tests	N/A	N/A	Performs Known Answer Test (KAT) and integrity check	N/A	U		N/A	N/A
Secure memory zeroization	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 key	W, EX
Release all resources of	N/A	N/A	Zeroises all sensitive	N/A	U		N/A	N/A

Service	Algorithm	Key Length	Note / Mode	ACVP Cert.	Role		CSPs	Access
hash context created by gcry_md_ope n()			information associated with this cipher 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		N/A	N/A
Installation and configuration of the module	N/A	N/A	N/A	N/A		CO	N/A	N/A

Table 4: Cryptographic Module's Approved Services

Service (involving algorithm)	Note / Mode	Role		CSPs	Acces s
RSA	Encryption/decryption with keys greater than or equal to 2048 bits (key wrapping; key establishment methodology provides between 112 and 150 bits of encryption strength).	U		RSA Private Key	R, EX
NDRNG	Seeding the module's DRBG	U		Seed	R

Table 5: Cryptographic Module's non-Approved but allowed in FIPS mode Services

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	
Symmetric encryption/decry ption	ARC4, Blake2, Blowfish, Camellia, Cast5, ChaCha20, DES, IDEA, RC2, SEED, Serpent, Twofish, 2-key Triple-DES (encryption only), Salsa20, GOST (28147)	U	
Cyclic redundancy code	CRC32	U	
Random number generation	Cryptographically Secure Pseudorandom Number Generator (CSPRNG)	U	
Asymmetric key pair generation	El Gamal	U	
	RSA (keys < 2048 bits)	U	
	EdDSA	U	
Asymmetric encryption/decry ption	El Gamal, RSA (with keys < 2048 bits)	U	
Signature generation	RSA (keys < 2048 bits), El Gamal, EdDSA.	U	
	RSA with SHA-1.	U	
Signature verification	RSA (keys < 2048 bits), El Gamal, EdDSA.	U	
	RSA with SHA-1.	U	

Service (involving algorithm)	Note / Mode	Role	
MAC	HMAC (Key size < 112 bits), Poly1305	U	
Message digest	MD4, MD5, RIPEMD160, TIGER, Whirlpool, GOST (R 34.11-94, R 34.11-2012 (Stribog))	U	
Key derivation	Scrypt KDF, OpenPGP S2K Salted and Iterated/salted (Password based key derivation compliant with OpenPGP (RFC4880))	U	

Table 6: Cryptographic Module's non-Approved Services and Algorithms

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.

The Red Hat Enterprise Linux operating system is used as the basis of other products which include but are not limited to:

- Red Hat Enterprise Linux CoreOS
- Red Hat Virtualization (RHV)
- Red Hat OpenStack Platform
- OpenShift Container Platform
- Red Hat Gluster Storage
- Red Hat Ceph Storage
- Red Hat CloudForms
- Red Hat Satellite.

Compliance is maintained for these products whenever the binary is found unchanged.

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

7.1 Random Number Generation

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

The seeding (and automatic reseeding) of the DRBG is done with getrandom().

The module performs the health tests for the SP800-90A DRBG as defined per section 11.3 of SP800-90A.

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

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

The NDRNG provides 128 bits of entropy to the DRBG. Therefore, the following caveat applies:

The module generates cryptographic keys whose strengths are modified by available entropy.

7.2 Key Establishment

The module provides RSA key wrapping (encapsulation) using public key encryption and private key decryption primitives as allowed by [FIPS140-2_IG] D.9.

RSA provides the following security strengths:

- RSA: key wrapping provides between 112 and 150 bits of encryption strength.

7.3 Key/Critical Security Parameter (CSP)

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	N/A	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	N/A	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	Use of the module's SP 800-90A DRBG and generated using FIPS 186-4 key generation method	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()
ECDSA private keys	Use of the module's SP 800-90A DRBG and generated using FIPS 186-4 key generation method	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_ecc_curve_free()

RSA private keys	Use of the module's SP 800-90A DRBG and generated using FIPS 186-4 key generation method	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()
SP 800-90A DRBG Entropy string	The seed data obtained from getrandom()	Application's memory	N/A	Automatically zeroized when freeing DRBG handler by calling gcry_free()
SP 800-90A DRBG Seed and internal state values (C, K and V values)	Based on entropy string as defined in SP 800-90A	Application's memory	N/A	Automatically zeroized when freeing DRBG handler by calling gcry_free()
HMAC Keys	N/A	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()
PBKDF Derived Key	SP800-132 PBKDF mechanisms	Application's memory	API output parameters and return values within the physical boundaries of the module	Automatically zeroized when freeing the cipher handler by calling gcry_free()
PBKDF Password	N/A	Application's memory	API input parameters and return values within the physical boundaries of the module	Automatically zeroized when freeing the cipher handler by calling gcry_free()

Table 7: Keys/CSPs

7.4 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.

7.5 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.

7.6 Key / CSP Zeroization

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.

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.

8 Electromagnetic Interference/Electromagnetic Compatibility (EMI/EMC)

MARKETING NAME.........................	PowerEdge R430
REGULATORY MODEL....................	E28S
REGULATORY TYPE........................	E28S001
EFFECTIVE DATE............................	December 02, 2014
EMC EMISSIONS CLASS.................	Class A

8.1 Statement of compliance

This product has been determined to be compliant with the applicable standards, regulations, and directives for the countries where the product is marketed. The product is affixed with regulatory marking and text as necessary for the country/agency. Generally, Information Technology Equipment (ITE) product compliance is based on IEC and CISPR standards and their national equivalent such as Product Safety, IEC 60950-1 and European Norm EN 60950-1 or EMC, CISPR 22/CISPR 24 and EN 55022/55024. Dell products have been verified to comply with the EU RoHS Directive 2011/65/EU. Dell products do not contain any of the restricted substances in concentrations and applications not permitted by the RoHS Directive.

9 Self Tests

9.1 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 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.

9.1.1 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.

9.1.2 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
Triple-DES-CMAC	KAT
AES-CMAC	KAT
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
SHA3-224	KAT
SHA3-256	KAT
SHA3-384	KAT
SHA3-512	KAT
SHAKE-128	KAT
SHAKE-256	KAT
HMAC SHA-1	KAT
HMAC SHA-224	KAT
HMAC SHA-256	KAT
HMAC SHA-384	KAT
HMAC SHA-512	KAT
HMAC-SHA3-224	KAT

Algorithm	Tests
HMAC-SHA3-256	KAT
HMAC-SHA3-384	KAT
HMAC-SHA3-512	KAT
DRBG (Hash, HMAC and CTR-based)	KAT
DRBG	Health test per section 11.3 of SP 800-90A DRBG
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

9.2 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.

9.3 Conditional Tests

The module only performs conditional tests when asymmetric key pairs are generated:

Algorithm	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 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())
ECDSA	The test uses a random number of the size of the nbits 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/ecc.c:test_keys())

Table 9: Conditional Tests

10 Guidance

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

10.1 Crypto Officer Guidance

The version of the RPMs containing the FIPS validated Module is stated in section 1 above.

The RPM package of the Module can be installed by standard tools recommended for the installation of RPM packages on a Red Hat Enterprise Linux system (for example, yum, rpm, and the RHN remote management tool).

1.1.1 The ciphers listed in Table 6 are not allowed to be used.

10.1.1 FIPS module installation instructions

10.1.1.1 Recommended method

The system-wide cryptographic policies package (crypto-policies) contains a tool that completes the installation of cryptographic modules and enables self-checks in accordance with the requirements of Federal Information Processing Standard (FIPS) Publication 140-2. We call this step "FIPS enablement". The tool named fips-mode-setup installs and enables or disables all the validated FIPS modules and it is the recommended method to install and configure a RHEL-8 system.

1. To switch the system to FIPS eablement in RHEL 8:

```
# fips-mode-setup --enable

Setting system policy to FIPS

FIPS mode will be enabled.

Please reboot the system for the setting to take effect.
```

2. Restart your system:

```
# reboot
```

3. After the restart, you can check the current state:

```
# fips-mode-setup --check
FIPS mode         is enabled.
```

Note: As a side effect of the enablement procedure the fips-mode-enable tool also changes the system-wide cryptographic policy level to a level named "FIPS", this level helps applications by changing configuration defaults to approved algorithms.

10.1.1.2 Manual method

The recommended method automatically performs all the necessary steps.

The following steps can be done manually but are not recommended and are not required if the systems has been installed with the fips-mode-setup tool:

- create a file named /etc/system-fips, the contents of this file are never checked
- ensure to invoke the command '`fips-finish-install --complete`' on the installed system.
- ensure that the kernel boot line is configured with the fips=1 parameter set

- Reboot the system

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

```
$ df |grep boot
```

```
/dev/sda1    233191    30454    190296    14%    /boot
```

The partition of the `/boot` file system is located on `/dev/sda1` in this example.

Therefore the parameter `boot=/dev/sda1` needs to be appended to the kernel command line in addition to the parameter `fips=1`.

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-mode-setup —-enable` command, as described above.

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.

If an application wants to explicitly request and force FIPS mode, it should 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.

10.2 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
```

10.2.1 Three-key Triple-DES

According to IG A.13, the same Triple-DES key shall not be used to encrypt more than 2^{16} 64-bit blocks of data. It is the user’s responsibility to make sure that the module complies with this requirement and that the module does not exceed this limit.

10.2.2 AES-XTS Guidance

The length of a single data unit encrypted or decrypted with the XTS-AES shall not exceed 2^{20} AES blocks that is 16MB of data per AES-XTS instance. An XTS instance is defined in section 4 of SP 800-38E.

The module implements a check to ensure that the two AES keys used in XTS-AES algorithm are not identical.

The AES-XTS mode shall only be used for the cryptographic protection of data on storage devices. The AES-XTS shall not be used for other purposes, such as the encryption of data in transit.

10.2.3 Key derivation using SP800-132 PBKDF

The module provides password-based key derivation (PBKDF), compliant with SP800-132. The module supports option 1a from section 5.4 of [SP800-132], in which the Master Key (MK) or a segment of it is used directly as the Data Protection Key (DPK).

In accordance to [SP800-132] and IG D.6, the following requirements shall be met.

- Derived keys shall only be used in storage applications. The Master Key (MK) shall not be used for other purposes. The length of the MK or DPK shall be of 112 bits or more.
- A portion of the salt, with a length of at least 128 bits, shall be generated randomly using the SP800-90A DRBG,
- The iteration count shall be selected as large as possible, as long as the time required to generate the key using the entered password is acceptable for the users. The minimum value shall be 1000.
- Passwords or passphrases, used as an input for the PBKDF, shall not be used as cryptographic keys.
- The length of the password or passphrase shall be of at least 20 characters, and shall consist of lower-case, upper-case and numeric characters. The probability of guessing the value is estimated to be $1/62^{20} = 10^{-36}$, which is less than 2^{-112}.

The calling application shall also observe the rest of the requirements and recommendations specified in [SP800-132].

11 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^{e} mod n) is decrypted and the unblinded value (x' = y'·r^{-1} mod n) returned. The blinding value “r” is a random value with the size of the modulus “n” and generated with `GCRY_WEAK_RANDOM' random level.

Weak Triple-DES keys are detected as follows:

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

```
static byte weak_keys[64][8] =
{
  { 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00 }, /*w*/
  { 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
CPACF	Central Processor Assist for Cryptographic Functions
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)** August 2015 https://nvlpubs.nist.gov/nistpubs/SpecialPublications/NIST.SP.800-90Ar1.pdf
FIPS186-4	**Digital Signature Standard (DSS)** July 2013 https://nvlpubs.nist.gov/nistpubs/FIPS/NIST.FIPS.186-4.pdf
FIPS197	**Advanced Encryption Standard** November 2001 https://nvlpubs.nist.gov/nistpubs/FIPS/NIST.FIPS.197.pdf
FIPS198-1	**The Keyed Hash Message Authentication Code (HMAC)** July 2008 https://nvlpubs.nist.gov/nistpubs/FIPS/NIST.FIPS.198-1.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 https://nvlpubs.nist.gov/nistpubs/Legacy/SP/nistspecialpublication800-38a.pdf
SP800-38B	**NIST Special Publication 800-38B - Recommendation for Block Cipher Modes of Operation: The CMAC Mode for Authentication** May 2005 https://nvlpubs.nist.gov/nistpubs/SpecialPublications/NIST.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** July 2007 https://nvlpubs.nist.gov/nistpubs/Legacy/SP/nistspecialpublication800-38c.pdf
SP800-38D	**NIST Special Publication 800-38D - Recommendation for Block Cipher Modes of Operation: Galois/Counter Mode (GCM) and GMAC** November 2007 https://nvlpubs.nist.gov/nistpubs/Legacy/SP/nistspecialpublication800-38d.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 https://nvlpubs.nist.gov/nistpubs/Legacy/SP/nistspecialpublication800-38e.pdf
SP800-38F	**NIST Special Publication 800-38F - Recommendation for Block Cipher Modes of Operation: Methods for Key Wrapping** December 2012 https://nvlpubs.nist.gov/nistpubs/SpecialPublications/NIST.SP.800-38F.pdf

SP800-56A	**NIST Special Publication 800-56C Revision 3 - Recommendation for Pair Wise Key Establishment Schemes Using Discrete Logarithm Cryptography** April 2018 https://nvlpubs.nist.gov/nistpubs/SpecialPublications/NIST.SP.800-56Ar3.pdf
SP800-56C	**NIST Special Publication 800-56A Revision 1 - Recommendation for Key Derivation in Key-Establishment Schemes** April 2018 https://nvlpubs.nist.gov/nistpubs/SpecialPublications/NIST.SP.800-56Cr1.pdf
SP800-67	**NIST Special Publication 800-67 Revision 2 - Recommendation for the Triple Data Encryption Algorithm (TDEA) Block Cipher** November 2017 https://nvlpubs.nist.gov/nistpubs/SpecialPublications/NIST.SP.800-67r2.pdf
SP800-90A	**NIST Special Publication 800-90A Revision 1- Recommendation for Random Number Generation Using Deterministic Random Bit Generators** June 2015 https://nvlpubs.nist.gov/nistpubs/SpecialPublications/NIST.SP.800-90Ar1.pdf
SP800-90B	**NIST Special Publication 800-90B - Recommendation for the Entropy Sources Used for Random Bit Generation** January 2018 https://nvlpubs.nist.gov/nistpubs/SpecialPublications/NIST.SP.800-90B.pdf
SP800-108	**NIST Special Publication 800-108 - Recommendation for Key Derivation Using Pseudorandom Functions** October 2009 https://nvlpubs.nist.gov/nistpubs/Legacy/SP/nistspecialpublication800-108.pdf
SP800-131A	**NIST Special Publication 800-131A Revision 2 - Transitions: Recommendation for Transitioning the Use of Cryptographic Algorithms and Key Lengths** March 2019 https://nvlpubs.nist.gov/nistpubs/SpecialPublications/NIST.SP.800-131Ar2.pdf