whiteCryption Secure Key Box 4.6.0 Crypto Module

FIPS 140-2 Level 1 Security Policy

Document version: 1.2

Last updated: 2014.11.06

Copyright Notice

Revision History

Authors	***Date***	***Version***	***Comment***
Kristaps Straupe	September 4, 2012	0.5	Initial draft
Kristaps Straupe	January 16, 2013	0.6	Updates according to design
Juris Olekss	January 23, 2013	0.7	Formatting and spell-checking
Kristaps Straupe	February 11, 2013	0.8	Various updates and clarification
Juris Olekss	February 15, 2013	0.9	Formatting and spell-checking. Added page numbers. SKB version changed to 4.6.0.
Kristaps Straupe	July 26, 2013	1.0	Final updates
Kristaps Straupe	August 12, 2014	1.1	Updates per CMVP comments
Kristaps Straupe	November 6, 2014	1.2	Updates per CMVP comments

Table of Contents

1 Introduction

This document comprises the non-proprietary FIPS 140-2 Security Policy for the whiteCryption Secure Key Box Crypto Module version 4.6.0.

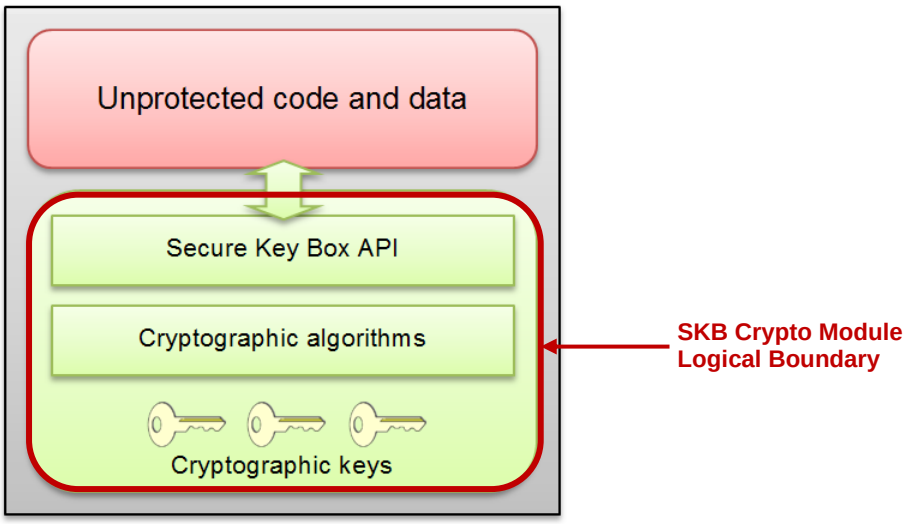
1.1 About FIPS 140-2

FIPS 140-2 (Federal Information Processing Standards Publication 140-2 – *Security Requirements for Cryptographic Modules*) details the U.S. and Canadian Government requirements for cryptographic modules. More information about the FIPS 140-2 standard and validation program is available on the Cryptographic Module Validation Program (CMVP) website, which is maintained by National Institute of Standards and Technology (NIST) and Communication Security Establishment Canada (CSEC): http://csrc.nist.gov/groups/STM/cmvp/index.html.

1.2 About Secure Key Box

Secure Key Box (SKB) is a C/C++ library that provides an extensive set of high-level classes and methods for working with the most popular cryptographic algorithms.

SKB exposes a comprehensive interface that provides access to a set of cryptographic algorithms. An application that is integrated with SKB executes the cryptographic algorithms via the SKB API.



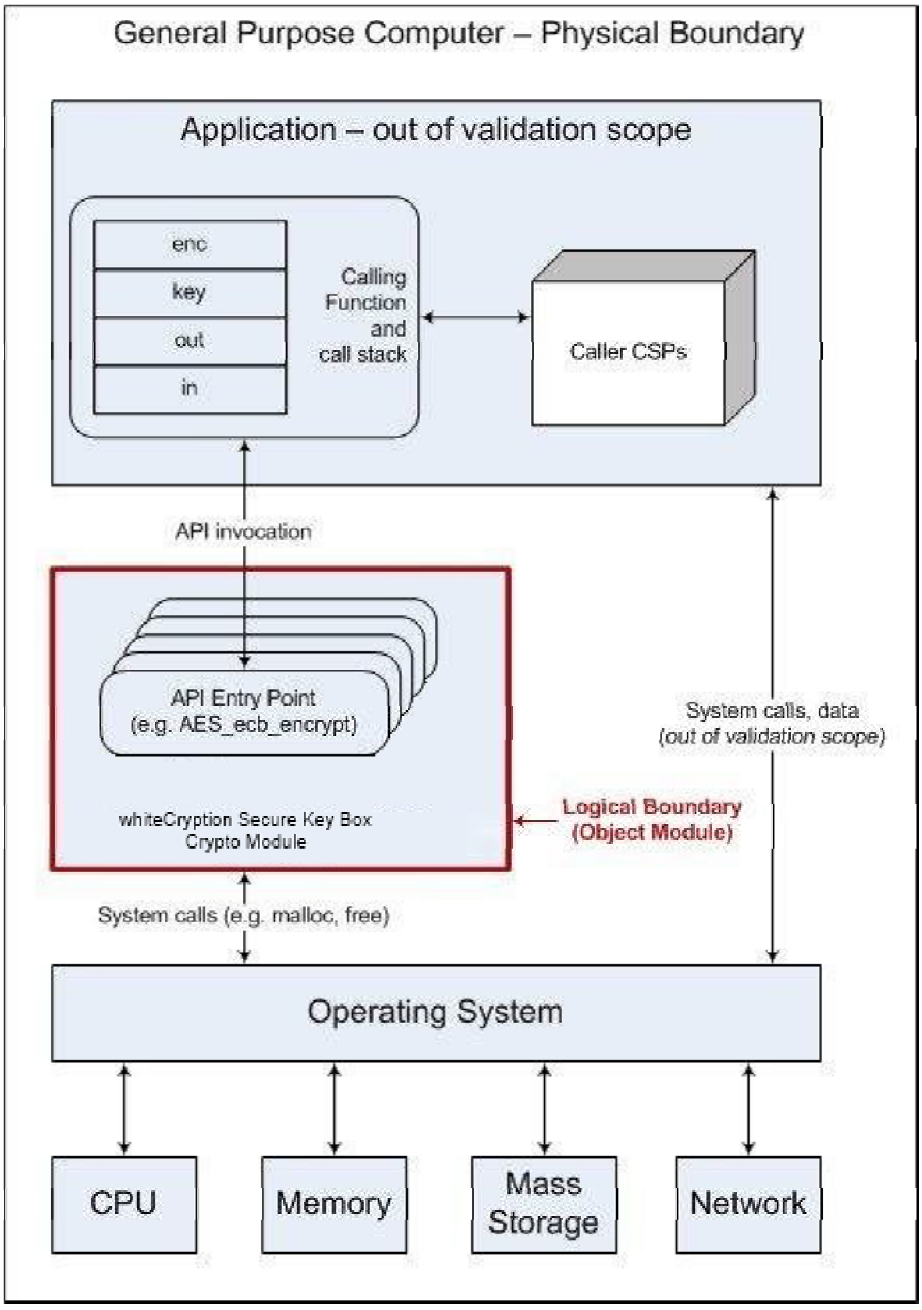
SKB overview

Because of the library's robust design, systems integrated with SKB can be safely deployed in insecure environments, such as mobile devices, tablets, and desktop computers, where anyone could have access to the program code and device memory.

In this document, whiteCryption Secure Key Box is referred to as SKB, the library, or the Module.

2 Module Specification

The Module is a software library providing a C-language application program interface (API) for use by other processes that require cryptographic functionality. The Module is classified by FIPS 140-2 as a software module, multi-chip standalone module embodiment. The physical cryptographic boundary is the general purpose computer on which the Module is installed. The logical boundary of the cryptographic Module is the single shared object (SO). The Module performs no communications other than with the calling application (the process that invokes the Module services).



System diagram

The FIPS 140-2 security levels for the Module are as follows:

Security Requirement	***Security Level***
Cryptographic Module Specification	1
Cryptographic Module Ports and Interfaces	1
Roles, Services, and Authentication	1
Finite State Model	1
Physical Security	N/A
Operational Environment	1
Cryptographic Key Management	1
EMI/EMC	1
Self-Tests	1
Design Assurance	3
Mitigation of Other Attacks	N/A

Table 1 – Security level of security requirements

The Module's software version for this validation is 4.6.0.

2.1 Tested Configurations

Operational Environment	***Processor***	***Optimizations (Target)***
Android 4.2.2	Qualcomm Snapdragon S4 (ARMv7)	None

Table 2 – Tested platforms

2.2 Cryptographic Functionality and Approved Mode of Operation

The Module supports FIPS 140-2 approved mode. Tables 3a and 3b list the approved and non-approved but allowed algorithms contained within the Module.

Function	***Algorithm***	***Options***	***Cert #***
Random number generation	[SP 800-90] DRBG	Hash_DRBG using SHA-256 at highest security strength (256 bit) and prediction resistance not supported	335

Function	***Algorithm***	***Options***	***Cert #***
Symmetric cipher	[FIPS 197] AES	128/192/256 ECB, CBC, CTR	2451, 2452, 2453, 2454, 2455, 2456, 2457, 2458, 2459, 2460, 2461, 2462, 2463, 2464, 2465, 2466, 2467
Message authentication code	[FIPS 198] HMAC	HMAC-SHA-1, HMAC-SHA-256	1516, 1517
	[SP 800-38B] CMAC	CMAC 128	AES 2470, 2471
Hashing	[FIPS 180-3] SHA	SHA-1 SHA-2: SHA-256, SHA-384	2084, 2085, 2086, 2087, 2088, 2089, 2090
Digital signature and asymmetric key generation	[FIPS 186-3] ECDSA	Key pair: P-224, P-256 SigGen: P-224 (SHA-256), P-256 (SHA-256	403
		Key pair: P-384, P-521 SigGen: P-384 (SHA-256), P-521 (SHA-256)	404
		ECDSA SigGen component: P-224 (SHA-256, SHA-384), P-256 (SHA-256, SHA-384)	CVL 83
		ECDSA SigGen component: P-384 (SHA-256, SHA-384), P-521 (SHA-256, SHA-384)	CVL 84
	[FIPS 186-3] RSA	SigGenPKCS #1 v1.5 and SigGenPSS 2048 with SHA-256 RSASP1 PKCS #1 v1.5 SigGenComponent 2048 with SHA-256	1263 CVL 94
Key derivation	[SP 800-108] KDF	KDF in Counter Mode using CMAC AES-128	KBKDF 11 and AES 2471
EC Diffie-Hellman key agreement primitive	[SP 800-56A] (§5.7.1.2)	Ephemeral unified ECC CDH primitive on all NIST defined P curves; P-224, P-256	CVL 79
		Ephemeral unified ECC CDH primitive; P-384, P-521	CVL 80

Table 3a – FIPS approved cryptographic functions

Function	Algorithm	Description
Key unwrap	AES	Key unwrap using AES 128, 192 or 256 (Certs. #2464 and #2465). No CSPs are established into or exported out of the Module. This is only used for system level key establishment, which is beyond the scope of this module. When used for system level key establishment, this service provides between 128 and 256 bits of security.
RSA primitives and operations	RSA	The RSA (OAEP) 2048 decrypt operation and RSADP 2048 primitive. No CSPs are established into or exported out of the Module. This is only used for system level key establishment, which is beyond the scope of this module. When used for system level key establishment, this service provides 112 bits of security.

Table 3b – Non-FIPS approved but allowed cryptographic functions

Table 3c lists the algorithms/options that are Disallowed as of January 1, 2014 per the NIST SP 800-131A algorithm transitions. Algorithms providing less than 112 bits of security strength are not allowed in the FIPS Approved mode of operation for use by Federal agencies.

Function	Algorithm	Options	Cert #
Digital signature and asymmetric key generation	[FIPS 186-3] ECDSA	Key pair: P-192 SigGen: P-192 (SHA-1, 256), P-224 (SHA-1), P-256 (SHA-1)	403
		SigGen: P-384 (SHA-1), P-521 (SHA-1)	404
		ECDSA SigGen component: P-192, P-224 (SHA-1), P-256 (SHA-1)	CVL 83
		ECDSA SigGen component: P-384 (SHA-1), P-521 (SHA-1)	CVL 84
	[FIPS 186-3] RSA	SigGenPKCS #1 v1.5 and SigGenPSS 1024 (SHA-1, SHA-256)	1262
		SigGenPKCS #1 v1.5 and SigGenPSS 2048 (SHA-1)	1263
		RSASP1 PKCS #1 v1.5 SigGenComponent 2048 (SHA-1)	CVL 94
EC Diffie-Hellman key agreement primitive	[SP 800-56A] (§5.7.1.2)	Ephemeral unified ECC CDH primitive; P-192	CVL 79
RSA primitives and operations	RSA	The RSA (OAEP) 1024 decrypt operation and RSADP 1024 primitive. No CSPs are established into or exported out of the Module.	N/A

Table 3c – Cryptographic Functions Disallowed (as of 1/01/2014) per NIST 800-131A Transitions

The Module does not provide full EC Diffie-Hellman key agreement scheme, only the primitive.

The following non-FIPS Approved and not allowed algorithms are included in the Module:

- ECDSA Key Pair and SigGen with SHA-1 and SHA-256 using SECG, secp160r1 curve
- PKCS #1 v1.5 RSA 1024 decryption operation
- PKCS #1 v1.5 RSA 2048 decryption operation
- RSASP1 PKCS #1 v1.5 SigGenComponent non-compliant mod length 1024

These non-approved and not allowed cryptographic algorithms shall not be used while operating the Module in the FIPS approved mode.

The Module operates in FIPS 140-2 Approved mode of operation as long as only the services using approved and allowed algorithms are used.

When the Module library is loaded into memory an integrity check along with power-up self-tests is performed. If any of these fail, the Module will enter a fail state and an error code will be returned resulting in no cryptographic services available. The Module is a cryptographic engine library, which can be used only in conjunction with additional software.

Aside from the use of the NIST-defined elliptic P curves as trusted third party domain parameters, all other FIPS 186-3 and SP 800-56A assurances are outside the scope of the Module, and are the responsibility of the calling process.

The algorithm selection in Module services is achieved by passing a specific enum value in the service call. For enum values corresponding to approved and allowed algorithms please refer to Appendix D.

Rules of operation enforced by the cryptographic Module to implement the security requirements of this FIPS 140-2 Level 1 Module are as follows:

- The operating system shall be restricted to a single operator mode of operation (i.e., concurrent operators are explicitly excluded).
- The Module shall be operated in single user/operator mode. The external application that makes calls to the cryptographic Module is the single user of the cryptographic Module, even when the application is serving multiple clients (IG 6.1).
- The unauthorized reading, writing, or modification of the address space of the Module is prohibited.
- The referencing application accessing the Module runs in a separate virtual address space with a separate copy of the executable code.
- The operating system is responsible for multi-tasking operations so that other processes cannot access the address space of the process containing the Module.
- The operator (calling application) shall use appropriate entropy sources (entropy source callback in the Module) for generating the seeding material for the FIPS-approved DRBG of the Module. The entropy in the seeding material input to the Module must be at least as much as the security strength of the Modules employed DRBG (256 bits).
- Operator (calling application) shall only use algorithms, key sizes, and elliptic curves outlined in Tables 3a and 3b.
- Operator (calling application) shall use the zeroization services provided by the Module on keys or service contexts no longer needed by the operator.
- The operator (calling application) shall operate the Module only through services and API calls and parameters listed in Table 10 in Appendix D.

3 Ports and Interfaces

The Module uses the same physical ports as the computer system on which it is executing.

The logical interface is a C-language API.

Logical Interface Type	***Description***
Control input	API entry point and corresponding stack parameters
Data input	API entry point data input stack parameters
Status output	API entry point return values and status stack parameters
Data output	API entry point data output stack parameters

Table 4 – Logical interfaces

As a software module, control of the physical ports is outside of the Module scope. However, when the Module is performing self-tests, or is in an error state, all output on the logical data output interface is inhibited. The Module is single-threaded and in error scenarios returns only an error value (no data output is returned).

4 Roles and Services

The roles of the Crypto Officer or User role are assumed implicitly when installing or operating the services of the Module. The Module does not support any kind of authentication. The Module is not allowed to be used with concurrent operators. The Module does not provide identification or authentication mechanisms that would distinguish between the two supported roles.

The Crypto Officer role can also install, remove, or instantiate the Module.

All services implemented by the Module (and accessible to both roles) are listed below, along with a description of service CSP access – read (R) the CSP, or write (W) the CSP.

** *It is the responsibility of the module operator to ensure that algorithms, modes, and key sizes Disallowed per the NIST SP 800-131A algorithm transitions are not used.*

Approved Mode	***Non-Approved Mode***	***Service***	***Impact of SP 800-131A Transitions (Disallowed Options in RED)****	***Description***	***Input***	***Output***	***CSP and Access Operation***
X		Instantiate	N/A	Instantiates the Module engine	None	Engine instance, Status	Seed (W, R) DRBG state (W)

Table 5a – Crypto Officer Services and CSP access

Approved Mode	***Non-Approved Mode***	***Service***	***Impact of SP 800-131A Transitions (Disallowed Options in RED)****	***Description***	***Input***	***Output***	***CSP and Access Operation***
X		Check Integrity	N/A	Checks the integrity of the Module using the SHA-256 digest of the Module in memory against the embedded known MAC value, sets the Module error status on mismatch	None	Status	None
X		Execute Self-Test	N/A	Runs the power-up tests of the Module including KAT and sign/verify where applicable, updates the Module error status on failure	None	Status	None
X		Get Error Status	N/A	Gets the error status indicating the state of the Module – operational or in fail state	None	Status	None

Approved Mode	Non-Approved Mode	Service	Impact of SP 800-131A Transitions (Disallowed Options in RED)**	Description	Input	Output	CSP and Access Operation
X		AES Encrypt/ Decrypt	N/A	Encrypt/decrypt plaintext/ciphertext	Parameters, AES key, plaintext/ ciphertext	Status, ciphertext/ plaintext	AES Key (R)
	X	RSA-OAEP Decrypt	RSA key size: 1024-bit; 2048-bit (SHA-1)	Decrypt ciphertext	Parameters, RSA private key, ciphertext	Status, keying material	RSA private key (R)
X			RSA key size: 2048-bit (SHA-2)				
X		Secure Hashing	N/A	Hash data	Parameters, Data	Status, digest	None
	X	RSA Signature Generation	RSA key size: 1024-bit; 2048-bit (SHA-1)	Signature generation and/or Signature primitive only (steps 3 to 6 of PKCS #1 v1.5 encoding included)	Parameters, RSA private key, data	Status, signature	RSA private key (R) If generating RSA PSS signature: Seed (R)* DRBG state (R, W)
X			RSA key size: 2048-bit (SHA-2)				
	X	ECDSA Signature Generation	ECDSA key size: P-192; P-224, P-256, P-384, P-521 (SHA-1)	Signature generation and/or Signature primitive only	Parameters, ECDSA private key, data	Status, signature	ECDSA private key (R) Seed (R)* DRBG state (R, W)
X			ECDSA key size: P-224, P-256, P-384, P-521 (SHA-2)				
X		HMAC	N/A	Generate (or verify) MAC using HMAC	Parameters, HMAC key, data/signature	Status, MAC/ verification result	HMAC Key (R)
X		CMAC	N/A	Generate (or verify) MAC using CMAC	Parameters, AES key, data/signature	Status, MAC/ verification result	AES Key (R)
X		Random Byte Generation	N/A	Generate random value which can be used for system level AES/HMAC keys	Parameters	Status, AES or HMAC key	AES Key (W) or HMAC Key (W) Seed (R)* DRBG state (R, W)
	X	Compute ECDH Key Agreement Shared Secret	ECDH: P-192	Elliptic Curve Diffie-Hellman key agreement shared secret computation	Parameters, ECDH private component and public key	Status, shared secret	ECDH Private component (R) ECDH Public key (R)
X			ECDH: P-224, P-256, P-384, P-521				
	X	Generation of ECDSA key pair	ECDSA key size: P-192; P-224, P-256, P-384, P-521 (SHA-1)	Generate an ECDSA key pair	Parameters	Status, ECDSA private key and public	ECDSA private key (W) ECDSA public key (W)

Approved Mode	***Non-Approved Mode***	***Service***	***Impact of SP 800-131A Transitions (Disallowed Options in RED)*****	***Description***	***Input***	***Output***	***CSP and Access Operation***
X			ECDSA key size: P-224, P-256, P-384, P-521 (SHA-2)			key	Seed (R)* DRBG state (R, W)
X		Key Derivation	N/A	SP 800-108 key derivation function in counter mode using CMAC	Parameters, AES key (KDK), data (label, context)	Status, keying material	AES KDK (R) AES Key (W) or HMAC Key (W)
X		Key Unwrap	N/A	AES Key Wrap algorithm	Parameters, AES key (KEK), wrapped AES key	Status, AES key	AES KEK (R) AES Key (W) or HMAC Key (W)
X		Import	N/A	Converts an exported key (from a pre- formatted byte buffer) to appropriate Module key type object for use in Module services. Key is loaded from inside the physical boundary of the GPC.	Key (byte buffer)	Status, Key	AES Key (W) or HMAC Key (W) or RSA private key (W) or ECDSA private key(W)
X		Export	N/A	Converts the key from a Module key type object to a pre-formatted byte buffer (within the physical boundary of the GPC)	Parameters, key	Status, key (byte buffer)	AES Key (R) or HMAC Key (R) or RSA private key (R) or ECDSA private key (R)
X		Key/service context zeroization	N/A	Zeroizes and deallocates the memory containing a symmetric key, asymmetric private key, DRBG state	Memory address	Status	All CSPs (W)

** only if periodic DRBG re-seed happens*

Table 5b – User Services and CSP access

5 Operational Environment

The FIPS 140-2 Area 6 Operational Environment requirements are applicable because the Module operates in a modifiable operational environment.

Operational testing of the Module was performed on the following environment:

- Android 4.2.2 (single-user mode)

The Module is considered to be FIPS 140-2 compliant when running on other binary compatible operating environments/hardware provided that rules described in “Cryptographic Functionality and Approved Mode of Operation” are met.

6 Cryptographic Key Management

The Module supports the following critical security parameters in tables below. It should be noted that the Module key paths are interpreted as being the input and output via “INT” paths as defined in IG 7.7. Therefore, key establishment mechanism is not applicable for this particular Module since all input and output is occurring within the physical boundary of the GPC.

All CSPs used by the Module are described in this section. All access to these CSPs by Module services are described in Table 5.

*** It is the responsibility of the module operator to ensure that algorithms, modes, and key sizes Disallowed per the NIST SP 800-131A algorithm transitions are not used.*

Approved Mode	***Non-Approved Mode***	***Key***	***Impact of SP 800-131A Transitions (Disallowed Options in RED)****	***Description/ Purpose***	***Generation***	***Destruction***
	X	RSA keys	1024-bit; 2048-bit (SHA-1)	Used to create RSA digital signatures and to decrypt (OAEP) keys	Externally	An application program which uses the API may destroy the key. The key/context destruction service zeroizes this CSP.
X			2048-bit (SHA-2)			
	X	ECDSA private keys	P-192; P-224, P-256, P-384, P-521 (SHA-1)	Used to create ECDSA digital signatures	May be generated internally by ECDSA key pair generation service or generated externally	An application program which uses the API may destroy the key. The key destruction service zeroizes this CSP.
X			P-224, P-256, P-384, P-521 (SHA-2)			
	X	ECDH private components	P-192	Used to derive the shared secret during ECDH key agreement protocol	May be generated internally (by underlying ECDSA key pair generation service) or externally	An application program which uses the API may destroy the key. The ECDH context destruction service zeroizes this CSP.
X			P-224, P-256, P-384, P-521			
X		AES keys	N/A	Used during AES encryption, decryption, CMAC operations, or as KDK/KEK for KDF/Wrapping	May be generated internally (by random byte generation service), derived/ unwrapped or generated externally	An application program which uses the API may destroy the key. The key/context destruction service zeroizes this CSP.
X		HMAC keys	N/A	Used during HMAC SHA-1/256 operations	May be generated internally (by random byte generation service), derived/ unwrapped or generated externally	An application program which uses the API may destroy the key. The key/context destruction service zeroizes this CSP.
X		DRBG seed	N/A	Used to seed the DRBG for random value generation	Generated internally through supplied SEI source	Automatically after use

Approved Mode	Non-Approved Mode	Key	Impact of SP 800-131A Transitions (Disallowed Options in RED)**	Description/ Purpose	Generation	Destruction
X		DRBG 'V' and 'C' values	N/A	Used in HASH_DRBG state	Internally	An application program which uses the API may destroy the DRBG state. The internal DRBG state used for key generation services is zeroized upon Module engine destruction.

Table 6a – Critical Security Parameters

Module integrity data is loaded into the Module during the Module build process and otherwise cannot be accessed.

The Module does not output intermediate key generation values.

Key	Description/Purpose	Generation
DH public components	Used to derive the shared secret during ECDH key agreement protocol	Internally using ECDSA key pair generation service or externally
ECDSA public keys	Public component of ECDSA key pair	Internally using ECDSA key pair generation service

Table 6b – Public keys

For all CSPs and public keys:

- **Storage**: RAM, associated to entities by memory location. The Module does not store any CSPs persistently.
- **Generation**: The Module implements SP 800-90 compliant DRBG algorithm used by services for creation of AES/HMAC keys, and elliptic curve as shown in Tables 6a and 6b.
- **Entry and output**: The cryptographic Module itself does not support key entry or key output from its physical boundary. CSPs enter the Module's logical boundary in plaintext as API parameters, associated by memory location. The Module does not output CSPs other than as explicit results of key generation services. The calling application using the Module is responsible for ensuring that the input or output of secret and private keys is accomplished in encrypted form.
- **Destruction**: Zeroization of sensitive data is performed automatically by API function calls for temporarily stored CSPs. The calling application is responsible for keys passed in and out of the Module as well as the employed Module service contexts which might store CSPs (such as expanded keys). The Module provides services for destruction of keys and service contexts which store the CSPs. The Module offers zeroization of these CSPs through API functions *SKB_FIPS_SecureData_Release* – for key destruction*, SKB_FIPS_Cipher_Release, SKB_FIPS_Transform_Release, SKB_FIPS_KeyAgreement_Release* – for service context destruction as well as *SKB_FIPS_Engine_Release* for Module employed DRBG CSPs destruction.

7 Self-Tests

The Module performs the self-tests listed below on Module library load (in Default Entry Point) or when self-test function is manually invoked during operation.

Algorithm	***Type***	***Test Attributes***
Software integrity	KAT	HMAC-SHA-256
AES	KAT	Separate encrypt and decrypt for each ECB, CBC, CTR mode and 128, 192, 256 bit key length combination on all supported algorithm implementations
SHS	KAT	SHA-1, SHA-256, and SHA-384
HMAC	KAT	MAC and verify with SHA-1 and SHA-256
AES CMAC	KAT	Sign and verify, 128 bit key length
RSA	KAT	Signature generation PKCS #1 v1.5 1024 and 2048 bit mod length and SHA-256.
	KAT	Decrypt OAEP with 1024 and 2048 bit mod length
ECDSA	PCT	Signature generation P-256 and P-384 curve bit length with SHA-256
DRBG	KAT	HASH_DRBG using SHA-256
ECC CDH	KAT	P-256 and P-384 curve bit length primitive “Z” computation KAT (SP 800-56A §5.7.1.2)
KDF	KAT	KDF in counter mode (SP 800-108)
KW-AD(C)	KAT	Key unwrap using AES

Table 7 – Power-on self-tests (KAT = known answer test; PCT = pairwise consistency test)

When Module library is being loaded in memory by a calling application, a run-time integrity check and all power-up self-tests listed above are performed returning a result value indicating success or failure. The power-up self-tests may also be performed on-demand by calling *SKB_FIPS_ExecuteSelfTests*. The same is true for integrity checking function *SKB_FIPS_ExecuteIntegrityCheck*. Interpretation of the return value is the responsibility of the calling application. If the integrity check or any component of the self-test fails an internal flag is set to prevent subsequent invocation of any cryptographic function calls.

The Module also implements the following conditional tests:

Algorithm	***Test***
ECDSA key generation	Pairwise consistency test on each generation of a key pair (sign with private part, verify with public part)
DRBG	Health tests as required by [SP800-90] Section 11 (refer to Appendix C for more information)

Algorithm	***Test***
DRBG	DRBG FIPS 140-2 continuous test for stuck fault

Table 8 – Conditional tests

If the pairwise consistency test fails, the Module enters a fail state to prevent subsequent invocation of any cryptographic function calls.

In case a health test failure occurs in the Module engine's internal DRBG used by the key generation service and other services the Module will enter a fail state and deny any further operation.

Once the Module is in a fail state, a successful reloading of the Module's library is required to continue operation.

8 Design Assurance

Subversion (SVN), Jenkins, and whiteCryption proprietary extensive SKB test suite and release scripts are used to provide testing and configuration management for the Module and related documentation. These solutions provide access control, versioning, testing, and logging.

9 Mitigation of Other Attacks

The Module does not claim any attack mitigation beyond FIPS 140-2 Level 1 requirements.

Appendix A – Module Integrity

The Secure Key Box shared library file is protected by a HMAC-SHA-256 MAC. The MAC value is pre-calculated and embedded in the delivered Module object. This value is calculated on the code (.text) and constant/read-only data (.rdata) sections. The value is stored in a specifically marked place in the constant data section of the Module. This place is not included in the message input for the digest algorithm, thus it does not influence the digest value. At run-time, the integrity validation check is executed once the Module library is being loaded into memory. The integrity check performs HMAC-SHA-256 on .text and .rdata sections of the Module in volatile memory and compares it to the embedded value. If the integrity check fails, further operations of services in the Module are denied. This check can also be initiated at any time during the operation of the Module by calling the integrity checking API method directly. By doing so, the Operator of the Module can reassure from time to time that the Module in memory has not been altered.

To assure the authenticity of the distributed module, compute and compare SHA-256 digest of the Modules file (libSkbFips.so) against this provided value:

AD8D2A5B1A8A3E188FC5B49D541C8292EC489586B8C4D5854D25D1EFB09F6C12

Appendix B – Source Entropy Input for the SKB Module

As the Module implements various algorithms that use underlying DRBG mechanism, a SEI has to be made available for the Module operation. The Module has a callback function for SEI that needs to be initialized before operation. The callback function's signature is as follows:

```
SKB_Result ApplicationEntropySource(SKB_Byte* buffer, SKB_Size count)
```

The first parameter is a byte buffer in which to store the entropy input. The second argument is the byte count of the requested entropy input.

The return value specifies if the callback was successful and shall return *SKB_SUCCESS* in case of normal operation. If an error is encountered, for instance the entropy of 256 bits cannot be achieved, the return value shall be *SKB_FAILURE*. The Module will enter error state if the callback is unsuccessful.

The entropy source callback will be used only for instantiating and re-seeding the Module's internal SHA-256 HASH_DRBG.

The SKB API call to set the callback function is as follows:

```
SKB_FIPS_SetEntropySource(ApplicationEntropySource)
```

Appendix C – HASH_DRBG Specification

The Module implements SHA-256 hash DRBG compliant to SP800-90A publication. The DRBG supports only maximum 256 bit security strength and has no prediction resistance capability. The source of entropy input for the DRBG is completely under application control and the callback for entropy source has to be set beforehand to make the operation of DRBG possible (see Appendix B).

DRBG Health Testing

Implementation health testing is performed for the following DRBG functions:

- instantiate
- re-seed
- generate

Health testing checks if the expected error codes are returned in response to invalid parameters or context, and verifies that the DRBG KATs pass. Health testing is done before the actual function is performed. If any health check fails, the given DRBG context (on which the call to DRBG service was made) is placed in an error state and no further operations can be performed on that DRBG instance until it has been re-instantiated.

The instantiate function health test is initiated on each instantiation and performs the following checks:

- Instantiate with invalid parameters (null context, already instantiated context, context is in error state, invalid nonce/personalization string)
- Instantiate with entropy source failure (when entropy source does not return SKB_SUCCESS because of insufficient entropy)
- Instantiate KAT test variants of nonce/personalization string passed/omitted

The re-seed function health test is initiated on each re-seeding (whether explicit or within the automatic re-seed inside the generate function) and performs the following checks:

- Re-seed with invalid parameters (null context, uninstantiated context, context in error state)
- Re-seed with entropy source failure (when entropy source does not return SKB_SUCCESS because of insufficient entropy)
- Instantiate then re-seed KAT test

The generate function health test is initiated on the first run of generate for given DRBG context and on each subsequent 2^{24} generation. The check interval can be adjusted via a function call. The health checks performed in the generate function are as follows:

- Generate with invalid parameters (null context, non-instantiated context, context in error state, null buffer for output requested bytes, requested byte count too large)
- Instantiate then generate KAT test
- Instantiate then generate with triggered re-seed KAT test
- Instantiate then generate, and then again generate a stuck fault test

The uninstantiate function health test is performed whenever the other DRBG health tests execute. The uninstantiate function's health test checks include:

- Uninstantiate with invalid (null) context
- Check that the CSPs (V and C) have been zeroized after the uninstantiate call

Appendix D – Service to Module API Mapping When Performing Services in Approved Mode

The following table maps the Module services to actual Module API functions and enumeration values used to select the algorithms. When operating the Module in FIPS approved mode, the Module user shall use only the functions and enumerations found in this table.

Service	***API Function***	***API Enumeration Value***
Initialize	SKB_FIPS_GetInstance	
Check integrity	SKB_FIPS_ExecuteIntegrityCheck	
Execute self-tests	SKB_FIPS_ExecuteSelfTests	
Get error status	SKB_FIPS_GetErrorStatus	
AES encrypt/ decrypt	SKB_FIPS_Engine_CreateCipher SKB_FIPS_Cipher_ProcessBuffer	SKB_CIPHER_ALGORITHM_AES_128_ECB SKB_CIPHER_ALGORITHM_AES_192_ECB SKB_CIPHER_ALGORITHM_AES_256_ECB SKB_CIPHER_ALGORITHM_AES_128_CBC SKB_CIPHER_ALGORITHM_AES_192_CBC SKB_CIPHER_ALGORITHM_AES_256_CBC SKB_CIPHER_ALGORITHM_AES_128_CTR SKB_CIPHER_ALGORITHM_AES_192_CTR SKB_CIPHER_ALGORITHM_AES_256_CTR SKB_CIPHER_DIRECTION_ENCRYPT SKB_CIPHER_DIRECTION_DECRYPT SKB_DERIVATION_ALGORITHM_CIPHER SKB_DATA_FORMAT_RAW SKB_DATA_TYPE_BYTES
	SKB_FIPS_Engine_CreateDataFromWrapped SKB_FIPS_SecureData_Wrap SKB_FIPS_Engine_WrapDataFromPlain	
	SKB_FIPS_SecureData_Derive	
RSA-OAEP Decrypt**	SKB_FIPS_Engine_CreateCipher SKB_FIPS_Cipher_ProcessBuffer	SKB_CIPHER_ALGORITHM_RSA_OAEP SKB_CIPHER_ALGORITHM_RSA SKB_CIPHER_DIRECTION_DECRYPT SKB_DATA_FORMAT_RAW SKB_DATA_TYPE_BYTES
	SKB_Engine_CreateDataFromWrapped	
Secure hashing	SKB_FIPS_Engine_CreateTransform SKB_FIPS_Transform_AddSecureData SKB_FIPS_Transform_AddBytes SKB_FIPS_Transform_GetOutput	SKB_DIGEST_ALGORITHM_SHA1 SKB_DIGEST_ALGORITHM_SHA256 SKB_TRANSFORM_TYPE_DIGEST
	SKB_FIPS_SecureData_Derive	SKB_DERIVATION_ALGORITHM_SHA_384

Service	***API Function***	***API Enumeration Value***
RSA signature generation**	SKB_FIPS_Engine_CreateTransform SKB_FIPS_Transform_AddSecureData SKB_FIPS_Transform_AddBytes SKB_FIPS_Transform_GetOutput	SKB_SIGNATURE_ALGORITHM_RSA (only on modulus size 2048) SKB_SIGNATURE_ALGORITHM_RSA_SHA1 SKB_SIGNATURE_ALGORITHM_RSA_SHA256 SKB_SIGNATURE_ALGORITHM_RSA_PSS_SHA1 SKB_SIGNATURE_ALGORITHM_RSA_PSS_SHA1_EX SKB_SIGNATURE_ALGORITHM_RSA_PSS_SHA256 SKB_SIGNATURE_ALGORITHM_RSA_PSS_SHA256_EX SKB_TRANSFORM_TYPE_SIGN
ECDSA signature generation**	SKB_FIPS_Engine_CreateTransform SKB_FIPS_Transform_AddSecureData SKB_FIPS_Transform_AddBytes SKB_FIPS_Transform_GetOutput	SKB_SIGNATURE_ALGORITHM_ECDSA SKB_SIGNATURE_ALGORITHM_ECDSA_SHA1 SKB_SIGNATURE_ALGORITHM_ECDSA_SHA256 SKB_TRANSFORM_TYPE_SIGN SKB_ECC_CURVE_NIST_192 SKB_ECC_CURVE_NIST_224 SKB_ECC_CURVE_NIST_256 SKB_ECC_CURVE_NIST_384 SKB_ECC_CURVE_NIST_521
HMAC	SKB_FIPS_Engine_CreateTransform SKB_FIPS_Transform_AddSecureData SKB_FIPS_Transform_AddBytes SKB_FIPS_Transform_GetOutput	SKB_SIGNATURE_ALGORITHM_HMAC_SHA1 SKB_SIGNATURE_ALGORITHM_HMAC_SHA256 SKB_TRANSFORM_TYPE_SIGN SKB_TRANSFORM_TYPE_VERIFY
CMAC	SKB_FIPS_Engine_CreateTransform SKB_FIPS_Transform_AddSecureData SKB_FIPS_Transform_AddBytes SKB_FIPS_Transform_GetOutput	SKB_SIGNATURE_ALGORITHM_AES_128_CMAC SKB_TRANSFORM_TYPE_SIGN SKB_TRANSFORM_TYPE_VERIFY
Random byte generation	SKB_FIPS_Engine_GenerateSecureData	SKB_DATA_TYPE_BYTES
Compute ECDH key agreement shared secret**	SKB_FIPS_Engine_CreateKeyAgreement SKB_FIPS_KeyAgreement_GetPublicKey SKB_FIPS_KeyAgreement_ComputeSecret	SKB_KEY_AGREEMENT_ALGORITHM_ECDH SKB_DATA_FORMAT_ECC_BINARY SKB_ECC_CURVE_NIST_192 SKB_ECC_CURVE_NIST_224 SKB_ECC_CURVE_NIST_256 SKB_ECC_CURVE_NIST_384 SKB_ECC_CURVE_NIST_521

Service	API Function	API Enumeration Value
Generation of ECDSA key pair**	SKB_FIPS_Engine_GenerateSecureData SKB_FIPS_SecureData_GetPublicKey	SKB_DATA_TYPE_ECC_PRIVATE_KEY SKB_DATA_FORMAT_ECC_BINARY SKB_ECC_CURVE_NIST_192 SKB_ECC_CURVE_NIST_224 SKB_ECC_CURVE_NIST_256 SKB_ECC_CURVE_NIST_384 SKB_ECC_CURVE_NIST_521
Key derivation	SKB_FIPS_SecureData_Derive	SKB_DERIVATION_ALGORITHM_NIST_800_108_COUNTER_CMAC_AES128
Key unwrap	SKB_FIPS_Engine_CreateDataFromWrapped	SKB_CIPHER_ALGORITHM_NIST_AES SKB_DATA_TYPE_BYTES SKB_DATA_FORMAT_RAW
Import	SKB_FIPS_Engine_CreateDataFromExported	
Export	SKB_FIPS_SecureData_Export	SKB_EXPORT_TARGET_CROSS_ENGINE SKB_EXPORT_TARGET_PERSISTENT
Key/service context zeroization	SKB_FIPS_Engine_Release SKB_FIPS_SecureData_Release SKB_FIPS_Transform_Release SKB_FIPS_Cipher_Release SKB_FIPS_KeyAgreement_Release	

*** Services impacted by the NIST SP 800-131A algorithm transitions. It is the responsibility of the module operator to ensure that algorithms, modes, and key sizes Disallowed per NIST SP 800-131A are not used.*

Table 10 – Module services to API mapping in approved mode of operation

Glossary

Term	*Description*
AES	Advanced Encryption Standard
API	Application programming interface
CBC	Cipher block chaining
CMAC	Cipher-Based Message Authentication Code (SP 800-38B)
CO	Crypto officer
CPU	Central processing unit
CSP	Critical security parameter
CTR	Counter
DRBG	Deterministic random bit generator
EC	Elliptic curve
ECB	Electronic codebook
ECC CDH	Elliptic curve cryptography cofactor Diffie-Hellman
ECDSA	Elliptic Curve Digital Signature Algorithm (FIPS 186-3)
FIPS	Federal Information Processing Standard
GPC	General purpose computer
HMAC-SHA-1 HMAC-SHA-256	Keyed-Hash Message Authentication Code (FIPS 198-1)
IV	Initialization vector
KAT	Known answer test
KDF	Key derivation function
MAC	Message authentication code
NIST	National Institute of Standards and Technology (USA)
OAEP	Optimal asymmetric encryption padding
OS	Operating system
PCT	Pair-wise consistency test

Term	***Description***
PKCS	Public-Key Cryptography Standard
RSA	Rivest, Shamir and Adleman algorithm (FIPS 186-3)
SECG	Standards for Efficient Cryptography Group
SEI	Source entropy input
SHA-1, SHA-256, SHA-384	Secure Hash Algorithm (FIPS 180-3)
SHS	Secure Hashing Standard
SKB	Secure Key Box by whiteCryption

Table 9 – Glossary