Samsung CryptoCore Module

FIPS 140-2 Non-Proprietary Security Policy

Version 1.4

Last Update: 10-28-2015

Prepared by:

atsec Information Security Corporation

9130 Jollyville Road, Suite 260

Austin, TX 78759

www.atsec.com

Table of Contents

Copyrights and Trademarks

1. Introduction

This document is the non-proprietary FIPS 140-2 Security Policy for the Samsung CryptoCore Module. The version number of this module is 0.2.9. 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.

1.1. Purpose of the Security Policy

There are three major reasons that a security policy is needed:

- It is required for FIPS 140-2 validation
- To provide a specification of the cryptographic security that will allow individuals and organizations to determine whether a cryptographic module, as implemented, satisfies a stated security policy.
- To describe to individuals and organizations the capabilities, protection, and access rights provided by the cryptographic module, thereby allowing an assessment of whether the module will adequately serve the individual or organizational security requirements.

1.2. Target Audience

This document is part of the package of documents that are submitted for FIPS 140-2 conformance validation of the module. It is intended for the following people:

- Developers
- FIPS 140-2 testing lab
- The Cryptographic Module Validation Program (CMVP)
- Administrators of the cryptographic module
- Users of the cryptographic module

2. Cryptographic Module Specification

The following section describes the cryptographic module and how it conforms to the FIPS 140-2 specification in each of the required areas.

2.1. Module Overview

The CryptoCore 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 also interacts with the operating system (e.g. for network I/O) via system calls. The module is implemented as a shared library libCryptoCore.so. This shared library file defines the module's logical cryptographic boundary.

As shown in the diagram below, the crypto module box defines the logical cryptographic boundary (*as indicated by the dotted red box*):

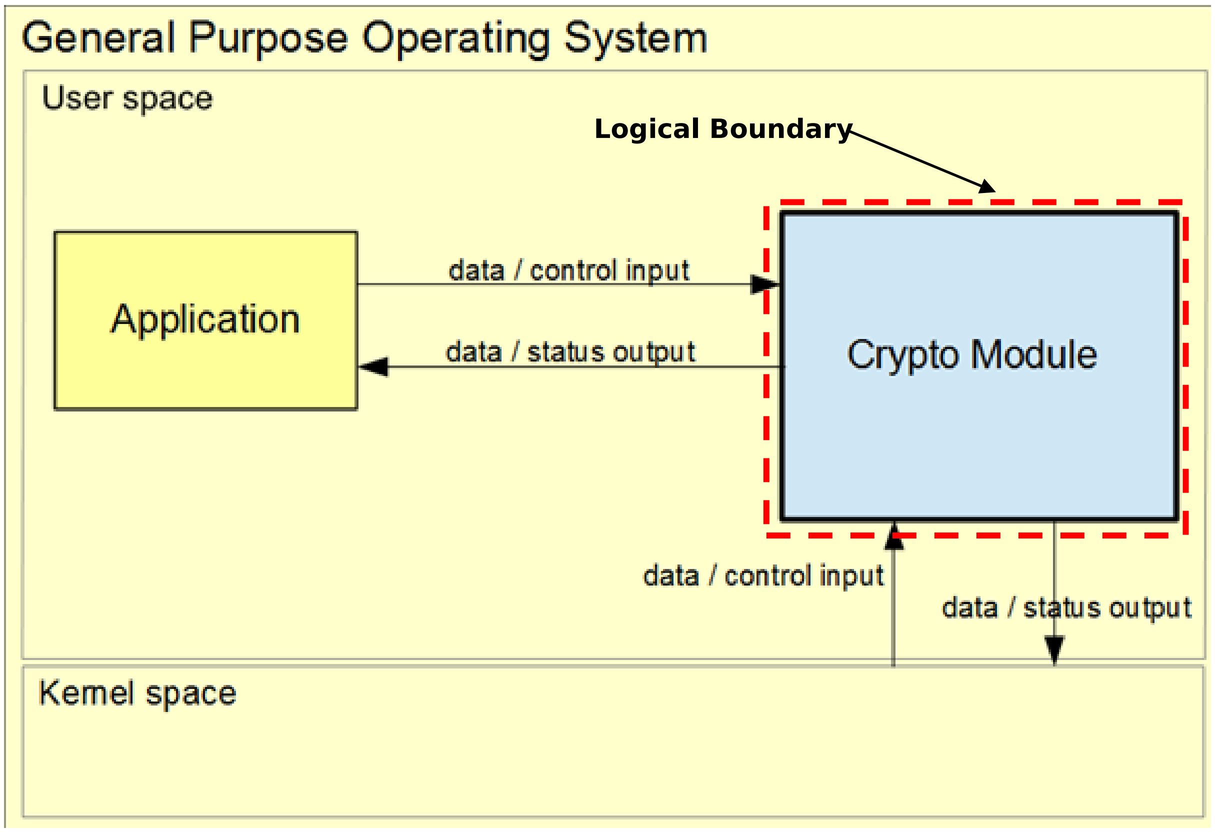

Figure 1 - Cryptographic Module Logical Boundary

For the module, the physical boundary is the surface of the case of the target platform, as shown below (*indicated by the red box*):

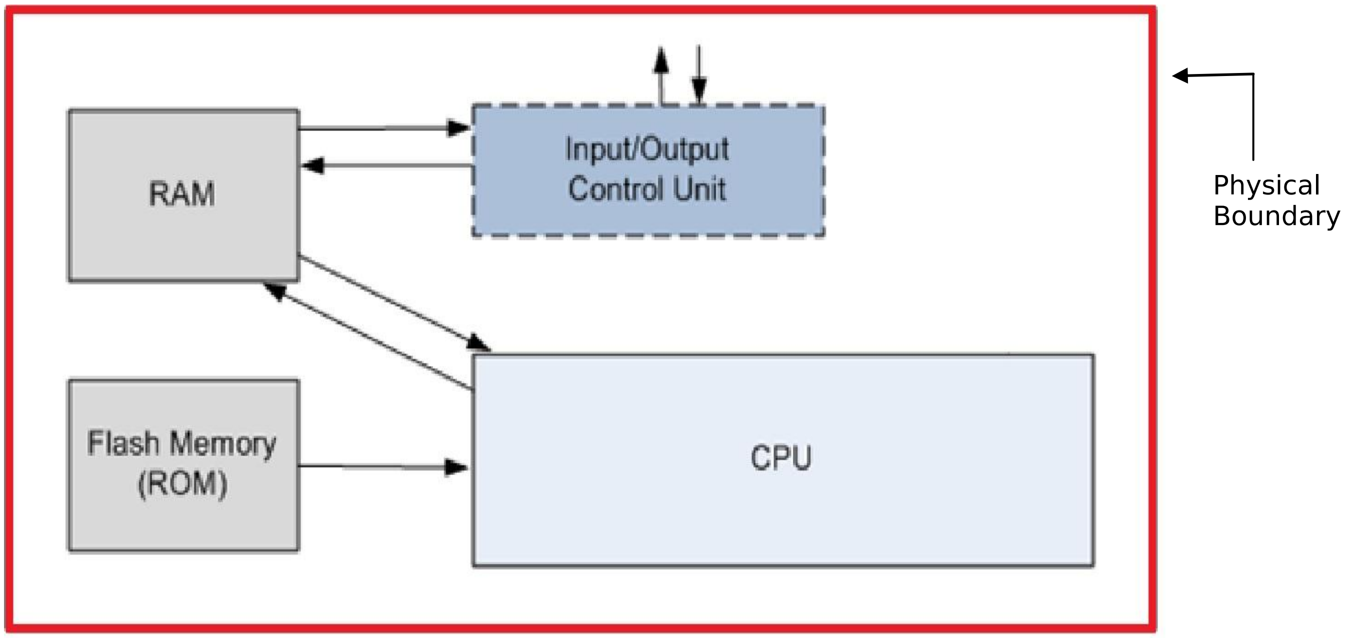

Figure 2 – Hardware Block Diagram

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 overall 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 Sections		Security Level
1	Cryptographic Module Specification	1
2	Cryptographic Module Ports and Interfaces	1
3	Roles, Services and Authentication	1
4	Finite State Model	1
5	Physical Security	N/A
6	Operational Environment	1
7	Cryptographic Key Management	1
8	EMI/EMC	1
9	Self-Tests	1
10	Design Assurance	1
11	Mitigation of Other Attacks	N/A

Table 1 - Security Levels

The module has been tested on the following platforms:

Hardware	Processor	Operating System
Samsung UN55JU6700	Samsung Hawk-MU	Tizen 2.3
Lenovo T540p	Intel i7	Ubuntu 14.04

Table 2 - Tested Platforms

In addition the vendor-affirmed testing was performed on following platforms:
-Samsung Galaxy S5 Mini with Samsung Exynos 3 Quad 3470 and Android 4.4.2
-Samsung Gear S with Snapdragon 400 Tizen 2.3

Note: the CMVP makes no statement as to the correct operation of the module or the security strengths of the generated keys when so ported if the specific operational environment is not listed on the validation certificate.

2.3. Modes of operation

The module supports both FIPS Approved and non-Approved modes of operations. After successful completion of Power-On self test the module enters FIPS mode. Invocation of any non-Approved algorithm will cause the module to operate in the non-Approved mode implicitly.

The following table shows the Approved and non-approved but allowed algorithms that can be used in FIPS-mode of operation:

Algorithm	Modes/Standard	Key Lengths	API Function	CAVP Cert #
AES	ECB, CBC, CFB128, OFB, CTR (with external IV) encryption and decryption [FIPS197] [SP800-38A]	128, 192, 256 bits	SE_init SE_process SE_final SE_EncryptOneBlock SE_DecryptOneBlock	3459 3460
CMAC AES	Message Integrity Generation [SP800-38B]	128, 192, 256 bits		
Triple-DES	ECB, CBC, CFB64, OFB, CTR (with external IV) encryption & decryption [SP800 67] [SP800 38A]	192 bits (168 without parity)	SE_init SE_process SE_final SE_EncryptOneBlock SE_DecryptOneBlock	1950 1951
SHA-1, SHA-224, SHA-256, SHA-384 SHA-512	Hashing [FIPS180-4]	N/A	MD_init MD_update MD_final MD_getHASH	2855 2856

HMAC-SHA-1 HMAC-SHA-224 HMAC-SHA-256 HMAC-SHA-384 HMAC-SHA-512	Message Integrity [FIPS198-1]	at least 112 bits	MAC_init MAC_update MAC_final MAC_getMAC	2205 2206
RSA Key Generation	X9.31 [FIPS186-4]	2048 and 3072 bits	RSA_genKeypair	1774 1775
RSA Signature Generation	RSA PKCS#1 v1.5 and PSS with SHA-224, SHA-256, SHA-384, SHA-512 [FIPS186-4]	2048 and 3072 bits	DS_sign	
RSA Signature Verification	RSA PKCS#1 v1.5, and PSS with SHA-1, SHA-224, SHA-256, SHA-384, SHA-512 [FIPS186-2] [FIPS186-4]	1024, 2048 and 3072 bits	DS_verify	

DSA
Key Generation
	[FIPS186-4]	L=2048, N=224
		L=2048, N=256
		L=3072, N=256
			DSA_genKeypair	976
				977

DSA
Domain Parameter Generation, Signature Generation
	SHA-224, SHA-256, SHA-384, SHA-512
	[FIPS186-4]
		L=2048, N=224
		L=2048, N=256
		L=3072, N=256
			DSA_genParam
			DS_sign	

DSA
Domain parameter Verification, Signature Verification
	SHA-1, SHA-224, SHA-256, SHA-384, SHA-512
	[FIPS186-4]
		L=1024, N=160
		L=2048, N=224
		L=2048, N=256
		L=3072, N=256
			DS_verify	

ECDSA Key generation	[FIPS186-4]	P-224 to P-521	EC_genKeypair	700 701
ECDSA Signature Generation	SHA-224, SHA-256, SHA-384, SHA-512 [FIPS186-4]	P-224 to P-521	DS_sign	
ECDSA Signature Verification	SHA-1, SHA-224, SHA-256, SHA-384, SHA-512 [FIPS186-4]	P-192 to P-521	DS_verify	
HMAC-based DRBG	(HMAC-SHA-1, HMAC-SHA-224, HMAC-SHA-256, HMAC-SHA-512) no PR [SP800-90A]		DRBG_HMAC_generate DRBG_HMAC_getRandom	847 848
RNG	ANSI X9.31	128, 192, 256 bits	PRNG_get	1381, 1382
EC Diffie Hellman Primitives	Key Agreement [SP800-56A]	P-224 to P-521	ECDH_GenAuthKey	530, 537
RSA (encrypt,	Key Wrapping	2048 and 3072	AE_encrypt	Non-approve

decrypt)	[FIPS186-4]	bits	AE_decrypt	d but allowed
Diffie Hellman Primitives	Key Agreement [SP800-56A]	2048 to 4096 bits	DH_GenerateParam	Non-approve d but allowed

Table 3 - FIPS 140-2 Approved and allowed Algorithms

In addition, the module supports the following Non-Approved algorithms that cannot be used in FIPS-mode with the exception of NDRNG:

Algorithm	Notes
RSA	Key Generation and Signature with key size smaller than 2048 bits or greater than 3072 bits and Signature Verification with key size smaller than 1024 bits and greater than 3072 bits
RSA (encrypt, decrypt)	Key wrapping for keys smaller than 2048 bits or greater than 3072 bits
DSA	Domain Parameters Generation, Key Generation and Signature Generation with keys smaller than 2048 bits or greater than 3072 bits
ECDSA	Key generation and signature generation with curve P-192
SHA-1	Used for digital signature generation.
Diffie Hellman	Key agreement with keys smaller than 2048 bits.
EC Diffie Hellman	Key agreement with curve P-192
RC4	non-Approved
SNOW2	non-Approved
DES	non-Approved
MD5	non-Approved
HMAC-MD5	non-Approved
IBS	non-Approved
NDRNG	non-Approved Note: The module uses NDRNG output for seeding the DRBG in FIPS-mode. The NDRNG is implemented by the underlying operating system (and not by the module) which is outside its logical boundary.

Table 4 - non-Approved Algorithms

Caveat:

1. RSA key wrapping; key establishment methodology provides 112 or 128 bits of encryption strength ; non-compliant less than 112 bits of encryption strength

2. Diffie Hellman key agreement; key establishment methodology provides between 112 and 150 bits of encryption strength; non-compliant less than 112 bits of encryption strength

3. EC Diffie Hellman key agreement; key establishment methodology provides between 112 and 256 bits of encryption strength; non-compliant less than 112 bits of encryption strength.

3. 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 that are used or processed by the module
Data Output	API output parameters and return values
Control Input	API input parameters that are used to initiate the module and the API functions or control the mode and behavior of the module
Status Output	API return codes, API output parameters for status

Table 5 - Ports and Interfaces

The Data Input interface consists of the input parameters of the API functions, and data received through the I/O system calls. The Data Output interface consists of the output parameters of the API functions and the data sent through the I/O system calls. 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.

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

4.2. Services

The module provides services to users that assume one of the available roles. All services are described in detail in the user documentation.

The following table shows the available services, the roles that can request the service, the Critical Security Parameters involved and how they are accessed:

Service	Roles		CSP	Access
	User	CO		
Symmetric encryption/decryption	✓		Symmetric key (AES, Triple-DES)	read/write/execute
Asymmetric key generation	✓		Private key (RSA, DSA, ECDSA)	read/write/execute
Digital signature generation	✓		Private key (RSA, DSA, ECDSA)	read/write/execute
Digital signature verification	✓		None	read/write/execute
Message digest generation	✓		None	read/write/execute
MAC generation	✓		HMAC key, AES key	read/write/execute
MAC verification	✓		HMAC key, AES key	read/write/execute
Random Number Generation	✓		seed and seed key for ANSI X9.31 RNG V, seed, key and entropy input for SP 800-90A DRBG	read/write/execute
Key Agreement	✓		Diffie-Hellman or EC Diffie-Hellman primitives	read/write/execute

Service	Roles		CSP	Access
	User	CO		
RSA Key wrapping	✓		RSA private key	read/write/execute
Show status	✓		None	execute
Self-Tests	✓		HMAC-SHA-512 key for integrity test	read/execute
Zeroization	✓		All CSPs	read/write/execute
Module Installation and Configuration		✓	None	execute

Table 6 – Approved Services

The following table lists the non-Approved services available in non-Approved mode. Please refer to Table 4 for the non-Approved key size or algorithm

Service	Roles		Access
	User	CO	
Symmetric encryption / decryption (DES,RC4,SNOW2,IBS)	✓		read/write/execute
Asymmetric key generation using non-Approved RSA, DSA or ECDSA key size	✓		read/write/execute
Digital signature generation/ verification using non-Approved RSA, DSA or ECDSA key size	✓		read/write/execute
Digital signature generation for RSA, DSA or ECDSA using SHA1	✓		read/write/execute
Message digest (MD5,HMAC-MD5)	✓		read/write/execute
key agreement using non-Approved key size for Diffie-Hellman or EC Diffie-Hellman	✓		read/write/execute
RSA Key wrapping using non-Approved RSA key size	✓		read/write/execute

Table 7 – Non Approved Services

4.3. Operator Authentication

The module 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 Table 2.

6.2. Policy

The operating system is restricted to a single operator (concurrent operators are explicitly excluded). The application that requests cryptographic services is the single user of the module, even when the application is serving multiple clients.

7. Cryptographic Key Management

All keys used by the Module is summarized in the table below.

Name	Auth Role	Generation	Entry/ Output	Storage	Zeroization
AES Keys	User	SP 800-90A DRBG	Entry/Output: API parameter	RAM for the lifetime of API call	destroy_CryptoCore Container ()
Triple-DES Key	User	SP 800-90A DRBG	Entry/Output: API parameter	RAM for the lifetime of API call	destroy_CryptoCore Container ()
ANSI X9.31 RNG seed and seed key	User	NDRNG from the OE	Entry: NDRNG from the OE Output: N/a	RAM for the lifetime of RNG instance	destroy_CryptoCore Container ()
CMAC keys	User	SP 800-90A DRBG	Entry/Output: API function	RAM for the lifetime of API call	destroy_CryptoCore Container ()
HMAC keys	User	SP 800-90A DRBG	Entry/Output: API parameter	RAM for the lifetime of API call	destroy_CryptoCore Container ()
RSA key pairs	User	SP 800-90A DRBG	Entry/Output: API parameter	RAM for the lifetime of API call	destroy_CryptoCore Container ()
ECDSA key pairs	User	SP 800-90A DRBG	Entry/Output: API parameter	RAM for the lifetime of API call	destroy_CryptoCore Container ()
DSA key pairs	User	SP 800-90A DRBG	Entry/Output: API parameter	RAM for the lifetime of API call	destroy_CryptoCore Container ()
Diffie-Hellman, EC Diffie-Hellman primitives	User	SP 800-90A DRBG	Entry/Output: API parameter	RAM for the lifetime of API call	destroy_CryptoCore Container ()
SP 800-90A DRBG V, seed, key and entropy input	User	NDRNG from the OE	Entry: NDRNG from the OE Output: N/A	RAM for the lifetime of DRBG instance	destroy_CryptoCore Container ()

Table 8: Key Life Cycle

7.1. Random Number Generation

The Module employs a SP 800-90A DRBG as random number generator for keys and primitives generation. The ANSI X9.31 RNG is present as a legacy use. The module uses NDRNG from the operational environment as the source of random numbers for DRBG seeds. The NDRNG produces the random numbers from an entropy pool maintained by the underlying Operating System. By default, the module gets the entropy input via the /dev/random interface of NDRNG. When /dev/random is not available, the module gets the entropy input via the /dev/urandom interface of NDRNG.

The minimum strength of the DRBG seeds is at least 112 bits. The following caveat is applicable when /dev/urandom interface is used:

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

The Module performs Continuous Random Number Generation Test (CRNGT) on the output of the SP 800-90A DRBG to ensure that consecutive random numbers do not repeat. The CRNGT on the random numbers for seeding the DRBG is performed by the kernel of the Operating System in OE.

7.2. Key/CSP Generation

The Module implements SP 800-90A compliant DRBG service for creation of symmetric and asymmetric keys.

The calling application is responsible for storage of generated keys returned by the Module. It is not possible for the module to output information during the key generating process.

7.3. Key/CSP Establishment

The module provides RSA key wrapping, Diffie-Hellman and EC Diffie-Hellman-based key establishment services.

7.4. Key/CSP Entry and Output

All keys enter the Module's logical boundary as cryptographic algorithm API parameters in plaintext. They are associated with memory locations and do not persist across power cycles. The Module does not output intermediate key generation values or other CSPs. The Module provides the resulting keys as output parameters of key generation service API to the calling application, but they do not cross the physical boundary.

7.5. Key/CSP Storage

The Module does not provide persistent key storage for keys or CSPs. The Module stores ANSI X9.31 RNG and SP 800-90A DRBG state values for the lifetime of their instance in RAM. The Module uses pointers to plaintext keys/CSPs that are passed in by the calling application. The Module does not store any CSP beyond the lifetime of an API call.

7.6. Key/CSP Zeroization

All keys and CSPs are ephemeral and are destroyed when released by the appropriate API function calls. The application is responsible for calling the destruction function listed in Table 8. This function overwrites the memory occupied by keys with “zeros” and deallocates the memory with the free() call. In case of abnormal termination, or swap in/out of a physical memory page of a process, the keys in physical memory are overwritten by the Linux kernel before the physical memory is allocated to another process.

8. Self-Tests

8.1. Power-Up Tests

The module performs power-up tests at module initialization which consists of Integrity test to ensure that the module is not corrupted and Cryptographic algorithm tests to ensure that the algorithms work as expected.

All the power-up tests are performed automatically without requiring any operator intervention. While the module is performing the power-up tests no other functions are available and all output is inhibited. Once the self-tests are completed successfully, the module enters operational mode and cryptographic services are available. If any of the power-up tests fails, the module enters ERROR State. In ERROR state, all output is inhibited and no cryptographic operations are allowed. The module needs to be reloaded in order to recover from the ERROR state. The result of the power-up tests can be obtained by calling the function GetErrorState(), which provides the status indicator. The function returns ‘0’ on successful completion of all the power-up tests and ‘1’ otherwise.

8.1.1. Integrity Tests

The integrity of the module is verified by comparing a HMAC-SHA-512 value calculated at run time with the value stored in the module that was computed at build time.

8.1.2. Cryptographic algorithm tests

The module performs self-tests on all FIPS-Approved cryptographic algorithms supported in the approved mode of operation, using the known answer tests (KAT) or a Pair wise consistency test (PCT) as shown in the following table:

Algorithm	Test
AES	• KAT AES ECB, encryption/decryption tested separately • KAT AES-CMAC, MAC generation and verification tested separately
Triple-DES	• KAT Triple-DES CBC, encryption/decryption tested separately
SHS	• KAT SHA-256
HMAC	• KAT HMAC-SHA-1 • KAT HMAC-SHA-224 • KAT HMAC-SHA-256 • KAT HMAC-SHA-384 • KAT HMAC-SHA-512
DSA	• PCT DSA (L=3072) signature generation and verification
ECDSA	• PCT ECDSA (P-224) signature generation and verification
RSA	• KAT RSA (PKCS) 2048, signature generation and verification • PCT RSA (PSS) 2048, signature generation and verification • KAT RSA 2048 encryption/decryption tested separately
DRBG	• KAT HMAC DRBG HMAC-SHA-1 • KAT HMAC DRBG HMAC-SHA-224 • KAT HMAC DRBG HMAC-SHA-256

Algorithm	Test
	• KAT HMAC DRBG HMAC-SHA-384 • KAT HMAC DRBG HMAC-SHA-512
ANSI X9.31 RNG	• KAT 128-, 192-, 256-bit AES keys
KAS ECC	• Primitive “Z” Computation KAT

Table 9- Self-Tests

8.2. On-Demand self-tests

The module provides the Self-Test service to perform self-tests on demand by calling fips_selftest_check() function. This service performs all the cryptographic algorithm tests listed in section 8.1.2. The Integrity test will be executed only during power up self-tests. During the execution of the on-demand self-tests, no other functions are available and all output is inhibited. If any of the tests fails, the module will enter ERROR state.

8.3. Conditional Tests

The module performs conditional tests on the cryptographic algorithms shown in the following table. If any of the conditional tests fail, the module enters ERROR State. The status indicator GetErrorState() function returns ‘2’ on continuous test failure and ‘5’ on any of the PCT failure.

Algorithm	Test
DSA key generation	• Pair-wise consistency test
ECDSA key generation	• Pair-wise consistency test
RSA key generation	• Pair-wise consistency test
DRBG	• Continuous test
ANSI X9.31 RNG	• Continuous test

Table 10 - Conditional Tests

In addition the module also performs DRBG Health check test as defined in section 11.3 of SP 800-90A DRBG. This test is performed as a part of power-on self test as well as during the normal operation at regular intervals. If DRBG Health check test fails due to a fatal error then the module enters ERROR State and the status indicator GetErrorState() function returns ‘-2’.

9. Installation and Usage Guidance

The module libCryptoCore.so is a shared library module. It is designed to work with the calling application which makes use of the service offered by the module. The applications can be dynamically linked to the module library with '-ldl' flag. The application should be compiled using the platform specific compiler i.e. 'gcc' for Linux platform and 'arm-linux-gnueabi-gcc' for ARM platforms.

During the module build process, the HMAC-SHA-512 digest of the entire module is calculated and appended to the module binary.

At runtime, when the application tries to utilize the module services for the first time the module is initialized automatically and power-up self-tests (POST) are executed by the module when the shared library is loaded in the calling application process space. The POST consists of integrity verification of the module where the HMAC-SHA-512 of the module binary is computed again and compared to the pre-calculated value stored within the module. If the comparison passes, this ensures that the binary is uncorrupted and the module is securely installed.

10. Mitigation of Other Attacks

The module does not implement security mechanisms to mitigate other attacks.

11. Glossary and Abbreviations

AES	Advanced Encryption Specification
CAVP	Cryptographic Algorithm Validation Program
CBC	Cipher Block Chaining
CFB	Cipher Feedback
CMAC	Cipher-based Message Authentication Code
CMVP	Cryptographic Module Validation Program
CSP	Critical Security Parameter
CVL	Component Validation List
DES	Data Encryption Standard
DFT	Derivation Function Test
DSA	Digital Signature Algorithm
ECC	Elliptic Curve Cryptography
FIPS	Federal Information Processing Standards Publication
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
PR	Prediction Resistance
PSS	Probabilistic Signature Scheme
RNG	Random Number Generator
RSA	Rivest, Shamir, Addleman
SHA	Secure Hash Algorithm
SHS	Secure Hash Standard

12. References

FIPS180-4	**Secure Hash Standard (SHS)** March 2012 http://csrc.nist.gov/publications/fips/fips180-4/fips 180-4.pdf
FIPS186-4	**Digital Signature Standard (DSS)** July 2013 http://nvlpubs.nist.gov/nistpubs/FIPS/NIST.FIPS.186-4.pdf
FIPS197	**Advanced Encryption Standard** November 2001 http://csrc.nist.gov/publications/fips/fips197/fips-197.pdf
FIPS198-1	**The Keyed Hash Message Authentication Code (HMAC)** July 2008 http://csrc.nist.gov/publications/fips/fips198 1/FIPS-198 1_final.pdf
PKCS#1	**Public Key Cryptography Standards (PKCS) #1: RSA Cryptography** Specifications Version 2.1 February 2003 http://www.ietf.org/rfc/rfc3447.txt
SP800-38A	**NIST Special Publication 800-38A - Recommendation for Block Cipher Modes of Operation Methods and Techniques** December 2001 http://csrc.nist.gov/publications/nistpubs/800-38a/sp800-38a.pdf
SP800-38B	**NIST Special Publication 800-38B - Recommendation for Block Cipher Modes of Operation: The CMAC Mode for Authentication** May 2005 http://csrc.nist.gov/publications/nistpubs/800-38B/SP_800-38B.pdf
SP800-38F	**NIST Special Publication 800-38F - Recommendation for Block Cipher Modes of Operation: Methods for Key Wrapping** December 2012 http://nvlpubs.nist.gov/nistpubs/SpecialPublications/NIST.SP.800-38F.pdf
SP800-56A	**NIST Special Publication 800-56A Revision 2 - Recommendation for Pair Wise Key Establishment Schemes Using Discrete Logarithm Cryptography** May 2013 http://nvlpubs.nist.gov/nistpubs/SpecialPublications/NIST.SP.800 56Ar2.pdf
SP800-67	**NIST Special Publication 800-67 Revision 1 - Recommendation for the Triple Data Encryption Algorithm (TDEA) Block Cipher** January 2012 http://csrc.nist.gov/publications/nistpubs/800-67-Rev1/SP-800-67-Rev1.pdf
SP800-90A	**NIST Special Publication 800-90A - Recommendation for Random Number Generation Using Deterministic Random Bit Generators** January 2012 http://csrc.nist.gov/publications/nistpubs/800-90A/SP800-90A.pdf