ESCRYPT's CycurLIB Version 3.5.3-FIPS-1.2 FIPS 140-2 Non-Proprietary Security Policy Deployed by Landis+Gyr

Table of Contents

List of Figures

List of Tables

1. Introduction

1.1 Purpose

This document is the non-proprietary security policy of escrypt's CycurLIB version 3.5.3-FIPS-1.2 module, hereafter referred to as the "Module". It describes the Module and the FIPS 140-2 cryptographic services it provides.

This document was prepared in fulfillment of the FIPS 140-2 requirements for cryptographic modules and is intended for security officers and end-users.

1.2 Module Description

The Module is a software library providing a C-language Application Program Interfaces (APIs) for use by other processes that require cryptographic functionality. The Module is a software module, a multiple-chip standalone module embodiment as classified by FIPS140-2.

The cryptographic services offered by the Module are:

- Data encryption/decryption.
- Generation of hash values.
- Message authentication.
- Signature generation/verification.
- Random number generation.
- Key generation.
- Key agreement and key derivation.
- Key wrapping.

1.3 FIPS 140-2 Validation Scope

Table 1 shows the security level for each of the eleven required areas of the validation.

Table 1. Module Validation Scope

FIPS 140-2 Security Requirement Area	Level
Cryptographic Module Specification	1
Cryptographic Module Ports and Interfaces	1
Roles, Services and Authentication	1
Finite State Machine Model	1
Physical Security	NA
Operational Environment	1
Cryptographic Key Management	1
EMI/EMC	3
Self-Tests	1
Design Assurance	3
Mitigation of Other Attacks	NA

2. Cryptographic Module Specifications

2.1 Logical and Physical Boundaries

The logical cryptographic boundary of the Module consists of the shared library file named "libfips.so". The physical boundary of the Module is the solid enclosure of the host system running the Module. Figure 1 shows the logical and physical boundaries of the Module executing in the memory of a host system.

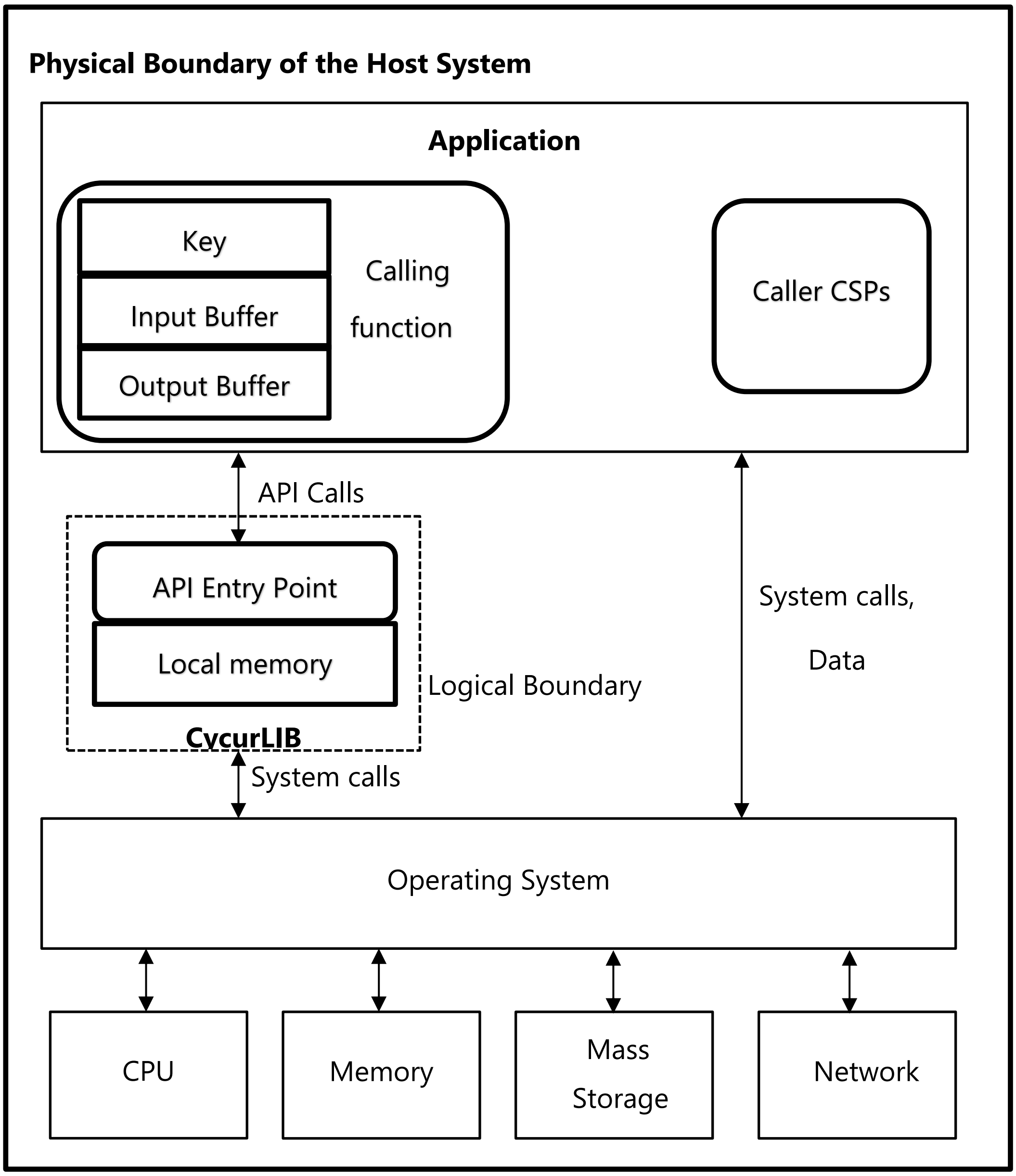

Figure 1. Module Block Diagram

2.2 Modes of Operation

The Module supports a FIPS 140-2 approved mode of operation and operates in this mode by default. If the Module is successfully loaded in memory, i.e., without any errors, this means that all self-tests have been executed successfully and the Module is in the FIPS140-2 approved mode. If the module encounters an error, the module enters an error state and is no longer in the approved mode. In the error state, no output from the Module is allowed as every cryptographic function in the Module checks the status of the Module before its execution. In the error state, no cryptographic function in the Module will be executed. In addition, the operator of the module must adhere to all rules in 10.4.1 and 10.4.2 and if not, the module is also considered to be in a non-approved mode of operation.

2.3 Approved Security Functions

The Module supports only a FIPS140-2 approved mode. Table 2 lists all the approved functions included in the Module.

Table 2. Approved Algorithms

Function	**Algorithm**	**Options**	**Cert. No.**
Symmetric Encryption/Decryption	AES	[FIPS 197] 128/192/256 ECB, CBC, [SP 800-38C] CCM, CTR, [SP 800-38D] GCM, GMAC. [SP 800-38E] 128/256 XTS	C971
	[SP 800-67 Rev1] Triple-DES	3-key CBC	C971
Key Wrap	[SP800-38F] AES	128/192/256 Key establishment methodology provides between 128 and 256 bits of encryption strength	C971
Message	AES	128/192/256	C971

Authentication Code (CMAC) [SP 800-38B]	Triple-DES	3-key	C971
Message Digest	[FIPS180]	SHA-1, SHA-224, SHA-256, SHA-384, SHA-512	C971
	[FIPS202]	SHA3-224, SHA3-256, SHA3-384, SHA3-512	C971
Keyed Hash [FIPS 198-1]	HMAC	SHA-1, SHA2-224, SHA2-256	C971
Digital Signature and key generation	[FIPS186-4] ECDSA	• Supported curves (P-224/P-256/ P-384/ P-521) • Public key validation • Key pair generation • Signature generation • Signature verification	C971
	[FIPS186-4] RSA	• RSASSA-PKCS1-v1_5 signature generation 2048/3072/4096[1] with SHA-224/SHA-256/SHA-384/SHA-512 and 2048/3072 with SHA-1 • RSASSA-PSS signature generation 2048/3072/4096[1] with SHA-224/SHA-256/SHA-384/SHA-512 and 2048/3072 with SHA-1 • RSASSA-PKCS1-v1_5 signature verification 1024/2048/3072/4096[1] with SHA-1/SHA-224/SHA-256/SHA-384/SHA-512 • RSASSA-PSS signature verification 1024/2048/3072/4096[1] with SHA-1/SHA-224/SHA-256/SHA-384/SHA-512	C971
	[FIPS 186-4] DSA	• DSA key generation for key length 2048 bits • Used in Ephemeral Diffie-	C971

[1] Although RSA 4096 bit modulus size is not tested under C971, it is approved for use under IG A.14.

		Hellman key agreement scheme	
Key agreement	[SP800-56A Rev2] ECC Diffie-Hellman	Ephemeral ECC Diffie-Hellman P-224 (SHA-256, HMAC) P-256 (SHA-256, HMAC) P-384 (SHA-256, HMAC) P-521 (SHA-256, HMAC) Key establishment methodology provides between 112 and 256 bits of encryption strength	Vendor Affirmed
	[SP800-56A Rev2] Diffie-Hellman	Ephemeral Diffie-Hellman: FB: Len P – 2048, Len q – 224 SHA-256, HMAC Key establishment methodology provides 112 bits of encryption strength	Vendor Affirmed
	[SP 800-56A Rev2] ECC Diffie-Hellman CVL	KAS-ECC CDH Primitive P-224 P-256 P-384 P-521	C971
Key derivation	ECC Diffie-Hellman KDA [SP800-56C Rev1]	Hash-KDF with SHA-256	Vendor Affirmed
	Diffie-Hellman KDA [SP800-56C Rev1]	Hash-KDF with SHA-256	Vendor Affirmed
Asymmetric decryption	RSADP CVL [SP800-56B]	RSADP primitive with 2048-bit modulus	C971
Random number generator	[SP800-90A] DRBG	Hash DRBG with SHA-256	C971
Key generation	CKG [SP 800-133]	N/A	Vendor

			Affirmed

Table 3. Allowed Algorithms

Function	**Algorithm**	**Options**	**Cert. No.**
Random number generator (for seeding the DRBG)	NDRNG	Entropy of at least 256 bits	

3. Cryptographic Module Ports and Interfaces

The physical ports of the Module are the ports of the underlying embedded system hosting the Module execution. The Module logical interfaces are C-language APIs which can be categorized into the following:

- Data input interface.
- Data output interface.
- Control input interface.
- Status output interface.

The description of these logical interfaces is given in the following table.

Table 4. Logical interfaces

Logical interface type	**Description**
Data input	The input variables passed as arguments to the APIs of the Module.
Data output	The variables passed as output arguments by the APIs of the Module to the calling application.
Control input	The configuration variables of the Module which control its cryptographic functions mode of operation.

Status output	The return codes and messages.

The data input is logically separated from the control input because the data input is passed by the input variables to the Module while the control input is passed in a configuration file. Also, the data output is logically separated from the status output because the data output is passed out by the Module's API arguments while the status output is passed by return codes and messages.

Since the Module is a software module, the control of the physical ports is outside the scope of the Module. When the Module is performing self-tests, or is in error-state, all outputs on the logical data output interface are inhibited. The Module is single-threaded and, in error scenarios, returns only an error value.

Since the Module is single-threaded, there is no data output passed by the Module during key generation and zeroization of cryptographic keys.

4. Roles, Services, and Authentication

4.1 Roles

The Module supports a Crypto Officer (CO) role and a User (U) role as required by FIPS 140-2. The Module does not support authentication mechanism for these roles but rather relies on the access control mechanisms of the underlying operation system. Hence, these roles are implicitly assumed by the entity accessing the Module. Moreover, the Module does not support concurrent users.

4.2 Operator Services and Descriptions

The services available to the User (U) and Crypto Officer (CO) are indicated in Table 5 below, along with the type of CSP access per service:

R = Read, W = Write, X = Execute, Z = Zeroize

Table 5. Services and CSP access

Service	Description	Key and CSP(s)	Roles	Type of access
Installation	Module Installation	None	U, CO	X
Self-Test	Initiate power-on self-tests	None	U, CO	R, X
Show Status	Provides Module status information	None	U, CO	R, X
Symmetric Encryption/Decryption	Encrypt/Decrypt blocks of data using AES or Triple- DES	AES key or Triple-DES 3 keys.	U, CO	R, W, X
Asymmetric Key	Generate Diffie-Hellman	Diffie-Hellman and	U, CO	R, W, X

Generation	and ECC Diffie-Hellman keys	ECC Diffie-Hellman keys		
Digital Signature Generation/Verification	Sign/Verify a message digest	RSA and ECDSA keys.	U, CO	R, W, X
Message Digest	Hash a block of data	None	U, CO	R, W, X
Keyed Message digest (HMAC)	Generate/Verify a signature on a block of data	HMAC key	U, CO	R, W, X
Keyed Message digest (CMAC)	Generate/Verify a signature on a block of data	AES key or Triple-DES 3 keys.	U, CO	R, W, X
Random Number Generator	Generate random numbers and keys	Random numbers and keys	U, CO	R, X, Z
Key Agreement	Calculate a shared key between two parties	Diffie-Hellman and ECC Diffie-Hellman private keys and shared keys	U, CO	R, W, X
Key Transport	Encrypt/decrypt a key.	AES key encryption key and key to be encrypted/ decrypted.	U, CO	R, W, X
Zeroization	All functions automatically overwrites CSPs stored in the functions stack frame. Each module provides a function for the user to zeroize the module context (created in the calling application's own	All keys and CSPs	U, CO	X, Z

	stack). Other CSPs provided as arguments to the functions are the responsibility of the calling application.			

4.3 Authentication

The Module does not support operator authentication mechanisms. The Module relies on the operating system for any operator authentication.

5. Physical Security

The Module is a software library and is intended to operate on a host system that has a production grade components and enclosure. The Module does not claim any physical security.

6. Operational Environment

The module operates in a modifiable operational environment per FIPS140-2 level 1 specifications. The Module was tested in a Linux 4.9 operating environment running on a Landis+Gyr Network Bridge N2250 with an ARM Cortex-A5. When the Module is loaded into memory by the operating system, FIPS Power-On Self-Test (POST) functionality along with any other mandatory FIPS tests are invoked.

The Module relies on the underlying operating system built-in kernel functionality to ensure that user access rights are in place for accessing the module and that one user accesses the library at any given time.

The underlying operating system also ensures process isolation in terms of execution and memory space. Moreover, the Module, being a software-only module, consists of an executable binary that does not store process context or state. Each process manages its own stack and data space.

7. Cryptographic Key Management

All the keys and Critical Security Parameters (CSPs) used by the Module are indicated in Table 6 below. The CSP names are generic and they correspond to API parameter data structures.

Table 6. Critical security parameters

CSP Name	Generation/Input	Use
AES (ECB, CBC, CCM, CTR, GCM) key (128, 192, 256 bits)	The key is passed to the Module via API input	- Encrypt/Decrypt operations. - Authenticated encryption/decryption in CCM, GCM
AES Key Wrap key (128, 192, 256 bits)	The keys are passed to the Module via API input	- Key Encrypt/Decrypt operations.
AES XTS keys (128, 256 bits)	The keys are passed to the Module via API input	- Encrypt/Decrypt operations.
Triple-DES (CBC) 3-key	The keys are passed to the Module via API input	- Encrypt/Decrypt operations.
CMAC (AES, Triple-DES) key	The keys are passed to the Module via API input	- Used to generate and verify MAC's with AES or Triple-DES as part of the CMAC algorithm.
HMAC (SHA-1, SHA-224, SHA-256) key	The key is passed to the Module via API input	- Used to generate and verify MAC as part of the HMAC algorithm.
ECDSA/ECC Diffie-Hellman key pair (P-224, P-256, P-384, P-521)	Keys are generated as per FIPS 186-4 and the random value used in the key generation is generated using Hash-DRBG as per SP800-90A	- ECDSA public/private keys used to sign and verify data. - ECC Diffie-Hellman key pair used as part of the key agreement protocol.

Diffie-Hellman key pair (2048 bits)	Keys are generated as per FIPS 186-4 and the random value used in the key generation is generated using Hash-DRBG as per SP800- 90A	- Diffie-Hellman key pair used as part of the key agreement protocol.
RSASSA (PKCS1-v1_5/PSS) key pair (2048, 3072, 4096 bits)	The keys are passed to the Module via API input	- RSA public/private keys used to sign and verify data.
DRBG seed	Seed is obtained internally from NDRNG	- Used as inputs to the DRBG internal functions.
DRBG input string	Input string is obtained internally form NDRNG	- Used as inputs to the DRBG internal functions.
DRBG V	DRBG V is generated and updated internally.	- Used to maintain the secret internal state of the DRBG.
DRBG C	DRBG C is generated and updated internally.	- Used to maintain the secret internal state of the DRBG.
Software Integrity key	HMAC SHA-256 key stored within the module, protected by the operating system access control mechanism.	- Used first by the make build system to create an HMAC tag on the module binary file and store it in a separate file. - When the module is loaded, its integrity is checked by calculating the HMAC tag on the module binary file and comparing it with the one stored at build time.

The following discussion in this section applies to all the CSPs listed in Table 6.

7.1 Random Number Generation

Entropy source:

- The Module implements NDRNG using "/dev/random" as the source of entropy.
- The Module relies on the underlying Linux kernel which is within the physical boundary of the Module but outside its logical boundary.
- The NDRNG provides at least 256 bits of entropy to the DRBG.
- As part of the conditional Self-Test, the Module performs Continuous Random Number Generator (CRNG) test to ensure that consecutive random numbers in the NDRNG are not repeated.

DRBG:

- The Module has an SP800-90A-compliant Hash-DRBG which is seeded from the NDRNG to generate keys and random numbers.
- Continuous health tests are performed on the DRBG whenever it is invoked by any function in the Module.

7.2 Key Generation

- The Module offers a DRBG seeded with 256-bit entropy and is SP-800-90A compliant of security strength 256 bits. Hence, it can be used to generate symmetric keys.
- The Module generates Diffie-Hellman, ECC Diffie-Hellman and ECDSA key pairs in compliance with FIPS 186-4 where the private keys are generated by an SP800-90A-compliant DRBG.
- The resulting symmetric key or a generated seed is an unmodified output from a DRBG.
- Keys residing in internally allocated data structures (during the lifetime of an API call) can only be accessed using the Module defined API.
- The DRBG is seeded from the NDRNG.
- The user can choose to use a single or multiple DRBGs to generate the different keys
- The calling application is responsible for storing the generated keys.
- No intermediate key generation values are output from the Module.
- Intermediate values computed by the module are zeroized before an API function returns its output.
- For GCM, the module generates the IV randomly from the DRBG and the IV length is 96 bits.

7.3 Key Entry

The Module takes CSPs as API input parameters designated by memory location:

- For symmetric-key algorithms or for HMAC, the keys are provided to the module via API plaintext input parameters for the cryptographic operations.

- Public-key key pairs are provided in pertinent data structures that are generated, formatted by other API functions.
- Keys residing in internally allocated data structures (during the lifetime of an API call) can only be accessed using the Module defined API.
- None of the entered CSPs cross the physical boundary of the Module.

7.4 Key Output

The Module does not output CSPs, other than the explicit results of the key generation services. Those CSPs are considered ephemeral. None of the generated CSPs cross the physical boundary of the Module.

7.5 Key Establishment

The Module supports the [SP800-56A Rev2] Diffie-Hellman with at least 2048 bits key size and ECC Diffie-Hellman with P-224, P-256, P-384, or P-521 curve. The Module performs key derivation directly on the shared secret established by Diffie-Hellman or ECC Diffie-Hellman and only outputs the derived key. The Module provides key establishment service using the AES Key Wrapping algorithm.

7.6 Key Storage

The module does not support persistent key storage. The keys and CSPs are stored as plaintext in the RAM. The keys are provided to the module via API input parameters, and are destroyed by the module using appropriate API function calls before they are released in the memory.

The HMAC key used for the Module’s integrity test is stored in the Module and relies on the operating system for protection.

7.7 Key Zeroization

- Key zeroization is performed by filling the corresponding memory location by zeros.
- Zeroization of sensitive data is performed automatically by API function calls for temporarily stored CSPs before the function returns.
- In addition, the Module provides functions to explicitly destroy CSPs related to random number generation services and API-specific data structures (function contexts).

- The calling application is responsible for parameters passed in and out of the Module.

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

The EMI/EMC properties of the Module is not pertinent to this policy as the Module is a software library.

9. Self-Tests

As required by FIPS 140-2, upon loading the Module in memory, the Module performs power-on self-tests to ensure its integrity and the correctness of its supported cryptographic functions. Moreover, conditional self-tests are performed upon calling some cryptographic functions as specified in FIPS140-2 to ensure their correctness. The power-on self-tests and the conditional self-tests are indicated below.

9.1 Power-ON Self-Tests (POST)

- The Module performs power-on self-tests automatically upon loading the Module in memory by making use of the default entry point and requires no operator intervention.
- The Module does not provide any output and it is not available during POST.
- The integrity of the Module is checked using HMAC SHA-256. The HMAC key is stored within the Module. The HMAC value is calculated at the build time and stored in a binary file. The run-time HMAC calculated value is compared to that stored in the HMAC file during the integrity test. If a match exists, then the rest of POST is performed.
- If POST is successful, the module returns a status of "Esc_FIPS_MODE_AVAILABLE" and it becomes operational and the cryptographic services become available.
- If the Module integrity check test or any of the tests in POST fail, the module returns an error code of "Esc_FIPS_MODE_DISABLED" and it goes into an error state where none of the cryptographic functions will be available.

Table 7 shows a list of the performed POST:

Table 7. Power-On Self-Tests (POST)

KAT = Known Answer Test, PCT = Pairwise Consistency Test.

Algorithm	**Type**	**Description**
Software integrity check	KAT	HMAC-SHA256
AES ECB	KAT	Separate encrypt and decrypt, 128 bits key length
AES CCM	KAT	Separate encrypt and decrypt, 256 bits key length
AES GCM	KAT	Separate encrypt and decrypt, 128 bits key length

AES XTS	KAT	Separate encrypt and decrypt, 128 bits key length
AES Key Wrap	KAT	Separate encrypt and decrypt, 128 bits key length
AES CMAC	KAT	Separate sign and verify, ECB mode, 256 bits key length
Triple-DES CMAC	KAT	Separate sign and verify, 256 bits key length
Triple DES	KAT	Separate encrypt and decrypt, CBC mode, 3-key
SHA-1	KAT	SHA-1
SHA-2	KAT	SHA-224/256/384/512
SHA-3	KAT	SHA-3: 224/256/384/512
HMAC	KAT	SHA1, SHA224, SHA256
Diffie-Hellman	KAT	Primitive "Z", 2048 bits key length
ECC Diffie-Hellman	KAT	Primitive "Z", 256 bits key length
ECDSA	PCT	Sign and verify, P-256
DRBG	KAT	HashDrbg: SHA256
RSA PKCS1	KAT	Separate sign and verify, 2048 bits key length
RSA PSS	KAT	Separate sign and verify, 4096 bits key length

- POST can be executed on demand at any time or to exit from "Esc_FIPS_MODE_DISABLED" status by calling EscFipsPost_Run( ), which returns "Esc_NO_ERROR" (defined as 0) in case of success, and a numeric error code otherwise.
- In all the Known-Answer Tests (KAT) listed in Table 7, the calculated answer is compared byte-by-byte with the expected answer. If there is a match, then test passes, otherwise, the test fails. If the test fails, the Module will enter in the error state "Esc_FIPS_MODE_DISABLED" which will render the Module unusable.

9.2 Conditional Self-Tests (CST)

Conditional Self-Tests are performed by the Module when some cryptographic functions are called as per FIPS 140-2 requirements. If any CST fails, the corresponding function returns an error and it does not produce the required output. Table 8 shows a list of the implemented CST.

Table 8. Conditional Self-Tests (CST)

Algorithm	Test
DRBG	SP800-90A Health Tests
ECC (ECDSA and ECC Diffie-Hellman)	Pairwise Consistency Test (sign/verify)
RSA	Pairwise Consistency Test (sign/verify)
Diffie-Hellman and ECC	SP800-56A Rev2 Assurance Test
NDRNG	Continuous Random Number Generator Test
AES-XTS	Key Test (IG A.9)

10. Design Assurance

10.1 Configuration Management

Escrypt hosts its own subversion (SVN) as a Version Control System (VCS) to manage and record the source code and associated documentation files. SVN provides a unique identification for each submitted version of any component of the Module and its documentation.

Document management utilities provide access control, versioning, and logging. Access to the SVN repository is granted or denied to Escrypt team members by the server administrator based on the vendor policy.

10.2 Delivery and Operation

10.2.1 Delivery

The Module will be delivered to the customer as a Software shared library binary file where the integrity of the Module is checked using HMAC SHA-256. The HMAC key is stored within the Module. The calculated HMAC value at the build time is stored in the Module. During run-time, an HMAC value is calculated and compared to that stored in the Module during the integrity test performed whenever the Module is loaded in memory. If a match exists, then the Module will be usable, otherwise the Module will not be usable.

10.2.2 Operation

The Module is a Software shared library:

- It does not require installation. It only needs to reside in a path accessible to the loader together with its signature file. See Sec.10.4.2 User Guidance.
- A user application would use the library by invoking any of the FIPS API functions as documented in the Modules section of the Doxygen documentation.
- The invocation of any of the Module's functions for the first time results in the kernel loading it into memory.
- Once loaded, the Power-On Self-Tests (POST) begin execution. See Sec.9.1 Power-ON Self-Tests (POST).
- Upon successful return of POST, the Module is available from then on for any subsequent invocation of a FIPS API function.

10.3 Development

The Module went through frequent builds. Extensive testing is done on each build including parameter testing, compliance testing, KAT consistency testing, POST, conditional self-testing and Cryptographic Algorithm Validation Program (CAVP) testing.

10.4 Guidance Documents

The following guidance items are to be used for assistance to maintain the FIPS-approved mode while in use.

10.4.1 Cryptographic Officer Guidance

The FIPS-approved mode is configured by default in Module. If the Module loads without any errors, this means that all the self-tests have been executed successfully and the Module is in the FIPS140-2 approved mode. Otherwise, the Module will be in error state and no FIPS API cryptographic function can be executed. If in error state, the host system should be restarted or POST re-executed by calling EscFipsPost_Run( ).

10.4.2 User Guidance

The FIPS-approved mode is configured by default in Module. If the Module loads without any errors, this means that all the self-tests have been executed successfully and the Module is in the FIPS140-2 approved mode. Otherwise, the Module will be in error state and no FIPS API cryptographic function can be executed. If in error state, the host system should be restarted or POST re-executed by calling EscFipsPost_Run( ).

Users of the Module should adhere to the following:

- Keeping the HMAC signature file `libfips_hmac.bin` with the shared library file `libfips.so` in the same directory.
- Setting the load path to the directory where both files reside, for example by running:
  ```
  $ export
  LD_LIBRARY_PATH=<path/to/shared_lib>:$LD_LIBRARY_PATH
  ```
- Always checking that the return status of a function from the FIPS API is `Esc_NO_ERROR` before collecting its output and using it.
- The user of the Module should use the Module and its FIPS APIs without any modifications, otherwise; the Module is not considered a validated FIPS140-2 module.
- The utilized HMAC keys must be at least 112 bits.
- AES-XTS must only be used in storage applications.
- When using Triple-DES, the number of encryptions with the same key must be less than 2^16 (IG A.13).

If the above rules are not followed, the module is not considered to be operating in an approved state.

11. Mitigation of Other Attacks

The Module is not designed to mitigate against attacks which are outside the scope of FIPS 140-2.