# FIPS 140-2 Security Policy
# for
# Centrify Cryptographic Module

Software Module Version: 2.0, 2.1

FIPS Security Level: 1
Document Version: 1.5
Date: August 15, 2018

 

# Table of Contents

# List of Tables



# List of Figures



 

# 1 Introduction

## 1.1 Purpose

This non-proprietary Security Policy for the Centrify Cryptographic Module version 2.0 and 2.1 by Centrify Corporation describes how the module meets the security requirements of FIPS 140-2 and how to run the module in a secure FIPS 140-2 mode.

This document was prepared as part of the Level 1 FIPS 140-2 validation of this cryptographic module. The following table lists the module's FIPS 140-2 security level for each section.

| Section | Section Title | 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 | N/A |

**Table 1 – FIPS 140-2 Section Security Levels**

## 1.2 Background

Federal Information Processing Standards Publication (FIPS PUB) 140-2 – *Security Requirements for Cryptographic Modules* details the requirements for cryptographic modules. More information on the National Institute of Standards and Technology (NIST) and the Communications Security Establishment Canada (CSEC) Cryptographic Module Validation Program (CMVP), the FIPS 140-2 validation process, and a list of validated cryptographic modules can be found on the CMVP website:

http://csrc.nist.gov/groups/STM/cmvp/index.html

More information about Centrify products can be found on the Centrify website:

http://www.centrify.com

### 1.3 Document Organization

This non-proprietary Security Policy is part of the Centrify Cryptographic Module version 2.0 and 2.1 FIPS 140-2 submission package. Other documentation in the submission package includes:

- Product documentation
- Vendor evidence documents
- Finite state model
- Additional supporting documents

The Centrify Cryptographic module is also referred to in this document as the cryptographic module, or the module.

# 2 Module Overview

The Centrify Cryptographic Module version 2.0 and 2.1 provides cryptographic functionality to Centrify software applications. For the purposes of FIPS 140-2, the module is classified as a software module.

## 2.1 Cryptographic Module Specification

The cryptographic module is installed into a General Purpose Computer. Since the computer where the module is installed is a multi-chip standalone device, the module is classified as a multi-chip standalone software module. The General Purpose Computer provides the physical cryptographic boundary. The Centrify Cryptographic Module forms the logical boundary.

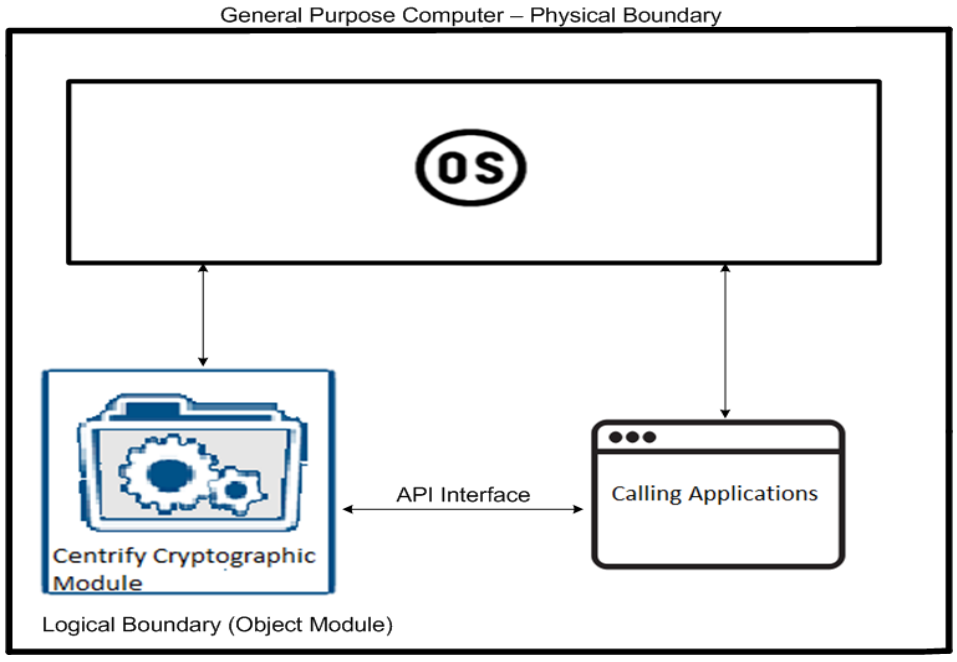


***Figure 1 – Physical and Logical Boundary***

## 2.2 Cryptographic Module Ports and Interfaces

The physical ports are provided by the General Purpose Computer. The module provides FIPS validated cryptographic functions that are available via a C-language Application Programming Interface (API). This API layer is the logical interface through which the calling applications can utilize the module's services.

Table 2 provides a description of the logical interfaces as well as the mapping of the module's physical interfaces to the logical interfaces defined in FIPS 140-2.

| FIPS 140-2 Interface | Physical Interface | Logical Interface |
|---|---|---|
| Data Input | Keyboard | Input parameters to the API calls. |
| Data Output | Display | Output parameters from API calls. |
| Control Input | Keyboard | API function calls. |
| Status Output | Display | Return values from API calls. |
| Power | General Purpose Computer power | None |

**Table 2 – Module Interface Description and Mappings**

## 2.3 Roles & Services

### 2.3.1 Roles

The module provides two operator roles: Crypto Officer and User.

The Crypto Officer is the system administrator who can install/uninstall, initialize the module and utilize the cryptographic functions provided by the module, while the Users are the calling applications that utilize the cryptographic functions.

### 2.3.2 Services

The following table describes the services that the two operator roles can perform. In the CSP Access column, Read and Execute mean the CSP is used by the API call to perform the service; and Write means the CSP is generated, modified or deleted by the API call.

 

| Service | Operator | Description | Input | Output | CSP | CSP ACCESS |
|---|---|---|---|---|---|---|
| Asymmetric key generation | Crypto Officer, User | Generate and return the specified type of asymmetric key pair | Key size to generate | RSA, DSA or ECDSA public and private key pair | Generated RSA, DSA or ECDSA key pair | Read Write Execute |
| Digital signature | Crypto Officer, User | Generate or verify digital signature | Generate: PlainText, Signing key<br>Verify: Signature, Public Key | Generate: Signature | RSA, DSA and ECDSA private key or public key | Read Write Execute |
| Installation, uninstallation and initialization | Crypto Officier | Install and uninstall the module[1] | N/A | N/A | N/A | N/A |
| Key agreement | Crypto Officer, User | Perform key agreement on behalf of calling process. Not used to establish keys into the module | EC DH public key and private Key | EC DH agreement key | EC DH agreement key | Read Write Execute |
| Key transport | Crypto Officer, User | Encrypt or Decrypt a key value on behalf of the calling process | Encrypt: key value, RSA Key Transport Key<br>Decrypt: Encrypted Key value, RSA Key Transport Key | Encrypt: Encrypted key value<br>Decrypt: key value | RSA Key Transport Key | Read Write Execute |
| Keyed Hash | Crypto Officer, User | Generate or verify data integrity with HMAC | HMAC key and message | Input's digest | HMAC key | Read Write Execute |
| Message digest | Crypto Officer, User | Generate a message digest using a Secure Hash Algorithm | PlainText message | Input's digest | N/A | Read Execute |
| Random number generation | Crypto Officer, User | Generate the specified number of random bits | Initialize: seed value | Random bits | Entropy input string and seed value | Read Write Execute |

[1] The module is installed/un-installed as part of Centrify's product installation/uninstallation.

| Service | Operator | Description | Input | Output | CSP | CSP ACCESS |
|---|---|---|---|---|---|---|
| Run self-tests | Crypto Officer, User | Perform self-tests | N/A | 1 for Success 0 for Failure | N/A | N/A |
| Status | Crypto Officer, User | Display module status | N/A | Version: Display the Module's version | N/A | N/A |
| Crypto-based MAC | Crypto Officer, User | Generate or verify data integrity with CMAC | Plaintext, CMAC key | Generate: Digest | CMAC key | Read Write Execute |
| Symmetric encryption and decryption | Crypto Officer, User | Encrypt plaintext using supplied key and specified algorithm<br>Decrypt ciphertext using supplied key and specified algorithm | Encrypt: Plaintext, Initialization Vector and key<br>Decrypt: Ciphertext, Initialization Vector and key | Encrypt: Ciphertext<br>Decrypt: Plaintext | AES and Triple-DES key | Read Write Execute |
| Zeroize keys | Crypto Officer, User | Zeroize (destroy CSPs) and de-allocate memory | N/A | N/A | All keys and CSPs | Write |

**Table 3 – Services**

## 2.4 Authentication Mechanisms

For Security level 1, no authentication is required. The role is implicitly assumed upon function entry/service invocation.

## 2.5 Physical Security

As a software module, physical security is outside the module's scope.

## 2.6 Operational Environment

The module is invoked and functions entirely within the logical process space of the calling application. The tested operating systems segregate user processes into separate process spaces.

The module is tested in the following multi-chip standalone platforms:

| Module Version | Manufactuer | Model | OS & Version |
|---|---|---|---|
| 2.0 | Apple | MacBook Pro Intel Core i7 without AES-NI (PAA) | Mac OS 10.11.5 |
| 2.0 | SuperMicro | Intel Xeon E5520 x86_64 without AES-NI (PAA) | Red Hat Enterprise Linux 7.2 |
| 2.1 | HP Proliant | Intel Xeon X5650 x86_64 with AES-NI (PAA) | Red Hat Enterprise Linux 7.2 |
| 2.0 | IBM | PowerPC Power7 Processor without AES-NI (PAA) | AIX 7.2 (32-bit)<br>AIX 7.2 (64-bit) |

## 2.7 Cryptographic Key Management

### 2.7.1 Algorithm Implementations

A list of FIPS-Approved algorithms implemented by the module can be found in Table 6.

| Algorithm | Modes and Key Sizes | Validation Number |
|---|---|---|
| AES -- FIPS 197<br>CCM -- SP 800-38C<br>GCM -- SP 800-38D<br>XTS -- SP 800-38E[2] | Encryption and decryption functions using 128/192/256 ECB, CBC, OFB, CFB1, CFB8, CFB128, CTR, XTS; CCM; GCM | #4087, #5181 |
| AES CMAC -- SP 800-38B | Message integrity generation and verification using AES 128/192/256 | #4087, #5181 |
| CTR-based DRBG -- SP 800-90A | Random number generation with AES 128/192/256. No reseed. | #1226, #1956 |
| Hash-based DRBG -- SP 800-90A | Random number generation with SHA-1/224/256/384/512 | #1226, #1956 |
| HMAC-based DRBG -- SP 800-90A | Random number generation with SHA-1/224/256/384/512. No reseed | #1226, #1956 |

[2] AES-XTS must only be used for storage

| Algorithm | Modes and Key Sizes | Validation Number |
|---|---|---|
| DSA -- FIPS186-4 | Digital signature generation/verification and asymmetric key pair generation<br><br>186-4:<br><br>PQGGen:<br>L = 2048, N = 224 SHA: SHA-224, SHA-256, SHA-384, SHA-512<br>L = 2048, N = 256 SHA: SHA-256, SHA-384, SHA-512<br>L = 3072, N = 256 SHA: SHA-256, SHA-384, SHA-512<br><br>PQGVer:<br>L = 1024, N = 160 SHA: SHA-1, SHA-224, SHA-256, SHA-384, SHA-512<br>L = 2048, N = 224 SHA: SHA-224, SHA-256, SHA-384, SHA-512<br>L = 2048, N = 256 SHA: SHA-256, SHA-384, SHA-512<br>L = 3072, N = 256 SHA: SHA-256, SHA-384, SHA-512<br><br>SigGen:<br>L = 2048, N = 224 SHA: SHA-224, SHA-256, SHA-384, SHA-512<br>L = 2048, N = 256 SHA: SHA-224, SHA-256, SHA-384, SHA-512<br>L = 3072, N = 256 SHA: SHA-224, SHA-256, SHA-384, SHA-512<br><br>SigVer:<br>L = 1024, N = 160 SHA: SHA-1, SHA-224, SHA-256, SHA-384, SHA-512<br>L = 2048, N = 224 SHA: SHA-1, SHA-224, SHA-256, SHA-384, SHA-512<br>L = 2048, N = 256 SHA: SHA-1, SHA-224, SHA-256, SHA-384, SHA-512<br>L = 3072, N = 256 SHA: SHA-1, SHA-224, SHA-256, SHA-384, SHA-512<br><br>KeyPair:<br>L = 2048, N = 224<br>L = 2048, N = 256<br>L = 3072, N = 256 | #1110, #1345 |
| CVL | Key agreement and establishment (ECC CDH). Curves: B-233, B-283, B-409, B-571, K-233, K-283, K-409, K-571, P-224, P-256, P-384, P-521 | #903, #1683 |

| Algorithm | Modes and Key Sizes | Validation Number |
|---|---|---|
| ECDSA -- FIPS 186-4 | Digital signature generation/verification and asymmetric key generation.<br><br>186-4:<br><br>Key Pair Generation:<br>Curves: B-233, B-283, B-409, B-571, K-233, K-283, K-409, K-571, P-224, P-256, P-384, P-521<br><br>Public Key Validation:<br>Curves: B-163, B-233, B-283, B-409, B-571, K-163, K-233, K-283, K-409, K-571, P-192, P-224, P-256, P-384, P-521<br><br>Signature Generation:<br>Curves: B-233, B-283, B-409, B-571, K-233, K-283, K-409, K-571, P-224, P-256, P-384, P-521 using SHA-224, SHA-256, SHA-384, SHA-512<br><br>Signature Verification:<br>Curves: B-163, B-233, B-283, B-409, B-571, K-163, K-233, K-283, K-409, K-571, P-192, P-224, P-256, P-384, P-521 using SHA-1, SHA-224, SHA-256, SHA-384, SHA-512 | #923, #1344 |
| HMAC -- FIPS 198-1 | Keyed Hash for message integrity using HMAC-SHA-1, HMAC-SHA-224, HMAC-SHA-256, HMAC-SHA-384, and HMAC-SHA-512 | #2667, #3437 |

| Algorithm | Modes and Key Sizes | Validation Number |
|---|---|---|
| RSA -- FIPS 186-4 | Digital signature generation/verification and asymmetric key generation<br><br>186-2:<br>Signature Verification 9.31:<br>Modulus lengths: 1024, 1536, 2048, 3072, 4096 (bits) using SHA-1, SHA-256, SHA-384, SHA-512<br><br>Signature Verification PKCS1.5:<br>Modulus lengths: 1024, 1536, 2048, 3072, 4096 (bits) using SHA-1, SHA-224, SHA-256, SHA-384, SHA-512<br><br>Signature Verification PSS:<br>Modulus lengths: 1024, 1536, 2048, 3072, 4096 (bits) using SHA-1, SHA-224, SHA-256, SHA-384, SHA-512<br><br>186-4:<br>Signature Generation 9.31:<br>Mod 2048 and 3072-bits using SHA-256, SHA-384, SHA-512<br><br>Signature Generation PKCS1.5:<br>Mod 2048 and 3072-bits using SHA-224, SHA-256, SHA-384, SHA-512<br><br>Signature Generation PSS:<br>Mod 2048 and 3072-bits using SHA-224, SHA-256, SHA-384, SHA-512 | #2212, #2782 |
| SHA-1, SHA-224, SHA-256, SHA-384, SHA-512 -- FIPS 180-4 | Hashing | #3363, #4185 |
| Triple-DES -- SP 800-67 | Encryption and decryption functions using TECB, TCBC, TCFB1, TCFB8, TCFB64, TOFB | #2232, #2636 |
| Triple-DES CMAC -- SP 800-38B | Message integrity generation and verification | #2232, #2636 |

**Table 4 – FIPS-Approved Algorithm Implementations**

The module supports only NIST curves for use with ECDSA and ECC CDH. The module supports two operational environment configurations for elliptic curve; NIST prime curve only and all NIST defined curves.

A list of non-FIPS Approved but allowed algorithms implemented by the module can be found in Table 5.

| Algorithm | Modes and Key Sizes |
|---|---|

 

| | |
|---|---|
| EC DH | Key agreement service provided for use by calling. Non-compliant (untested) EC DH scheme using elliptic curve, supporting all NIST defined B, K and P curves. |
| RSA Key Wrapping | Used by calling applications for key encryption and decryption. No claim is made for SP 800-56B compliance. |
| NDRNG | Used to seed the module's Approved DRBG. The estimated amount of minimum entropy this provides is 184 bits. |

**Table 5 – Non-FIPS Approved But Allowed Algorithm Implementations**

EC DH Key Agreement provides between 112 and 256 bits of encryption strength. RSA Key wrapping provides provides between 112 and 256 bits of encryption strength.

### 2.7.2 Key Management Overview

| Key or CSP | Usage | Storage | Generation | Input | Output | Zeroization |
|---|---|---|---|---|---|---|
| AES EDK | AES (128/192/256 bit) encrypt/decrypt key | Please refer to Section 2.7.5 | Please refer to Section 2.7.3 | Please refer to Section 2.7.3 | Please refer to Section 2.7.4 | Please refer to Section 2.7.6 |
| AES CMAC | AES (128/192/256 bit) CMAC generate/verify key | Please refer to Section 2.7.5 | Please refer to Section 2.7.3 | Please refer to Section 2.7.3 | Please refer to Section 2.7.4 | Please refer to Section 2.7.6 |
| AES GCM | AES (128/192/256 bit) encrypt/decrypt/generate/ verify key | Please refer to Section 2.7.5 | Please refer to Section 2.7.3 | Please refer to Section 2.7.3 | Please refer to Section 2.7.4 | Please refer to Section 2.7.6 |
| AES XTS | AES (256/512) XTS encrypt/decrypt key | Please refer to Section 2.7.5 | Please refer to Section 2.7.3 | Please refer to Section 2.7.3 | Please refer to Section 2.7.4 | Please refer to Section 2.7.6 |
| CTR_DRBG | CTR-based DRBG<br>V: 128 bits<br>Key: AES 128/192/256 bits<br>Entropy input : length dependent on security strength | Please refer to Section 2.7.5 | Please refer to Section 2.7.3 | Please refer to Section 2.7.3 | Please refer to Section 2.7.4 | Please refer to Section 2.7.6 |
| Hash_DRBG | Hash based DRBG<br>V: 440/888 bits<br>C: 440/888 bits<br>Entropy input : length dependent on security strength | Please refer to Section 2.7.5 | Please refer to Section 2.7.3 | Please refer to Section 2.7.3 | Please refer to Section 2.7.4 | Please refer to Section 2.7.6 |

 

| Key or CSP | Usage | Storage | Generation | Input | Output | Zeroization |
|---|---|---|---|---|---|---|
| HMAC_DRBG | HMAC based DRBG<br>V: 160/224/256/284/512 bits<br>Key: 160/224/256/284/512 bits<br>Entropy input : length dependent on security strength | Please refer to Section 2.7.5 | Please refer to Section 2.7.3 | Please refer to Section 2.7.3 | Please refer to Section 2.7.4 | Please refer to Section 2.7.6 |
| DSA SGK | DSA (2048/3072) signature generation key | Please refer to Section 2.7.5 | Please refer to Section 2.7.3 | Please refer to Section 2.7.3 | Please refer to Section 2.7.4 | Please refer to Section 2.7.6 |
| DSA SVK | DSA (1024/2048/3072) signature verification public key | Please refer to Section 2.7.5 | Please refer to Section 2.7.3 | Please refer to Section 2.7.3 | Please refer to Section 2.7.4 | Please refer to Section 2.7.6 |
| ECDSA SGK | ECDSA (All NIST defined B, K and P curves) signature generation key | Please refer to Section 2.7.5 | Please refer to Section 2.7.3 | Please refer to Section 2.7.3 | Please refer to Section 2.7.4 | Please refer to Section 2.7.6 |
| ECDSA SVK | ECDSA (All NIST defined B, K and P curves) signature verification public key | Please refer to Section 2.7.5 | Please refer to Section 2.7.3 | Please refer to Section 2.7.3 | Please refer to Section 2.7.4 | Please refer to Section 2.7.6 |
| EC DH Private | EC DH (All NIST defined B, K and P curves) private key agreement key | Please refer to Section 2.7.5 | Please refer to Section 2.7.3 | Please refer to Section 2.7.3 | Please refer to Section 2.7.4 | Please refer to Section 2.7.6 |
| EC DH Public | EC DH (All NIST defined B, K and P curves) public key agreement key | Please refer to Section 2.7.5 | Please refer to Section 2.7.3 | Please refer to Section 2.7.3 | Please refer to Section 2.7.4 | Please refer to Section 2.7.6 |
| HMAC Key | Keyed hash key (160/224/256/384/512) | Please refer to Section 2.7.5 | Please refer to Section 2.7.3 | Please refer to Section 2.7.3 | Please refer to Section 2.7.4 | Please refer to Section 2.7.6 |
| RSA KDK | RSA (2048-16384 bits) key decryption (private key transport) key | Please refer to Section 2.7.5 | Please refer to Section 2.7.3 | Please refer to Section 2.7.3 | Please refer to Section 2.7.4 | Please refer to Section 2.7.6 |

| Key or CSP | Usage | Storage | Generation | Input | Output | Zeroization |
|---|---|---|---|---|---|---|
| RSA KEK | RSA (2048-16384 bits) key encryption (public key transport) key | Please refer to Section 2.7.5 | Please refer to Section 2.7.3 | Please refer to Section 2.7.3 | Please refer to Section 2.7.4 | Please refer to Section 2.7.6 |
| RSA SGK | RSA (2048 to 16384 bits) signature generation key | Please refer to Section 2.7.5 | Please refer to Section 2.7.3 | Please refer to Section 2.7.3 | Please refer to Section 2.7.4 | Please refer to Section 2.7.6 |
| RSA SVK | RSA (1024 to 16384 bits) signature verification public key | Please refer to Section 2.7.5 | Please refer to Section 2.7.3 | Please refer to Section 2.7.3 | Please refer to Section 2.7.4 | Please refer to Section 2.7.6 |
| Triple-DES CMAC | Triple-DES CMAC generate/verify key | Please refer to Section 2.7.5 | Please refer to Section 2.7.3 | Please refer to Section 2.7.3 | Please refer to Section 2.7.4 | Please refer to Section 2.7.6 |
| Triple-DES EDK | Triple-DES encrypt/decrypt key | Please refer to Section 2.7.5 | Please refer to Section 2.7.3 | Please refer to Section 2.7.3 | Please refer to Section 2.7.4 | Please refer to Section 2.7.6 |

**Table 6 – Cryptographic Keys, Key Components, and CSPs**

### 2.7.3 Key Generation & Input

The module implements SP 800-90A compliant DRBG services for creation of symmetric keys, and for generation of DSA, ECDSA and RSA keys as shown in Table 4.

For random number generation (based on hash functions Hash_DRBG and HMAC_DRBG; and block cipher CTR_DRBG), the calling application should use entropy sources that meet the security strength required in SP 800-90A. This entropy is supplied by means of callback functions. Those functions must return an error if the minimum entropy strength cannot be met.

CSPs are passed to the module in plaintext as API parameters. Private and secret keys as well as seed and entropy are also provided to the module by the calling application.

### 2.7.4 Key Output

The module does not output CSPs, other than the explicit results of key generation requests.

### 2.7.5 Storage

For all CSPs and public keys, storage is in volatile memory. For the DRBGs, the state values are stored only for the lifetime of the DRBG instance. The module uses CSPs passed in by the calling application on the stack. The module does not store any CSPs persistently beyond the lifetime of an API call. The one exception is DRBG state values used for the module's default key generation service, which are stored in volatile memory while the module is operational.

The calling application is responsible for storage of generated keys returned by the module.

### 2.7.6 Zeroization

Zeroization of sensitive data is performed automatically by API function calls for temporarily stored CSPs. There are also functions provided to explicitly destroy CPs related to random number generation services.The calling application is responsible for parameters passed in and out of the module. Private and secret keys as well as seed and entropy are destroyed when the API function calls return.

## 2.8 Electromagnetic Interference / Electromagnetic Compatibility

This section is not applicable.

## 2.9 Self Tests

### 2.9.1 Power Up Self Tests

The module performs the following tests upon power up (via *FIPS_mode_set* function) or on demand (via FIPS_selftest function):

| Algorithm | Type | Description |
|---|---|---|
| AES | KAT[3] | Encryption and decryption are tested separately. ECB mode, 128 bit length |
| AES CCM | KAT | Encryption and decryption are tested separately, 192 key length |
| AES CMAC | KAT | Sign and verify CBC mode, 128, 192, 256 key lengths |
| AES GCM | KAT | Encryption and decryption are tested separately, 256 key length |
| XTS-AES | KAT | 128, 256 bit key sizes to support either the 256-bit key size (for XTS-AES-128) or the 512-bit key size (for XTS-AES-256) |
| DSA | PCT[4] | Sign and verify using 2048 bit key, SHA-256, PCKS#1 |
| CTR-based DRBG | KAT | AES, 256 bit with and without derivation function |
| Hash-based DRBG | KAT | SHA-256 |
| HMAC-based DRBG | KAT | HMAC-SHA-256 |
| SHS[5] | KAT | SHA-1, SHA-224, SHA-256, SHA-384, SHA-512 |
| HMAC | KAT | HMAC SHA-1, HMAC SHA-224, HMAC SHA-256, HMAC SHA-384, HMAC SHA-512 |
| DSA | PCT | Sign and verify using 2048 bit key, SHA-256, PCKS#1 |
| ECC CDH | KAT | Shared secret calculation |
| ECDSA | PCT | Keygen, sign and verify using P-224, K-233 and SHA512. |
| Module Integrity | KAT | HMAC-SHA1 |
| RSA | KAT | Signature generation and verification are tested separately using 2048 bit key, SHA-256, PCKS#1 |
| Triple-DES | KAT | Encryption and decryption are tested separately, ECB mode, 3-Key |
| Triple-DES CMAC | KAT | Encryption and decryption are tested separately, CBC mode, 3-Key |

**Table 7 – Power-On Self-Tests**

Power-on self tests return 1 if all self tests succeed, and 0 if not. If a self-test fails, the module enters an error state and all data output is inhibited. During self-tests, cryptographic functions cannot be performed until the tests are complete. If a self-test fails, subsequent invocation of any cryptographic function calls will fail.

### 2.9.2 Conditional Self Tests

The module performs the following conditional self tests:

| Algorithm | Modes and Key Sizes |
|---|---|
| DRBG | •Continuous Random Number Generation Test as required by SP800-90A<br><br>•SP 800-90B DRBG Health Tests |
| NDRNG | •Continuous Random Number Generation Test |
| DSA | Pairwise consistency test for both Sign/Verify and Encrypt/Decrypt |
| ECDSA | Pairwise consistency test for both Sign/Verify and |

[3] KAT: Known Answer Test
[4] PCT: Pairwise Consistency Test
[5] SHA KATs are tested as part of HMAC KATs

| | |
|---|---|
| | Encrypt/Decrypt |
| RSA | Pairwise consistency test for both Sign/Verify and Encrypt/Decrypt |

**Table 8 – Conditional Self-Tests**

In the event of a DRBG self-test failure the calling application must uninstantiate and re-instantiate the DRBG per SP 800-90A requirements. If a conditional self-test fails, the module enters an error state and all data output is inhibited. During a conditional self-test, cryptographic functions cannot be performed until the test is complete. If a self-test fails, subsequent invocation of any cryptographic function calls will fail.

## 2.10 Design Assurance

Configuration management for the module is provided by Perforce source code management system which uniquely identifies each configuration item and the version of each configuration item.

To uniquely identify each version of a document, the document date is updated manually in order to uniquely identify each version of a document. Documentation version control is performed via Perforce source code management system. There is a Perforce source code branch for each Centrify software release. Perforce labeling is used to distinguish one check-in from another.

## 2.11 Mitigation of Other Attacks

The module does not claim to mitigate any attacks outside the requirements of FIPS 140-2.

 

# Secure Operation

## 2.12 Configuration and Initialization

When installed, configured and initialized following these instructions the module only provides access to FIPS Approved algorithms and security functions. To initialize the module the FIPS_mode_set () function is invoked. During initialization, the Power-On Self Test described in Section 2.9 Self Tests are run. If any component of the self tests fails, subsequent invocation of any cryptographic function calls will fail. FIPS_mode_set () initializes the module (FIPS_mode flag is TRUE) only if all tests are successful.

## 2.13 Crypto Officer Guidance

The installation of the module is performed by the Crypto Officer role.

The module is installed as follows:

- Copy fipscanister.o to the target machine. In order to maintain security of the module's operation, the Crypto Officer shall verify that the module is installed and executed in a physically secure location.
- Link fipscanister.o with the application code by running the corresponding make script.
- During application execution, call FIPS_mode_set () function and verify the return code is successful.
- After the crypto module is built, it is packaged into a rpm file. This rpm file is bundled with other rpm packages into a tgz file. A MD5 signature of that tgz file is generated.
- The tgz file and the corresponding MD5 signature are distributed through the download center at www.centrify.com.

The following steps can be followed to verify the distributed tgz which contains the crypto module built from the certified fipscanister.o.

1. Get the tgz file, e.g. centrify-suite-2018-rhel5-x86_64.tgz, from the download center in www.centrify.com.
2. Get the corresponding MD5 signature from download center. Customer can also request a copy of the MD5 signature via email from support@centrify.com.
3. Generate the MD5 signature using the command md5sum, e.g. md5sum -b centrify-suite-2018-rhel5-x86_64.tgz
4. Compare the generated MD5 signature against the distributed one
5. If the two MD5 signatures match, then the distributed crypto module is verified

## 2.14 User Guidance

The module must be successfully configured and initialized to ensure proper operation. To initialize the module, invoke FIPS_mode_set() function which returns 1 for success and 0 for failure.

The calling application is responsible to interpret and handle the return code from the module.

 

In the event the module power is lost and restored, the calling application must ensure that any AES-GCM keys used for encryption and decryption are re-distributed.

The user shall not use disallowed key sizes (less than 2048 bits for RSA and DSA).

In accordance to NIST guidance, operators are responsible for insuring that a single Triple-DES key shall not be used to encrypt more than $2^{16}$ 64-bit data blocks.

Historically, for FIPS 140-2 validated software cryptographic module on a server to meet the single user requirement of Security Level 1, the server has to be configured so that only one user at a time could access the server. The application that makes calls to the modules is the single user of the modules, even when the application is serving multiple clients.

Please also note the guidance in Section 6.1 of NIST's publication
*Implementation Guidance for FIPS PUB 140-2 and the Cryographic Module Validation Program*

"Software cryptographic modules implemented in client/server architecture are intended to be used on both the client and the server. The cryptographic module will be used to provide cryptographic functions to the client and server applications. When a crypto module is implemented in a server environment, the server application is the user of the cryptographic module. The server application makes the calls to the cryptographic module. Therefore, the server application is the single user of the cryptographic module, even when the server application is serving multiple clients."

# 3 Acronyms

| Acronym | Definition |
|---|---|
| AES | Advanced Encryption Standard |
| CA | Certificate Authority |
| CBC | Cipher Block Chaining |
| CMVP | Cryptographic Module Validation Program |
| CO | Crypto Officer |
| CSE | Communications Security Establishment Canada |
| CSP | Critical Security Parameter |
| CVS | Concurrent Versions System |
| DRBG | Deterministic Random Bit Generator |
| ECC | Elliptic Curve Cryptography |
| EFP | Environmental Failure Protection |
| EMI/EMC | Electromagnetic Interference / Electromagnetic Compatibility |
| FCC | Federal Communications Commission |
| FIPS | Federal Information Processing Standards |
| HMAC | (Keyed-) Hash Message Authentication Code |
| KAS | Key Agreement Scheme |
| KAT | Known Answer Test |
| LED | Light Emitting Diode |
| NIST | National Institute of Standards and Technology |
| NRBG | Non-Deterministic Random Bit Generator |
| NVM | Non-Volatile Memory |
| PAA | Processor Algorithm Acceleration |
| ROM | Read Only Memory |
| RSA | Rivest, Shamir, and Adleman |
| SHA | Secure Hash Algorithm |
| Triple-DES | Triple Data Encryption Standard |
| USB | Universal Serial Bus |

**Table 9 – Acronym Definitions**

 