Kasten Chase Cryptographic Engine

Security Policy

Synopsis: This document is the non-proprietary security policy of the Kasten Chase Cryptographic Engine (KCCE) version 2.0. KCCE is a FIPS 140-2 Level 1 and Level 2 (as a function of the target environments) validated cryptographic module that allows for the secure implementation of cryptographic protocols. KCCE provides a reliable and secure Application Programming Interface (API) allowing software developers to build secure cryptographic applications.

Table of Contents

1. Introduction

1.1 Purpose

This document is the non-proprietary security policy for the Kasten Chase Cryptographic Engine (KCCE) Version 2.0. This security policy describes how KCCE meets the requirements of FIPS 140-2 Level 1 for specified target environments and FIPS 140-2 Level 2 for other specified target environments and how applications using KCCE may operate in a FIPS 140-2 mode.

Its purpose is to specify the identification and authentication, access control, physical security, mitigation against specific attacks policies and to specify rules for secure operation.

FIPS 140-2 is a US Government publication describing the requirements for cryptographic modules. Additional information can be obtained on the NIST website: http://csrc.nist.gov/

The Kasten Chase Cryptographic Engine is a shared library that provides for the secure implementation of cryptographic protocols within an application program. KCCE provides a reliable and secure Application Programming Interface (API) enabling software developers to build secure cryptographic applications.

1.2 References

Ref.	Document Identification	Document Title
[1]	KCCE Application Programming Interface	Kasten Chase Cryptographic Engine Application Programming Interface Specification
[2]	RSA Data Security, Inc. Public Key Cryptography Standards (PKCS)	PKCS #5 v2.0: Password-Based Cryptography Standard, March 25, 1999
[3]	csrc.nist.gov/CryptoToolkit/kms/key-wrap.pdf	AES Key Wrap Specification, 17 November 2001
[4]		Shamir, A, How to Share a Secret, Communications of the ACM Vol.22 No. 11, November 1979
[5]	KCCE Finite State Model	Kasten Chase Cryptographic Engine (KCCE) Finite State Model

2. Cryptographic Module Definition

KCCE consists of the following three general components:

- One of the commercially available, general-purpose hardware computing platforms listed in Table 1 (for FIPS 140-2 Level 1 validation) and in Table 2 (for FIPS 140-2 Level 2 validation). A high level view of the target computing platform and the cryptographic boundary is shown in Figure 1.
- The target platform's associated operating system listed in Table 1 (for FIPS 140-2 Level 1 validation) and in Table 2 (for FIPS 140-2 Level 2 validation).
- An independent library of cryptographic object modules, which are linked to an application via the KCCE Application Programming Interface (API), that executes within the specified operating system and on the specified computing platform.

The FIPS 140-2 cryptographic boundary for KCCE from a hardware perspective is shown in Figure 1.

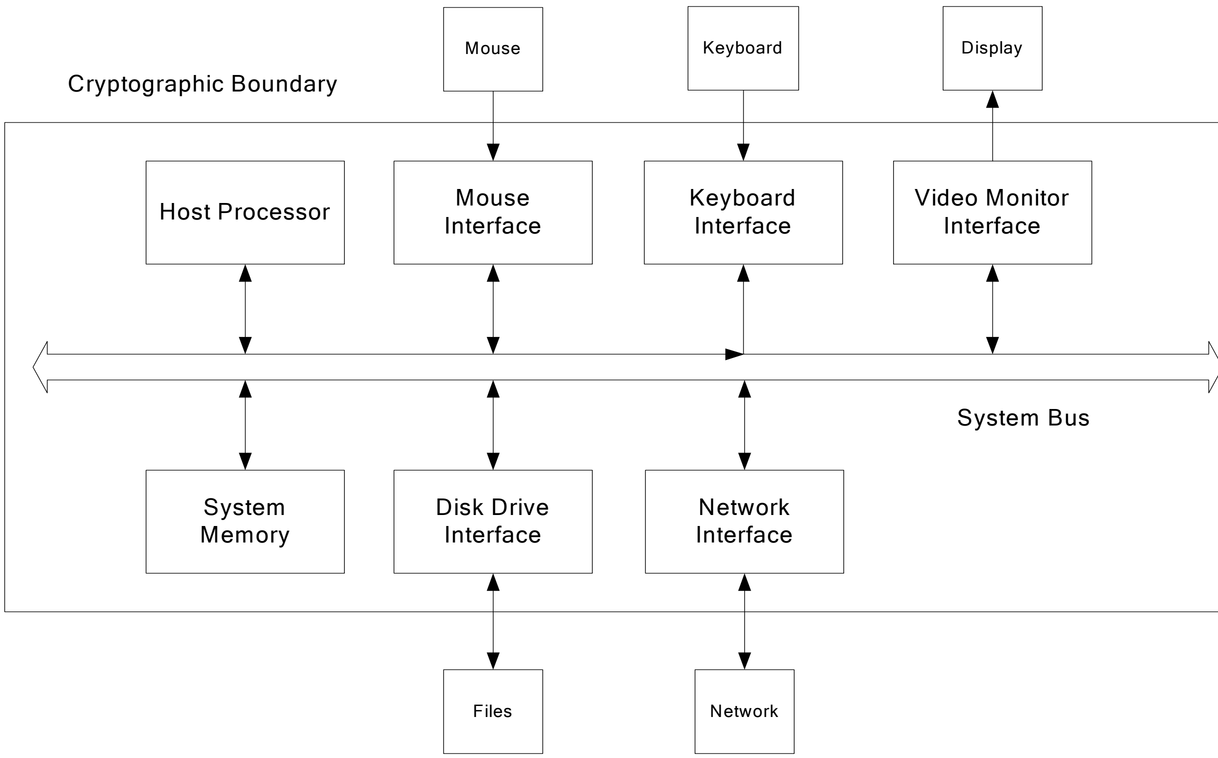

Figure 1 KCCE Cryptographic Boundary from a Hardware Perspective

Designed and developed by Kasten Chase, the Kasten Chase Cryptographic Engine (KCCE) was written in the C programming language with the exception of performance-critical functions that were written in the host platform's assembly language. With the exception of the platform-specific software integrity test, the source code of KCCE is independent of the target environment. For each platform, KCCE is compiled into an independent module.

KCCE is linked to an application via the KCCE Application Programming Interface (API). The API is detailed in Ref [1]. The application program uses the KCCE API in order to gain access to the cryptographic services provided by KCCE. Figure 2 is a block diagram of a typical KCCE implementation and shows KCCE's cryptographic boundary from a software perspective.

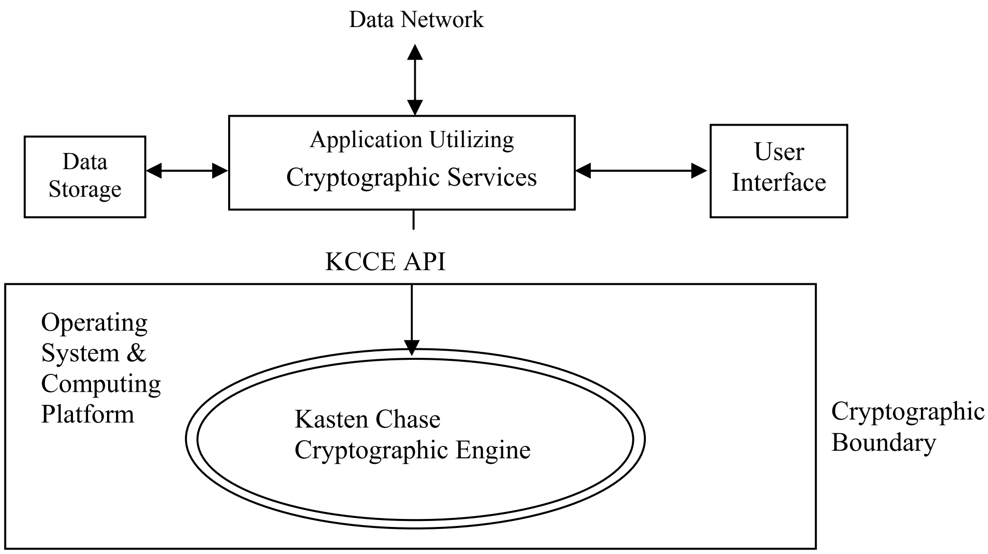

Figure 2: Typical KCCE Environment from a Software Perspective

2.1 Target Environments

For the purpose of FIPS 140-2, KCCE is implemented as a multi-chip stand-alone module, and as such has been tested upon the target environments listed in Table 1.

Table 1 FIPS 140-2 Level 1 Target Environments

Target Platform	Operating System	Mode
IBM Compatible PC with a x86 processor	Windows™2000	User Mode
IBM Compatible PC with a x86 processor	Windows™ 2000	Kernel Mode
IBM Compatible PC with a x86 processor	Linux ver 2.4	User Mode
IBM Compatible PC with a x86 processor	Linux with 2.4 kernel	Kernel Mode
IBM server with a PowerPC processor	AIX V5.2	User Mode
IBM server with a PowerPC processor	AIX V5.2	Kernel Mode
Sun SPARC™ processor based systems	Sun Solaris™ version 9	User Mode
Sun SPARC™ processor based systems	Sun Solaris™ version 9	Kernel mode
Sun SPARC™ processor based systems	Sun Trusted Solaris™ Version 8 4/01	Kernel mode

Table 2 lists the target environments on which KCCE has been tested for FIPS 140-2 Level 2, together with their Common Criteria Evaluated Assurance Level (EAL) and Protection Profile.

Note 1: KCCE has been validated with Win2000 but also remains FIPS 140-2 compliant when used with WIN98/ME/XP/2003.

Nore 2: All operating systems specified for level 1 are configured in single-user mode.

Table 2: FIPS 140-2 Level Two Target Environments, EAL and Protection Profile

Target Platform	Operating System	Mode	Common Criteria Assurance Level	Protection Profile
Sun SPARC™ processor based systems	Sun Trusted Solaris™ Version 8 4/01	User mode	EAL 4 (note 1)	CAPP
IBM Compatible PC with a x86 processor	Windows™ 2000 Professional, Server and Advanced Server with SP3 and Hotfix Q326886	User Mode	EAL 4 Augmented (note 2)	CAPP v1.d, 10/8/99
IBM eServer pSeries systems with a PowerPC™ processor	AIX 5L for POWER V5.2	User Mode	EAL 4+ (note 3)	CAPP v1.d

Note 1: Certification report available at http://www.cesg.gov.uk/site/iacs/itsec/media/certreps/CRP170v3.pdf

Note 2: Validation report available at http://niap.nist.gov/cc-scheme/st/ST_VID4002-VR.pdf

Note 3: Certification report available at http://www.bsi.bund.de/zertifiz/zert/reporte/0217a.pdf

2.2 FIPS 140-2 Security Level

KCCE is validated to meet the FIPS 140-2 security requirements at the levels listed in Table 3 for the target environments listed in Table 1.

Table 3 KCCE Security Levels for Level 1 Target Environments

FIPS 140-2 Security Requirements Section	Level
1. Cryptographic Module Specification	1
2. Cryptographic Module, Ports and Interfaces	1
3. Roles, Services and Authentication	3
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	2
11. Mitigation of Other Attacks	N/A

KCCE is validated to meet the FIPS 140-2 security requirements at the levels listed in Table 4 for the target environments listed in Table 2.

Table 4: KCCE Security Levels for Level 2 Target Environments

FIPS 140-2 Security Requirements Section	Level
1. Cryptographic Module Specification	2
2. Cryptographic Module, Ports and Interfaces	2
3. Roles, Services and Authentication	3
4. Finite State Model	2
5. Physical Security	N/A
6. Operational Environment	2
7. Cryptographic Key Management	2
8. EMI / EMC	2
9. Self Tests	2
10. Design Assurance	2
11. Mitigation of Other Attacks	N/A

2.3 KCCE Interfaces

KCCE operates within the confines of a host-computing platform with physical interfaces consisting of a mouse, keyboard, serial ports, network adaptors and a monitor.

KCCE's logical interface is defined by its Application Programming Interface and is specified in Ref [1]. Control input to KCCE is made through the API calls. Data input to KCCE and output from KCCE is through the arguments of the API functions. Each API function returns a status or an error value – the meaning of which is detailed in Ref [1].

2.4 Finite State Model

KCCE behaves in accordance with the finite state model described in Ref [5].

3. Security Policy

This chapter contains the security policy for the Kasten Chase Cryptographic Engine as identified in chapter 2.
KCCE operates under several rules that form the basis for its security policy:

- Cryptographic services are permitted only after authentication via login. Login requires the presentation of Private Key material that is wrapped using a symmetric key derived in part from a User supplied Passphrase. The PBKDF2 passphrase-based key derivation algorithm, as specified in PKCS#5 Ref [2], is used. Although the private key material is wrapped, it must be considered as plaintext with regard to its handling or transmission over a network since the key encrypting the private key material is not generated by a FIPS Approved method.
- The provided cryptographic services have permission restrictions based on the role of the user, which in turn is determined at login. A matrix designating the functions available for each role is presented in Section 5. Table 14.
- No private key of an asymmetric key pair is permitted outside the cryptographic boundary in the clear. A private key can only be exported from KCCE after being wrapped using a symmetric key. In instances where this symmetric key is derived from a passphrase using the PBKDF2 algorithm as specified in PKCS#5, then the exported private key must be considered as plaintext with regard to its handling or transmission over a network since the key wrapping the private key is not generated by a FIPS Approved method.

- No symmetric key is permitted outside the cryptographic boundary in the clear. A symmetric key can only be exported from KCCE after being wrapped using a symmetric key derived from a shared secret computation.
- When keys are imported or exported, the wrapping algorithm conforms to the technique described in AES Key Wrap Specification Ref [3], regardless of the symmetric algorithm specified in the function call.
- The Default Security Officer selects and sets the Public Key Method prior to the creation of a Root Security Officer's key material. Ref [1] enumerates the cryptographic algorithms that are recommended based on the selected Public Key Method.
- The Default Security Officer must re-seed the random number generator before the Default Security Officer generates key material. It is also recommended that a Security Officer re-seed the random number generator prior to generating key material.
- A context must be terminated by calling one of: Logout, Finalize or Zeroize functions. Among these, the Logout function is preferred as it both zeroizes the context and updates the key material with additional random data.
- The cryptographic application program accessing KCCE must obscure the feedback of authentication (passphrase) data to the operator of the application program.

3.1 FIPS 140-2 Approved Operational Modes

KCCE operates in FIPS mode at all times except when one of the following conditions exists:

- An application makes a call to the K_Hash_HASHALG_Init or K_HMAC_Init functions (see Ref [1]) with MD5 as the specified algorithm. Then, the KCCE context goes into a non-FIPS mode of operation and remains in a non-FIPS mode until the hash termination function, K_HASHALG_FINALIZE or HMAC termination function, K_HMAC_Finalize is called and completes.
- An application makes a call to the asymmetric encryption initialization function K_Asymmetric_Encrypt_Init (see Ref [1]) or decryption initialization function K_Asymmetric_Decrypt_Init (see Ref [1]). The KCCE context goes into a non-FIPS mode of operation and remains in a non-FIPS mode until the context is terminated.
- The key derived from K_Generate_Key_Passphrase is intended to be used as a symmetric session key between two communicating parties. The KCCE context goes into a non-FIPS mode of operation upon the application call to K_Generate_Key_Passphrase and remains in a non-FIPS mode until the context is terminated.

Keys generated in a FIPS mode of operation may not be used in a non-FIPS mode of operation and keys generated in a non-FIPS mode of operation may not be used in a FIPS mode of operation.

3.2 Roles and Authentication

There are three roles within the Kasten Chase Cryptographic Engine - Default Security Officer, Security Officer and User. The roles are identified within the key material required for Login to KCCE. Each role requires a user identifier and a passphrase. Only one role may be active per context.

3.2.1 Default Security Officer

Along with the object libraries for KCCE, Kasten Chase also provides Key Material for the Default Security Officer. This Key Material has a globally known username and passphrase. The services provided to this role are limited to those needed to create Root Security Officer Key Material. This role is used to "bootstrap" the creation of key material for an organization and for this reason the key material should be physically protected. KCCE cannot be used to create Default Security Officer key material.

3.2.2 Security Officer

While formally within KCCE there is only one Security Officer role, it is strongly recommended that in practice there are two – a Root Security Officer and a Local Security Officer. The distinction between the two has to do with what is recommended they do, rather than what they may do.

3.2.2.1 Root Security Officer

The Root Security Officer for an organization is a Security Officer that only creates private/public key pairs and signs public keys of Local Security officers. The Root Security Officers private key material should be physically protected because it is from the signatures generated by the root's public key that all trust is derived for a community of users. A copy of the Root Security Officer's Public Key is included in all subsequently created Private Key Material.

3.2.2.2 Local Security Officer

Local Security Officers for an organization are Security Officers that only create private-public key pairs for Users and sign their public keys.

3.2.3 User

Users have full access to the suite of cryptographic algorithms; however, some of their choices may be limited based on selections made by the Security Officers. Users are not permitted to sign public keys, generate key material, or establish the public key method to use. These functions are reserved for the Security Officers and/or the Default Security Officer.

3.2.4 User Authentication

A passphrase (Security Officer or User) must have at least eight ASCII characters and less than 256 characters.

Login must occur before any cryptographic service is made available through the API. Login includes the presentation of Private Key material.

After 10 consecutive failures to login using an incorrect passphrase, KCCE is zeroized and must be initialized before another login may be attempted

3.3 Installation Guidance

3.3.1 KCCE FIPS 140-2 Level 1 Environments

For the KCCE FIPS 140-2 Level 1 Environments, identified in Table 1, installation guidance can be found in the API [Ref 1].

3.3.2 Physical Security Policy for KCCE FIPS 140-2 Level 2 Environments

For the KCCE FIPS 140-2 Level 2 Environments, identified in Table 2, tamper-evident labels must be applied to locations on the host computing platform enclosure such that any attempt to penetrate the host computing platform would be evident by broken seals or residual adhesive from removed seals. The tamper-evident seals must be controlled by the local security officer so that seals may not be removed, the platform penetrated, and new seals reapplied. A typical use of the seals is shown in Figure 3. Additional installation requirements are described in API [Ref 1].

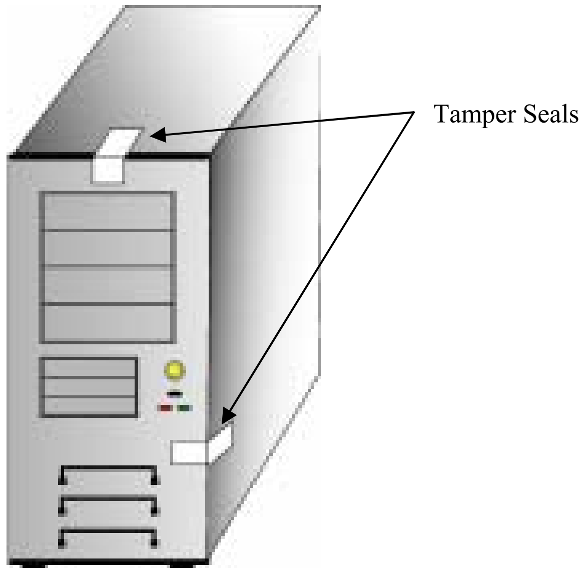

Figure 3: Location of Tamper-Evident Seals on Level 2 computing Platforms

3.4 Cryptographic Key Management

3.4.1 Critical Security Parameters

KCCE employs the critical security parameters listed in Table 5.

Table 5 Critical Security Parameters

Key and Cryptographic Sensitive Parameters
Asymmetric private keys
Symmetric keys - Session or message encryption keys (MEKs) and Token encryption keys (TEKs)
Shared Secrets
Passphrases
Split-secret share
HMAC Keys
Random Number Seeds

Each time an application invokes the 'initialize' function, K_Initialize, a unique KCCE context is created within which there are neither keys nor CSPs. After 'initialization' keys can be imported by two specific function calls (Login, Import). Login imports a private key (Security Officer or User) and stores the key within KCCE while the Import function imports a key of a specified type and also stores it within KCCE.

It is not possible to extract a key from KCCE in plaintext.

3.4.2 Key Generation

Keys can be generated from a random number by the following processes:

- using either of the functions: Generate_Key (see Ref [1]) or Generate_Key_Pair(see Ref [1]),
- derived from a key exchange computation using either of the functions: Shared_Secret (see Ref [1]) or Generate_TEK (see Ref [1]),

- or derived from a passphrase and random salt using Generate_Key_Passphrase (see Ref [1]).

Calls to the following API-level functions generate keys as indicated:

- Generate_Key_Pair – generates an asymmetric key pair of the same type as the login key material, stores the key pair within KCCE and provides a reference for subsequent use of the key pair,
- Generate_Key – generates a symmetric key for a specified symmetric algorithm, stores the key within KCCE and provides a reference for subsequent use,
- Shared_Secret – generates a shared secret, stores it within KCCE and provides a reference for subsequent use,
- Generate_TEK – generates a token encryption key, stores it within KCCE and provides a reference for subsequent use.
- Generate_Key_Passphrase – derives a key from a passphrase following the method described in Ref [2] (employs the ANSI X9.62 deterministic random number generator), stores the key within KCCE and provides a reference for subsequent use.

3.4.3 Key Entry and Output

Symmetric Keys can be exported by the function Export_Key (see Ref [1]) or imported by the function Import_Key (see Ref [1]). The function Export_Key wraps the key to be exported and exports the wrapped key. Import_Key imports a wrapped key and unwraps the imported key.

3.4.4 Key Storage

KCCE does not provide persistent storage of keys. Keys exported from KCCE can be safely stored in their encrypted form, but this is outside the scope of the cryptographic module.

3.4.5 Key Archival

KCCE does not directly archive cryptographic keys. Keys can be exported and safely stored in their encrypted form; however, management of the archival of such a key is the responsibility of the user or the application that is using KCCE.

3.4.6 Key Destruction

All keys (regardless of context) are destroyed by the Zeroize function (see Ref [1]).
All keys associated with a specific context are destroyed by the Finalize function (see Ref [1]).
A specific key within a context is destroyed by the Delete Key function (see Ref [1]).

3.4.7 Secret Sharing

A secret (key) may be split into an arbitrary number of shares such that subsets of those shares (containing a pre-determined number of shares) can be used to recover the original secret. One's key material wrapping key (derived from the passphrase) can be split and this mechanism used to recover the wrapping key if one's passphrase is forgotten.

3.5 Operational Environment

KCCE operates within a general purpose operational environment

3.5.1 Level 1 Mode of Operation

For KCCE to operate in a FIPS mode each of the FIPS 140-2 Level 1 target environments (listed in Table 1) must be configured as a single user mode of operation.

The operating system must be configured such that the KCCE shared library is only allowed to support one application program.

The operating system must be configured to ensure that no other process can interrupt the execution of KCCE.

The software must be installed in the secure manner. The specific installation and O/S configuration instructions are described in Ref [1].

3.5.2 Level 2 Mode of Operation

KCCE must be installed on one of the FIPS 140-2 Level 2 target environments (listed in Table 2). Specific installation instructions are found in Ref [1].

The cryptographic application linked to KCCE must output the following events to an operating system log:

- Attempts to provide invalid input for all Default Security Officer (DSO) and Security Officer (SO) functions,
- The addition or deletion of user to/from a DSO or SO role
- Access to all DSO and SO functions
- Logins to KCCE
- Attempts by a User to implement a DSO or SO function

In addition to the requirements identified in section 3.5.1 all Level 2 target environment's operating systems must be configured to provide an audit mechanism to record modifications, accesses, deletions, and additions of crytographic data and CSPs. In addition, Level 2 target environment's operating systems must provide operations to process audit data stored in the audit trail.

3.6 Self Tests

In order to ensure that KCCE is functioning properly, KCCE performs power-up (Initialization) tests upon invocation. KCCE also executes conditional self-tests whenever certain conditions exist.

The power-up (Initialization) tests consist of cryptographic algorithm tests and a software integrity test. Cryptographic algorithm or "known answer" tests are executed for the following cryptographic algorithms:

- ECC public key cryptography algorithm
- RSA public key cryptography algorithm
- AES symmetric algorithm
- Triple DES symmetric algorithm
- SHA1, SHA256, SHA384 and SHA512 hashing algorithms
- HMAC-SHA-1, HMAC-SHA256, HMAC-SHA384 and HMAC-SHA512 keyed hashing algorithms
- ANSI X9.62 Random Number Generator

In the cases where there is more than one key size for a specified algorithm, only the largest key size is tested. The power-up tests must be performed and passed before any cryptographic function / service can be used. In addition to performing these tests on 'power-up', with the exception of the software integrity they are also performed when the K_Self_Test function is invoked.

In the context of the KCCE, "Power-up" occurs the first time an application allocates a context to perform cryptographic functions (i.e. the K_Initialize function is called). If the power-up tests fail, then a context is not allocated and KCCE enters an error state that prevents all other cryptographic functions from performing operations. If the power-up tests pass a success indicator and a context ID are returned to the calling application. If a subsequent direct call to K_Self_Test fails then the error state is also entered.

KCCE executes pair wise consistency tests when K_Generate_Key_Pair is called, continuous random generator tests when the random number generator is invoked

If KCCE fails a self-test, the following occurs: 1) an error state is entered 2) an error indicator is output (via status output interface) to the cryptographic application program, 3) KCCE is 'zeroized' - causing all data input and output to and from KCCE to be inhibited and constituting a Major Error State.

3.7 Mitigation Against Specific Attacks

KCCE is not designed to mitigate any known specific attacks to the Cryptographic Module.

3.8 Administrative Services

3.8.1 Initialize

This administrative service initializes a new context and, if the first call to initialize, performs a self-test of internal cryptographic functions. A valid context must be created before any cryptographic service can be accessed.

3.8.2 Finalize

This administrative service securely terminates a KCCE session (context). The service zeroizes all CSPs and private data before freeing all space allocated to the specific context. This service must be executed before the user application terminates.

3.8.3 Self Test

This service causes the power-up self tests, with the exception of the software integrity test, to execute. If a self-test fails all the contexts are zeroized.

3.8.4 Zeroize

This service zeroizes the CSPs and private data of all contexts and prevents any further use of the same instance of KCCE.

3.8.5 Login

This service authenticates the session by unwrapping the Key Material to obtain the private key, state information and known data. Login has to be successfully executed before any cryptographic service is provided.

3.8.6 Logout

This service closes the specific session and securely stores persistent data – user's private key, state of the random number generator - in the user's Key Material. The Key Material is wrapped using a derived key generated during the login process from the user's passphrase and login name. Although the private key material is wrapped, it must be considered as plaintext with regard to its handling or transmission over a network since the key encrypting the private key material is not generated by a FIPS Approved method.

3.9 Key Management Services

3.9.1 Set Public Key Method

This service is only available to the Default Security Officer. It must be executed prior to the generation of the Private/Public Key Pair for the Root Security Officer. The selected Public Key Method becomes the defined Public Key method for all Key Material generated from this Root Security Officer and subordinate Security Officers.

3.9.2 Generate Key Pair

This service generates an Elliptic Curve Cryptography (ECC) or RSA key pair and stores the private key internally.

3.9.3 Generate Key Material

This service generates key material to be given to a Security Officer or a User. The Security Officer or User will then be able to use the key material data to login to KCCE. This function will return an error if there has not been a prior call to seed the random number generator. The key material exported by this service is wrapped using a symmetric key derived in part from a user passphrase according to the PBKDF2 algorithm Although the private key material is wrapped, it must be considered as plaintext with regard to its handling or transmission over a network since the key encrypting the private key material is not generated by a FIPS Approved method.

3.9.4 Delete Key

This service zeroizes a specific internal key that is no longer needed

3.9.5 Set User Passphrase

This service changes the passphrase associated with the Key Material from the previous Login. The actual change does not take place until the Logout function is completed.

3.9.6 Generate Key

This service generates a key appropriate for use with a symmetric encryption/decryption algorithm and stores the key internally.

3.9.7 Export Key

This service wraps (encrypts) an internal key for exporting and subsequent secure transfer to another party. In instances where the wrapping key is derived from a passphrase using the PBKDF2 algorithm as specified in PKCS#5, then the exported key must be considered as plaintext with regard to its handling or transmission over a network since the key wrapping the private key is not generated by a FIPS Approved method.

3.9.8 Import Key

This service imports a wrapped key and unwraps (decrypts) it for internal use.

3.9.9 Generate Key From a Passphrase

This service generates a key from a user-supplied passphrase according to the method described in Ref [2] and stores the key internally.

3.10 Key Exchange Services

3.10.1 Generate a Shared Secret

This service computes a shared secret. The shared secret can be used as a key for encryption or decryption with a symmetric encryption algorithm. A shared secret generated by this service cannot be used to generate a token encryption key. This function is only accessible if ECC is the public key method used by the calling user. In this case, ECDH is performed.

3.10.2 Generate Token Encryption Key

This service generates a Token Encryption Key (TEK) appropriate to wrap or unwrap a key for symmetric encryption / decryption algorithm.

3.11 Data Encryption and Decryption Services

3.11.1 Symmetric Encryption

The symmetric encryption service encrypts plaintext using one of the algorithms and modes listed in Table 6.

Table 6 Symmetric Encryption / Decryption Functions Supported By KCCE

Symmetric Function	Modes	Block Size (bytes)	Key Size (bits)	FIPS Approved Algorithm
AES128	ECB, CBC	16	128	Yes
AES192	ECB, CBC	16	192	Yes
AES256	ECB, CBC	16	256	Yes
TDES	ECB, CBC	8	168*	Yes

*Note: due to the nature of the TDES algorithm the key's effective strength is 112 bits

3.11.2 Symmetric Decryption

The symmetric decryption service decrypts ciphertext using one of the algorithms and modes listed in Table 6.

3.11.3 Asymmetric Encryption

The asymmetric encryption service encrypts plaintext using one of the asymmetric algorithms listed in Table 7. Use of this asymmetric encryption service puts KCCE into a non-FIPS mode of operation.

Table 7 Asymmetric Encryption / Decryption Functions Supported by KCCE

Asymmetric Function	Key Size (bits)
ECC_NISTK163	163
ECC_NISTK233	233
ECC_NISTK283	283
ECC_NISTK409	409
ECC_NISTK571	571
ECC_NISTB163	163
ECC_NISTB233	233
ECC_NISTB283	283
ECC_NISTB409	409
ECC_NISTB571	571
ECC_NISTP192	192
ECC_NISTP224	224
ECC_NISTP256	256
ECC_NISTP384	384
ECC_NISTP521	521
RSA_512	512
RSA_1024	1024
RSA_2048	2048
RSA_4096	4096

3.11.4 Asymmetric Decryption

The asymmetric decryption service decrypts ciphertext or an Internal Key using one of the asymmetric algorithms listed in Table 7. Use of this asymmetric decryption service puts KCCE into a non-FIPS mode of operation.

3.12 Hashing and Digital Signature Services

3.12.1 Hash Services

The hash service creates a message digest of a data entity using one of the hashing algorithms listed in Table 8.

Table 8 Hash Functions Supported by KCCE

Hashing Function	Digest Size [bits]	FIPS Approved Algorithm
SHA1	160	Yes
SHA256	256	Yes
SHA384	384	Yes
SHA512	512	Yes
MD5	128	No

3.12.2 Keyed-Hash Services

The keyed-hash service creates a message digest of a secret key and a data entity using one of the methods listed in Table 9.

Table 9 Key-Hash Services

Hashing Function	Digest Size [bits]	FIPS Approved Algorithm
HMAC-SHA-1	160	Yes
HMAC-SHA256	256	Yes
HMAC-SHA384	384	Yes
HMAC-SHA512	512	Yes
HMAC-MD5	128	No

3.12.3 Digital Signature

KCCE supports the Digital Signature algorithms listed in Table 10

Table 10 Digital Signature Algorithms Supported by KCCE

Digital Signature Algorithm	FIPS Approved Algorithm
ECDSA (ANSI X9.62) Curves sizes: K163, K233, K283, K409, K571, B163, B233, B283, B409, B571, P192, P224, P256, P384, P521	Yes
RSA (PKCS#1) Key sizes: 512*, 1024, 2048, 4096	Yes

*RSA keys of size smaller than 1024 are not recommended for use in FIPS mode.
The digital signature service signs the data entity (typically a message digest) with a private key.

3.12.4 Verification of a Signed Data Entity

This service verifies the signature of a signed data entity (typically a message digest) with the appropriate public key.

3.12.5 Signing a Public Key

This security officer service signs a public key (User or Security Officer) and appends the public key of the signer to the signature.

3.12.6 Verification of the Signature of a Public Key

This service verifies the signature chain associated with a signed public key.

3.13 Key Exchange Services

The key exchange service provides for the secure exchange of a symmetric encryption key between two parties. KCCE supports the key exchange protocols listed in Table 11.

Table 11 Key Exchange Protocols Supported by KCCE

Key Exchange Protocol
ECDH
KEA

3.14 Secret Sharing Services

The secret sharing service securely splits using Shamir's method described in Ref [4].

3.14.1 Splitting a Secret into Shares

This service splits a secret into a number of shares such that a pre-determined number of these shares can be used to recover the secret.

3.14.2 Recovering a Split Secret

This service recovers the split secret from a subset of the split secret's shares.

3.14.3 Recovering Key Material

One must use the "Recovering a Split Secret" service to recover one's key material wrapping key before using this service. The Recovering Key Material service generates new key material from pre-existing key material by changing the passphrase.

3.15 Random Number Services

KCCE uses one of the FIPS approved Deterministic Random Number Generators allowed by FIPS 186-2 and listed in Table 12

Table 12 Pseudo Random Number Generator Employed by KCCE

Pseudo Random Number Generation	FIPS Approved
ANSI X9.62	Yes

3.15.1 Seeding the Random Number Generator

This service adds entropy to the existing seed (originally derived from the private key material and internal system specific information) for the Pseudo Random Number Generator. Before the Default Security Officer can generate new key material the Pseudo Random Number Generator must be re-seeded.

3.15.2 Generate a Random Number

This service generates a random number of a requested size (number of bits). The random number generated by this service cannot be used as a key.

4. KCCE Cryptographic Algorithms

KCCE supports the following cryptographic algorithms listed in Table 13

Table 13 KCCE Cryptographic Algorithms

FIPS Approved Cryptographic Algorithms:	Other Cryptographic Algorithms
AES	
SHA-1, SHA-256, SHA-384, SHA-512	MD5
HMAC-SHA-1, HMAC-SHA-256, HMAC- SHA 384 HMAC-SHA-512	HMAC-MD5
Triple-DES	
RSA, ECDSA	KEA, ECDH

5. Cryptographic Services, Roles and Access to CSPs and Keys

The cryptographic services available to the three KCCE defined roles and the CSPs and Keys available to the role for the specificed service are presented in Table 14. Passphrases and random number seeds are available, to the specified role, in the clear while private key material, imported keys, exported keys that are available to the role are wrapped. For some functions the right-most three columns have been merged into one cell where this occurs the information in the merged cells applies to DSO, SO and User access rights to Keys/CSPs and authentication mechanism.

Table 14 Cryptographic Services, Roles and Access to CSPs Matrix

Service	Approved security functions used by service	Cryptographic Keys/CSPs accessed by service	DSO access rights to Keys/CSPs and authentication mechanism	SO access rights to Keys/CSPs and authentication mechanism	User access rights to Keys/CSPs and authentication mechanism
K_Initialize*	SHA-1, SHA-256, SHA-384, SHA-512, AES, TDES, HMAC-SHA-1, HMAC-SHA-256, HMAC-SHA-384, HMAC-SHA-512, RSA, ECDSA, RNG	None	No CSP access No authentication		
K_Finalize	None	AES keys(w), TDES keys(w), HMAC-SHA-1 keys(w), HMAC-SHA-256 keys(w), HMAC-SHA-384 keys(w), HMAC-SHA-512 keys(w)RSA private key(w), ECDSA private key(w), RNG state(w)	AES keys(w), TDES keys(w), HMAC-SHA-1 keys(w), HMAC-SHA-256 keys(w), HMAC-SHA-384 keys(w), HMAC-SHA-512 keys(w), RSA private key(w), ECDSA private key(w), RNG state(w) No authentication		

Service	Approved security functions used by service	Cryptographic Keys/CSPs accessed by service	DSO access rights to Keys/CSPs and authentication mechanism	SO access rights to Keys/CSPs and authentication mechanism	User access rights to Keys/CSPs and authentication mechanism
K_Self_Test*	SHA-1, SHA-256, SHA-384, SHA-512, AES, TDES, HMAC-SHA-1, HMAC-SHA-256, HMAC-SHA-384, HMAC-SHA-512, RSA, ECDSA, RNG	None	None No authentication	None No authentication	None No authentication
K_Get_Role	None	None	None Authenticated state if context authentication flag is set	None Authenticated state if context authentication flag is set	None Authenticated state if context authentication flag is set
K_Get_Public_ Key_Method	None	None	None Authenticated state if context authentication flag is set	None Authenticated state if context authentication flag is set	None Authenticated state if context authentication flag is set
K_Zeroize	None	AES keys(w), TDES keys(w), HMAC-SHA-1 keys(w), , HMAC-SHA256 key(w), HMAC-SHA384 key(w), HMAC-SHA512 key(w), RSA private key(w), ECDSA private key(w), RNG(w) state	AES keys(w), TDES keys(w), HMAC-SHA-1 keys(w), HMAC-SHA-256 keys(w), HMAC-SHA-384 keys(w), HMAC-SHA-512 keys(w), RSA private key(w), ECDSA private key(w), RNG state(w) No Authentication		

Service	Approved security functions used by service	Cryptographic Keys/CSPs accessed by service	DSO access rights to Keys/CSPs and authentication mechanism	SO access rights to Keys/CSPs and authentication mechanism	User access rights to Keys/CSPs and authentication mechanism
K_Login	AES, TDES, HMAC-SHA-1, HMAC-SHA-256, HMAC-SHA-384, HMAC-SHA-512, RNG	RSA private key(rw), ECDSA private key(rw), RNG(rw)	RSA private key(rw), ECDSA private key(rw), RNG(rw) Authentication with passphrase, username and key material	RSA private key(rw), ECDSA private key(rw), RNG(rw) Authentication with passphrase, username and key material	RSA private key(rw), ECDSA private key(rw), RNG(rw) Authentication with passphrase, username and key material
K_Logout	AES, TDES, HMAC-SHA-1, HMAC-SHA-256, HMAC-SHA-384, HMAC-SHA-512, RNG	RSA private key(rw), ECDSA private key(rw), RNG(rw)	RSA private key(rw), ECDSA private key(rw), RNG(rw) Authenticated state if context authentication flag is set	RSA private key(rw), ECDSA private key(rw), RNG(rw) Authenticated state if context authentication flag is set	RSA private key(rw), ECDSA private key(rw), RNG(rw) Authenticated state if context authentication flag is set
K_Set_Public_ Key_Method	None	None	No access to CSPs Authenticated state if context authentication flag is set	N/A	N/A
K_Generate_Key_ Pair*	RSA, ECDSA, RNG	RSA private key(rw), ECDSA private key(rw), RNG state(rw), RSA public key(rw), ECDSA public key(rw)	RSA private key(rw), ECDSA private key(rw), RNG state(rw), RSA public key(rw), ECDSA public key(rw) Authenticated state if context authentication flag is set	RSA private key(rw), ECDSA private key(rw), RNG state(rw), RSA public key(rw), ECDSA public key(rw) Authenticated state if context authentication flag is set	RSA private key(rw), ECDSA private key(rw), RNG state(rw), RSA public key(rw), ECDSA public key(rw) Authenticated state if context authentication flag is set

Service	Approved security functions used by service	Cryptographic Keys/CSPs accessed by service	DSO access rights to Keys/CSPs and authentication mechanism	SO access rights to Keys/CSPs and authentication mechanism	User access rights to Keys/CSPs and authentication mechanism
K_Generate_Key_ Material*	AES, TDES, HMAC-SHA-1, HMAC-SHA-256, HMAC-SHA-384, HMAC-SHA-512, RNG	RSA private key(r), ECDSA private key(r), RNG state(r)	RSA private key(r), ECDSA private key(r), RNG state(r) Authenticated state if context authentication flag is set	RSA private key(r), ECDSA private key(r), RNG state(r) Authenticated state if context authentication flag is set	N/A
K_Delete_Key	None	AES key(w), TDES key(w), RSA private key(w), ECDSA private key(w), RSA public key(rw), ECDSA public key(rw), HMAC-SHA256 key(rw), HMAC-SHA384 key(rw), HMAC-SHA512 key(rw), HMAC-SHA-1 keys(rw)	AES key(w), TDES key(w), RSA private key(w), ECDSA private key(w), RSA public key(rw), ECDSA public key(rw) , HMAC-SHA256 key(rw), HMAC-SHA384 key(rw), HMAC-SHA512 key(rw), HMAC-SHA-1 keys(rw) Authenticated state if context authentication flag is set	AES key(w), TDES key(w), RSA private key(w), ECDSA private key(w), RSA public key(rw), ECDSA public key(rw) , HMAC-SHA256 key(rw), HMAC-SHA384 key(rw), HMAC-SHA512 key(rw), HMAC-SHA-1 keys(rw) Authenticated state if context authentication flag is set	AES key(w), TDES key(w), RSA private key(w), ECDSA private key(w), RSA public key(rw), ECDSA public key(rw) , HMAC-SHA256 key(rw), HMAC-SHA384 key(rw), HMAC-SHA512 key(rw), HMAC-SHA-1 keys(rw) Authenticated state if context authentication flag is set
K_Set_User_ Passphrase	AES, TDES, HMAC-SHA-1, HMAC-SHA-1, HMAC-SHA-256, HMAC-SHA-384, HMAC-SHA-512, RNG	RSA private key(rw), ECDSA private key(rw), RNG state(rw)	N/A	RSA private key(rw), ECDSA private key(rw), RNG state(rw) Authenticated state if context authentication flag is set	RSA private key(rw), ECDSA private key(rw), RNG state(rw) Authenticated state if context authentication flag is set

Service	Approved security functions used by service	Cryptographic Keys/CSPs accessed by service	DSO access rights to Keys/CSPs and authentication mechanism	SO access rights to Keys/CSPs and authentication mechanism	User access rights to Keys/CSPs and authentication mechanism
K_Generate_Key*	RNG	AES key(w), TDES key(w), RNG state(rw) , HMAC-SHA256 key(rw), HMAC-SHA384 key(rw), HMAC-SHA512 key(rw), HMAC-SHA-1 keys(rw)	N/A	N/A	AES key(w), TDES key(w), RNG state(rw) , HMAC-SHA256 key(rw), HMAC-SHA384 key(rw), HMAC-SHA512 key(rw), HMAC-SHA-1 keys(rw) Authenticated state if context authentication flag is set
K_Export_Key	AES	AES key(r), TDES key(r), RSA private key(rw), ECDSA private key(rw) , HMAC-SHA256 key(rw), HMAC-SHA384 key(rw), HMAC-SHA512 key(rw), HMAC-SHA-1 keys(rw)	N/A	N/A	AES key(r), TDES key(r), RSA private key(rw), ECDSA private key(rw) , HMAC-SHA256 key(rw), HMAC-SHA384 key(rw), HMAC-SHA512 key(rw), HMAC-SHA-1 keys(rw) Authenticated state if context authentication flag is set

Service	Approved security functions used by service	Cryptographic Keys/CSPs accessed by service	DSO access rights to Keys/CSPs and authentication mechanism	SO access rights to Keys/CSPs and authentication mechanism	User access rights to Keys/CSPs and authentication mechanism
K_Import_Key	AES	AES key(rw), TDES key(rw), RSA private key(rw), ECDSA private key(rw) , HMAC-SHA256 key(rw), HMAC-SHA384 key(rw), HMAC-SHA512 key(rw), HMAC-SHA-1 keys(rw)	N/A	N/A	AES key(rw), TDES key(rw), RSA private key(rw), ECDSA private key(rw) , HMAC-SHA256 key(rw), HMAC-SHA384 key(rw), HMAC-SHA512 key(rw), HMAC-SHA-1 keys(rw) Authenticated state if context authentication flag is set
K_Generate_Key_ Passphrase	HMAC-SHA-1, HMAC-SHA-256, HMAC-SHA-384, HMAC-SHA-512	AES key(w), TDES key(w)	N/A	AES key(w), TDES key(w) Authenticated state if context authentication flag is set	AES key(w), TDES key(w) Authenticated state if context authentication flag is set
K_Shared_Secret (Uses ECDH)	SHA-1	AES key(w), TDES key(w), EC private key(r)	N/A	N/A	AES key(w), TDES key(w), EC private key(r) Authenticated state if context authentication flag is set

Service	Approved security functions used by service	Cryptographic Keys/CSPs accessed by service	DSO access rights to Keys/CSPs and authentication mechanism	SO access rights to Keys/CSPs and authentication mechanism	User access rights to Keys/CSPs and authentication mechanism
K_Generate_TEK (Uses KEA)	SHA-1	AES key(w), TDES key(w), EC private key(r)	N/A	N/A	AES key(w), TDES key(w), EC private key(r) Authenticated state if context authentication flag is set
K_Encrypt_ SYMALG_Init	AES, TDES	AES key(rw), TDES key(rw)	N/A	N/A	AES key(rw), TDES key(rw) Authenticated state if context authentication flag is set
K_Encrypt_ SYMALG_ Update	None	None	N/A	N/A	None Authenticated state if context authentication flag is set
K_Encrypt_ SYMALG_ Finalize	None	AES key(w), TDES key(w)	N/A	N/A	AES key(w), TDES key(w) Authenticated state if context authentication flag is set
K_Asymmetric_ Encrypt_Init	SHA-1, SHA256, SHA384, SHA512, HMAC-SHA-1, HMAC-SHA-256, HMAC-SHA-384, HMAC-SHA-512	ECDSA private key(rw), RSA public key(rw), ECDSA public key(rw)	N/A	N/A	ECDSA private key(rw), RSA public key(rw), ECDSA public key(rw) Authenticated state if context authentication flag is set

Service	Approved security functions used by service	Cryptographic Keys/CSPs accessed by service	DSO access rights to Keys/CSPs and authentication mechanism	SO access rights to Keys/CSPs and authentication mechanism	User access rights to Keys/CSPs and authentication mechanism
K_Asymmetric_ Encrypt_Update	AES, HMAC-SHA-1, HMAC-SHA-256, HMAC-SHA-384, HMAC-SHA-512	None	N/A	N/A	None Authenticated state if context authentication flag is set
K_Asymmetric_ Encrypt_ Finalize	HMAC-SHA-1, HMAC-SHA-256, HMAC-SHA-384, HMAC-SHA-512	None	N/A	N/A	None Authenticated state if context authentication flag is set
K_Asymmetric_ Encrypt_ Secret	SHA-1, SHA256, SHA384, SHA512, HMAC-SHA-1, HMAC-SHA-256, HMAC-SHA-384, HMAC-SHA-512, AES	ECDSA private key(rw), AES key(rw), TDES key(rw), RSA public key(rw), ECDSA public key(rw)	N/A	N/A	ECDSA private key(rw), AES key(rw), TDES key(rw), RSA public key(rw), ECDSA public key(rw) Authenticated state if context authentication flag is set
K_Decrypt_ SYMALG_Init	AES, TDES	AES key(rw), TDES key(rw)	N/A	N/A	AES key(rw), TDES key(rw) Authenticated state if context authentication flag is set

Service	Approved security functions used by service	Cryptographic Keys/CSPs accessed by service	DSO access rights to Keys/CSPs and authentication mechanism	SO access rights to Keys/CSPs and authentication mechanism	User access rights to Keys/CSPs and authentication mechanism
K_Decrypt_ SYMALG_ Update	None	None	N/A	N/A	None Authenticated state if context authentication flag is set
K_Decrypt_ SYMALG_ Finalize	None	AES key(w), TDES key(w)	N/A	N/A	AES key(w), TDES key(w) Authenticated state if context authentication flag is set
K_Asymmetric_ Decrypt_Init	SHA-1, SHA256, SHA384, SHA512, HMAC-SHA-1, HMAC-SHA-256, HMAC-SHA-384, HMAC-SHA-512	RSA private key(rw), ECDSA private key(rw), , ECDSA public key(rw)	N/A	N/A	RSA private key(rw), ECDSA private key(rw), ECDSA public key(rw) Authenticated state if context authentication flag is set
K_Asymmetic_ Decrypt_Update	AES, HMAC-SHA-1, HMAC-SHA-256, HMAC-SHA-384, HMAC-SHA-512	None	N/A	N/A	None Authenticated state if context authentication flag is set
K_Asymmetric_ Decrypt_Finalize	HMAC-SHA-1, HMAC-SHA-256, HMAC-SHA-384, HMAC-SHA-512	None	N/A	N/A	None Authenticated state if context authentication flag is set

Service	Approved security functions used by service	Cryptographic Keys/CSPs accessed by service	DSO access rights to Keys/CSPs and authentication mechanism	SO access rights to Keys/CSPs and authentication mechanism	User access rights to Keys/CSPs and authentication mechanism
K_Asymmetric_ Decrypt_Secret	SHA-1, SHA256, SHA384, SHA512, HMAC-SHA-1, HMAC-SHA-256, HMAC-SHA-384, HMAC-SHA-512, AES	RSA private key(rw), ECDSA private key(rw), AES key(rw), TDES key(rw), ECDSA public key(rw)	N/A	N/A	RSA private key(rw), ECDSA private key(rw), AES key(rw), TDES key(rw), ECDSA public key(rw) Authenticated state if context authentication flag is set
K_Hash_ HASHALG_Init	SHA1, SHA256, SHA384, SHA512	None	N/A	N/A	None Authenticated state if context authentication flag is set
K_Hash_ HASHALG_ Update	SHA1, SHA256, SHA384, SHA512	None	N/A	N/A	None Authenticated state if context authentication flag is set
K_Hash_ HASHALG_ Finalize	SHA1, SHA256, SHA384, SHA512	None	N/A	N/A	None Authenticated state if context authentication flag is set

Service	Approved security functions used by service	Cryptographic Keys/CSPs accessed by service	DSO access rights to Keys/CSPs and authentication mechanism	SO access rights to Keys/CSPs and authentication mechanism	User access rights to Keys/CSPs and authentication mechanism
K_HMAC_ Init	HMAC-SHA-1, HMAC-SHA-256, HMAC-SHA-384, HMAC-SHA-512	AES key(r), TDES key(r), RSA private key(r), ECDSA private key(r) AES and TDES keys are used as keys for the HMAC algorithms	N/A	N/A	AES key(rw), TDES key(rw) AES and TDES keys are used as keys for the HMAC algorithms Authenticated state if context authentication flag is set
K_HMAC_ Update	HMAC-SHA-1, HMAC-SHA-256, HMAC-SHA-384, HMAC-SHA-512	None	N/A	N/A	None Authenticated state if context authentication flag is set
K_HMAC_ Finalize	HMAC-SHA-1, HMAC-SHA-256, HMAC-SHA-384, HMAC-SHA-512	None	N/A	N/A	None Authenticated state if context authentication flag is set
K_Sign*	RSA, ECDSA, SHA1, SHA256, SHA384, SHA512, RNG	RSA private key(rw), ECDSA private key(rw), RNG state(rw)	N/A	N/A	RSA key(rw), ECDSA key(rw), RNG state(rw) Authenticated state if context authentication flag is set

Service	Approved security functions used by service	Cryptographic Keys/CSPs accessed by service	DSO access rights to Keys/CSPs and authentication mechanism	SO access rights to Keys/CSPs and authentication mechanism	User access rights to Keys/CSPs and authentication mechanism
K_Sign_Public_ Key*	RSA, ECDSA, SHA1, SHA256, SHA384, SHA512	RSA private key(rw), ECDSA private key(rw), RSA public key(rw), ECDSA public key(rw), RNG state(rw)	N/A	RSA private key(rw), ECDSA private key(rw), RSA public key(rw), ECDSA public key(rw), RNG state(rw) Authenticated state if context authentication flag is set	N/A
K_Verify	RSA, ECDSA, SHA1, SHA256, SHA384, SHA512	RSA public key(rw), ECDSA public key(rw)	N/A	N/A	RSA public key(rw), ECDSA public key(rw) Authenticated state if context authentication flag is set
K_Verify_Signed_ Public_Key	RSA, ECDSA, SHA1, SHA256, SHA384, SHA512	RSA public key(rw), ECDSA public key(rw)	N/A	N/A	RSA public key(rw), ECDSA public key(rw) Authenticated state if context authentication flag is set
K_Split_Secret	RNG	RNG state(rw), AES key(rw), TDES key(rw), RSA private key(rw), ECDSA private key(rw)	N/A	N/A	RNG state(rw), AES key(rw), TDES key(rw), RSA private key(rw), ECDSA private key(rw) Authenticated state if context authentication flag is set

Service	Approved security functions used by service	Cryptographic Keys/CSPs accessed by service	DSO access rights to Keys/CSPs and authentication mechanism	SO access rights to Keys/CSPs and authentication mechanism	User access rights to Keys/CSPs and authentication mechanism
K_Recover_Secret	None	AES key(rw), TDES key(rw), RSA private key(rw), ECDSA private key(rw)	N/A	N/A	AES key(rw), TDES key(rw), RSA private key(rw), ECDSA private key(rw) Authenticated state if context authentication
K_Recover_Key_ Material	HMAC-SHA-1, HMAC-SHA-256, HMAC-SHA-384, HMAC-SHA-512, RNG, AES, TDES	RSA private key(rw), ECDSA private key(rw), RNG state(rw)	N/A	N/A	RSA private key(rw), ECDSA private key(rw), RNG state(rw) Authenticated state if context authentication
K_Seed_Random	None	RNG state(rw), RNG seed(rw)	RNG state(rw), RNG seed(rw) Authenticated state if context authentication flag is set	RNG state(rw), RNG seed(rw) Authenticated state if context authentication flag is set	RNG state(rw), RNG seed(rw) Authenticated state if context authentication flag is set
K_Generate_ Random*	RNG, SHA1	RNG state(rw)	N/A	N/A	RNG state(rw) Authenticated state if context authentication flag is set

Within the table “r” implies read access, “w” implies write access and “rw” implies read write access.

∗ These functions execute self-tests or conditional self-tests, either explicitly or implicitly (internally).