Supporting Document
Mandatory Technical Document

collaborative Protection Profile for Dedicated Security Component
Version: 1.0
2020-09-10
National Information Assurance Partnership

Foreword

This is a Supporting Document (SD), intended to complement the Common Criteria version 3 and the associated Common Evaluation Methodology for Information Technology Security Evaluation.

SDs may be "Guidance Documents", that highlight specific approaches and application of the standard to areas where no mutual recognition of its application is required, and as such, are not of normative nature, or "Mandatory Technical Documents", whose application is mandatory for evaluations whose scope is covered by that of the SD. The usage of the latter class is not only mandatory, but certificates issued as a result of their application are recognized under the CCRA.

Technical Editor:
National Information Assurance Partnership (NIAP)

Document history:

Version	Date	Comment
1.0	2020-09-10	First published release version.
1.0x	2021-04-06	Start of first XML version.

General Purpose:
The purpose of this SD is to define evaluation methods for the functional behavior of Dedicated Security Components products.

Acknowledgements:
This SD was developed with support from NIAP Dedicated Security Components Technical Community members, with representatives from industry, government agencies, Common Criteria Test Laboratories, and members of academia.

Table of Contents

1 Introduction

1.1 Technology Area and Scope of Supporting Document

The scope of the collaborative Protection Profile for Dedicated Security Component is to describe the security functionality of Dedicated Security Components products in terms of [CC] and to define functional and assurance requirements for them.

Although Evaluation Activities are defined mainly for the evaluators to follow, in general they also help developers to prepare for evaluation by identifying specific requirements for their TOE. The specific requirements in Evaluation Activities may in some cases clarify the meaning of Security Functional Requirements (SFR), and may identify particular requirements for the content of Security Targets (ST) (especially the TOE Summary Specification), user guidance documentation, and possibly supplementary information (e.g. for entropy analysis or cryptographic key management architecture).

1.2 Structure of the Document

Evaluation Activities can be defined for both SFRs and Security Assurance Requirements (SAR), which are themselves defined in separate sections of the SD.

If any Evaluation Activity cannot be successfully completed in an evaluation, then the overall verdict for the evaluation is a 'fail'. In rare cases there may be acceptable reasons why an Evaluation Activity may be modified or deemed not applicable for a particular TOE, but this must be approved by the Certification Body for the evaluation.

In general, if all Evaluation Activities (for both SFRs and SARs) are successfully completed in an evaluation then it would be expected that the overall verdict for the evaluation is a 'pass'. To reach a 'fail' verdict when the Evaluation Activities have been successfully completed would require a specific justification from the evaluator as to why the Evaluation Activities were not sufficient for that TOE.

Similarly, at the more granular level of assurance components, if the Evaluation Activities for an assurance component and all of its related SFR Evaluation Activities are successfully completed in an evaluation then it would be expected that the verdict for the assurance component is a 'pass'. To reach a 'fail' verdict for the assurance component when these Evaluation Activities have been successfully completed would require a specific justification from the evaluator as to why the Evaluation Activities were not sufficient for that TOE.

1.3 Terms

The following sections list Common Criteria and technology terms used in this document.

1.3.1 Common Criteria Terms

Assurance	Grounds for confidence that a TOE meets the SFRs [CC].
Base Protection Profile (Base-PP)	Protection Profile used as a basis to build a PP-Configuration.
Common Criteria (CC)	Common Criteria for Information Technology Security Evaluation (International Standard ISO/IEC 15408).
Common Criteria Testing Laboratory	Within the context of the Common Criteria Evaluation and Validation Scheme (CCEVS), an IT security evaluation facility, accredited by the National Voluntary Laboratory Accreditation Program (NVLAP) and approved by the NIAP Validation Body to conduct Common Criteria-based evaluations.
Common Evaluation Methodology (CEM)	Common Evaluation Methodology for Information Technology Security Evaluation.
Distributed TOE	A TOE composed of multiple components operating as a logical whole.
Operational Environment (OE)	Hardware and software that are outside the TOE boundary that support the TOE functionality and security policy.
Protection	

Profile (PP)	An implementation-independent set of security requirements for a category of products.
Protection Profile Configuration (PP-Configuration)	A comprehensive set of security requirements for a product type that consists of at least one Base-PP and at least one PP-Module.
Protection Profile Module (PP-Module)	An implementation-independent statement of security needs for a TOE type complementary to one or more Base Protection Profiles.
Security Assurance Requirement (SAR)	A requirement to assure the security of the TOE.
Security Functional Requirement (SFR)	A requirement for security enforcement by the TOE.
Security Target (ST)	A set of implementation-dependent security requirements for a specific product.
TOE Security Functionality (TSF)	The security functionality of the product under evaluation.
TOE Summary Specification (TSS)	A description of how a TOE satisfies the SFRs in an ST.
Target of Evaluation (TOE)	The product under evaluation.

1.3.2 Technical Terms

Address Space Layout Randomization (ASLR)	An anti-exploitation feature which loads memory mappings into unpredictable locations. ASLR makes it more difficult for an attacker to redirect control to code that they have introduced into the address space of a process.
Administrator	An administrator is responsible for management activities, including setting policies that are applied by the enterprise on the operating system. This administrator could be acting remotely through a management server, from which the system receives configuration policies. An administrator can enforce settings on the system which cannot be overridden by non-administrator users.
Application (app)	Software that runs on a platform and performs tasks on behalf of the user or owner of the platform, as well as its supporting documentation.
Application Programming Interface (API)	A specification of routines, data structures, object classes, and variables that allows an application to make use of services provided by another software component, such as a library. APIs are often provided for a set of libraries included with the platform.
Credential	Data that establishes the identity of a user, e.g. a cryptographic key or password.
Critical Security Parameters (CSP)	Information that is either user or system defined and is used to operate a cryptographic module in processing encryption functions including cryptographic keys and authentication data, such as passwords, the disclosure or modification of which can compromise the security of a cryptographic module or the security of the information protected by the module.
DAR Protection	Countermeasures that prevent attackers, even those with physical access, from extracting data from non-volatile storage. Common techniques include data encryption and wiping.
Data Execution Prevention (DEP)	An anti-exploitation feature of modern operating systems executing on modern computer hardware, which enforces a non-execute permission on pages of memory. DEP prevents pages of memory from containing both data and instructions, which makes it more difficult for an attacker to introduce and execute code.
Developer	An entity that writes OS software. For the purposes of this document, vendors and developers are the same.
General Purpose Operating System	A class of OSes designed to support a wide-variety of workloads consisting of many concurrent applications or services. Typical characteristics for OSes in this class include support for third-party applications, support for multiple users, and security separation between users and their respective resources. General Purpose Operating Systems also lack the real-time constraint that defines Real Time Operating Systems (RTOS). RTOSes typically power routers, switches, and embedded devices.

Host-based Firewall	A software-based firewall implementation running on the OS for filtering inbound and outbound network traffic to and from processes running on the OS.
Operating System (OS)	Software that manages physical and logical resources and provides services for applications. The terms *TOE* and *OS* are interchangeable in this document.
Personally Identifiable Information (PII)	Any information about an individual maintained by an agency, including, but not limited to, education, financial transactions, medical history, and criminal or employment history and information which can be used to distinguish or trace an individual's identity, such as their name, social security number, date and place of birth, mother's maiden name, biometric records, etc., including any other personal information which is linked or linkable to an individual.[OMB]
Sensitive Data	Sensitive data may include all user or enterprise data or may be specific application data such as PII, emails, messaging, documents, calendar items, and contacts. Sensitive data must minimally include credentials and keys. Sensitive data shall be identified in the OS's TSS by the ST author.
User	A user is subject to configuration policies applied to the operating system by administrators. On some systems under certain configurations, a normal user can temporarily elevate privileges to that of an administrator. At that time, such a user should be considered an administrator.
Virtual Machine (VM)	Blah Blah Blah

2 Evaluation Activities for SFRs

The EAs presented in this section capture the actions the evaluator performs to address technology specific aspects covering specific SARs (e.g. ASE_TSS.1, ADV_FSP.1, AGD_OPE.1, and ATE_IND.1) – this is in addition to the CEM work units that are performed in Section 3 Evaluation Activities for SARs.

Regarding design descriptions (designated by the subsections labelled TSS, as well as any required supplementary material that may be treated as proprietary), the evaluator must ensure there is specific information that satisfies the EA. For findings regarding the TSS section, the evaluator's verdicts will be associated with the CEM work unit ASE_TSS.1-1. Evaluator verdicts associated with the supplementary evidence will also be associated with ASE_TSS.1-1, since the requirement to provide such evidence is specified in ASE in the PP.

For ensuring the guidance documentation provides sufficient information for the administrators/users as it pertains to SFRs, the evaluator's verdicts will be associated with CEM work units ADV_FSP.1-7, AGD_OPE.1-4, and AGD_OPE.1-5.

Finally, the subsection labelled Tests is where the authors have determined that testing of the product in the context of the associated SFR is necessary. While the evaluator is expected to develop tests, there may be instances where it is more practical for the developer to construct tests, or where the developer may have existing tests. Therefore, it is acceptable for the evaluator to witness developer-generated tests in lieu of executing the tests. In this case, the evaluator must ensure the developer's tests are executing both in the manner declared by the developer and as mandated by the EA. The CEM work units that are associated with the EAs specified in this section are: ATE_IND.1-3, ATE_IND.1-4, ATE_IND.1-5, ATE_IND.1-6, and ATE_IND.1-7.

2.1 TOE SFR Evaluation Activities

2.1.1 Cryptographic Support (FCS)

FCS_CKM.1 Cryptographic Key Generation

TSS
The evaluator shall examine the TSS to verify that it describes how the TOE obtains a cryptographic key through importation of keys from external sources as specified in FDP_ITC_EXT.1 and FDP_ITC_EXT.2. The evaluator shall also examine the TSS to determine whether it describes any supported asymmetric or symmetric key generation functionality consistent with the claims made in FCS_CKM.1.1.
Guidance
The evaluator shall verify that the guidance instructs the administrator how to configure the TOE to use the selected key types for all uses identified in the ST.
KMD
The evaluator shall confirm that the KMD describes:

- The parsing interface and how the TSF imports keys for internal use
- The asymmetric key generation interfaces and how the TSF internally creates asymmetric keys, if claimed
- The symmetric key generation interfaces and how the TSF internally creates symmetric keys, if claimed

If the TOE uses the generated key in a key chain/hierarchy then the KMD shall describe how the key is used as part of the key chain/hierarchy.
Tests
Testing for this function is performed in conjunction with FDP_ITC_EXT.1 and FDP_ITC_EXT.2. If asymmetric

or symmetric key generation functionality is claimed, testing for this function is also performed in conjunction with FCS_CKM.1/AK or FCS_CKM.1/SK.

FCS_CKM.1/AK Cryptographic Key Generation (Asymmetric Keys)

TSS
The evaluator shall examine the TSS to verify that it describes how the TOE generates an asymmetric key based on the methods selected from cPP Table 13: "Supported Methods for Asymmetric Key Generation". The evaluator shall examine the TSS to verify that it describes how the TOE invokes the methods selected in the ST from the same table. The evaluator shall examine the TSS to verify that it identifies the usage for each row identifier (key type, key size, and list of standards) selected in the ST.

Guidance
The evaluator shall verify that the AGD guidance instructs the administrator how to configure the TOE to use the selected key types for all uses identified in the ST.

KMD
If the TOE uses the generated key in a key chain/hierarchy then the evaluator shall confirm that the KMD describes:

- If <u>AK1</u> is selected, then the KMD describes which methods for generating p and q are used
- How the key is used as part of the key chain/hierarchy.

Tests
The following tests require the developer to provide access to a test platform that provides the evaluator with tools that are typically not found on factory products.

AK1: RSA Key Generation
The below tests are derived from The 186-4 RSA Validation System (RSA2VS), Updated 8 July 2014, Section 6.2, from the National Institute of Standards and Technology.

The evaluator shall verify the implementation of RSA Key Generation by the TOE using the Key Generation test. This test verifies the ability of the TSF to correctly produce values for the key components including the public verification exponent e, the private prime factors p and q, the public modulus n and the calculation of the private signature exponent d.

FIPS 186-4 Key Pair generation specifies 5 methods for generating the primes p and q.

These are:

1. Random Primes:
    - Provable primes
    - Probable primes
2. Primes with Conditions:
    - Primes p1, p2, q1, q2, p and q shall all be provable primes.
    - Primes p1, p2, q1, and q2 shall be provable primes and p and q shall be probable primes
    - Primes p1, p2, q1, q2, p and q shall all be probable primes.

To test the key generation method for the Random Provable primes method and for all the Primes with Conditions methods, the evaluator must seed the TSF key generation routine with sufficient data to deterministically generate the RSA key pair.

For each key length supported, the evaluator shall have the TSF generate 25 key pairs. The evaluator shall verify the correctness of the TSF's implementation by comparing values generated by the TSF with those generated by a known good implementation using the same input parameters.

If the TOE generates Random Probable Primes then if possible, the Random Probable primes method should also be verified against a known good implementation as described above. If verification against a known good implementation is not possible, the evaluator shall have the TSF generate 25 key pairs for each supported key length nlen and verify that all of the following are true:

- $n = p*q$
- p and q are probably prime according to Miller-Rabin tests with error probability $<2^{(-125)}$
- $2^{16} < e < 2^{256}$ and e is an odd integer
- $GCD(p-1,e) = 1$
- $GCD(q-1,e) = 1$
- $|p-q| > 2^{(nlen/2 - 100)}$
- $p >= squareroot(2)*(2^{(nlen/2 -1)})$
- $q >= squareroot(2)*(2^{(nlen/2 -1)})$
- $2^{(nlen/2)} < d < LCM(p-1,q-1)$
- $e*d = 1 \mod LCM(p-1,q-1)$

AK2 & AK3: ECC Key Generation with NIST and Brainpool Curves

These tests are derived from The FIPS 186-4 Elliptic Curve Digital Signature Algorithm Validation System (ECDSA2VS), Updated 18 Mar 2014, Section 6.

ECC Key Generation Test

For each selected curve, and for each key pair generation method as described in FIPS 186-4, section B.4, the evaluator shall require the implementation under test to generate 10 private/public key pairs (d, Q). The private key, d, shall be generated using a random bit generator as specified in FCS_RBG_EXT.1. The private key, d, is used to compute the public key, Q'. The evaluator shall confirm that 0<d<n (where n is the order of the group), and the computed value Q' is then compared to the generated public/private key pairs' public key, Q, to confirm that Q is equal to Q'.

Public Key Validation (PKV) Test

For each supported curve, the evaluator shall generate 12 private/public key pairs using the key generation function of a known good implementation and modify six of the public key values so that they are incorrect, leaving six values unchanged (i.e., correct). To determine correctness, the evaluator shall submit the 12 key pairs to the public key validation (PKV) function of the TOE and shall confirm that the results correspond as expected to the modified and unmodified values.

AK4: DSA Key Generation using Finite-Field Cryptography (FFC)

The evaluator shall verify the implementation of the Parameters Generation and the Key Generation for FFC by the TOE using the Parameter Generation and Key Generation test. This test verifies the ability of the TSF to correctly produce values for the field prime p, the cryptographic prime q (dividing p-1), the cryptographic group generator g, and the calculation of the private key x and public key y.

The Parameter generation specifies 2 ways (or methods) to generate the cryptographic prime q and the field prime p:

- Primes q and p shall both be provable primes
- Primes q and field prime p shall both be probable primes

and two ways to generate the cryptographic group generator g:

- Generator g constructed through a verifiable process
- Generator g constructed through an unverifiable process.

The Key generation specifies 2 ways to generate the private key x:

- len(q) bit output of RBG where $1 \le x \le q-1$
- len(q) + 64 bit output of RBG, followed by a mod q-1 operation and a +1 operation, where $1 \le x \le q-1$.

The security strength of the RBG must be at least that of the security offered by the FFC parameter set.

To test the cryptographic and field prime generation method for the provable primes method or the group generator g for a verifiable process, the evaluator must seed the TSF parameter generation routine with sufficient data to deterministically generate the parameter set.

For each key length supported, the evaluator shall have the TSF generate 25 parameter sets and key pairs. The evaluator shall verify the correctness of the TSF's implementation by comparing values generated by the TSF with those generated from a known good implementation. Verification must also confirm

- g != 0,1
- q divides p-1
- $g^q \bmod p = 1$
- $g^x \bmod p = y$

for each FFC parameter set and key pair.

AK5: Curve25519 Key Generation

The evaluator shall require the implementation under test (IUT) to generate 10 private/public key pairs. The private key shall be generated as specified in RFC 7748 using an approved random bit generator (RBG) and shall be written in littleendian order (least significant byte first. To determine correctness, the evaluator shall submit the generated key pairs to the public key verification (PKV) function of a known good implementation.

Note: Assuming the PKV function of the good implementation will (using little-endian order):

- Confirm the private and public keys are 32-byte values
- Confirm the three least significant bits of the first byte of the private key are zero
- Confirm the most significant bit of the last byte is zero
- Confirm the second most significant bit of the last byte is one
- Calculate the expected public key from the private key and confirm it matches the supplied public key

The evaluator shall generate 10 private/public key pairs using the key generation function of a known good implementation and modify 5 of the public key values so that they are incorrect, leaving five values unchanged (i.e. correct). The evaluator shall obtain in response a set of 10 PASS/FAIL values.

FCS_CKM.1/SK Cryptographic Key Generation (Symmetric Encryption Key)

TSS
The evaluator shall examine the TSS to verify that it describes how the TOE obtains an SK through direct generation as specified in FCS_RBG_EXT.1, FCS_COP.1/KDF, or FCS_COP.1/PBKDF. The evaluator shall review the TSS to verify that it describes how the ST invokes the functionality described by FCS_RBG_EXT.1 and FCS_COP.1/PBKDF where applicable.

[conditional] If the symmetric key is generated by an RBG, the evaluator shall review the TSS to determine that it describes how the functionality described by FCS_RBG_EXT.1 is invoked. The evaluator uses the description of the RBG functionality in FCS_RBG_EXT.1 or documentation available for the operational environment to determine that the key size being requested is greater than or equal to the key size and mode to be used for the encryption/decryption of the data.

Guidance
The evaluator shall verify that the AGD guidance instructs the administrator how to configure the TOE to use the selected key types for all uses identified in the ST.

KMD

The evaluator shall confirm that the KMD describes, as applicable:

- The RBG interface and how the ST uses it in symmetric key generation
- The KDF interface and how the ST uses it in symmetric key generation
- The PBKDF interface and how the ST uses it in symmetric key generation
- If the TOE uses the generated key in a key chain/hierarchy then the KMD shall describe how the ST uses the key as part of the key chain/hierarchy.

Tests

For each selected key generation method, the evaluator shall configure the selected generation capability. The evaluator shall use the description of the RBG interface to verify that the TOE requests and receives an amount of RBG output greater than or equal to the requested key size. The evaluator shall perform the tests as described for FCS_COP.1/KDF and FCS_COP.1/PBKDF.

FCS_CKM.1/KEK Cryptographic Key Generation (Key Encryption Key)

TSS

The evaluator shall examine the key hierarchy section of the TSS to ensure that the formation of all KEKs is described and that the key sizes match that described by the ST author. The evaluator shall examine the key hierarchy section of the TSS to ensure that each KEK encrypts keys of equal or lesser security strength using one of the selected methods.

[conditional] If the KEK is generated according to an asymmetric key scheme, the evaluator shall review the TSS to determine that it describes how the functionality described by FCS_CKM.1/AK is invoked. The evaluator uses the description of the key generation functionality in FCS_CKM.1/AK or documentation available for the operational environment to determine that the key strength being requested is greater than or equal to 112 bits.

[conditional] If the KEK is generated according to a symmetric key scheme, the evaluator shall review the TSS to determine that it describes how the functionality described by FCS_CKM.1/SK is invoked. The evaluator uses the description of the RBG functionality in FCS_RBG_EXT.1, or the key derivation functionality in either FCS_CKM_EXT.5 or FCS_COP.1/PBKDF, depending on the key generation method claimed, to determine that the key size being requested is greater than or equal to the key size and mode to be used for the encryption/decryption of the data.

[conditional] If the KEK is formed from derivation, the evaluator shall verify that the TSS describes the method of derivation and that this method is consistent with FCS_CKM_EXT.5.

Guidance

There are no guidance evaluation activities for this component.

KMD

The evaluator shall iterate through each of the methods selected by the ST and confirm that the KMD describes the applicable selected methods.

Tests

The evaluator shall iterate through each of the methods selected by the ST and perform all applicable tests from the selected methods.

FCS_CKM.2 Cryptographic Key Establishment

TSS

The evaluator shall examine the TSS to ensure that ST supports at least one key establishment scheme. The evaluator also ensures that for each key establishment scheme selected by the ST in FCS_CKM.2.1 it also supports one or more corresponding methods selected in FCS_COP.1/KAT. If the ST selects RSA in FCS_CKM.2.1, then the TOE must support one or more of "KAS1," or "KAS2," "KTS-OAEP," from FCS_COP.1/KAT. If the ST selects elliptic curve-based, then the TOE must support one or more of "ECDH-NIST" or "ECDH-BPC" from FCS_COP.1/KAT. If the ST selects Diffie-Hellman-based key establishment, then the TOE must support "DH" from FCS_COP.1/KAT.

Guidance

The evaluator shall verify that the guidance instructs the administrator how to configure the TOE to use the selected key establishment scheme.

KMD

There are no KMD evaluation activities for this component.

Tests

Testing for this SFR is performed under the corresponding functions in FCS_COP.1/KAT.

FCS_CKM.4 Cryptographic Key Destruction

TSS

The evaluator shall examine the TSS to ensure it lists all relevant keys and keying material (describing the source of the data, all memory types in which the data is stored (covering storage both during and outside of a session, and both plaintext and non-plaintext forms of the data)), all relevant destruction situations (including the point in time at which the destruction occurs; e.g. factory reset or device wipe function, change of authorization data, change of DEK, completion of use of an intermediate key) and the destruction method used in each case. The evaluator shall confirm that the description of the data and storage locations is consistent with the functions carried out by the TOE (e.g. that all keys in the key chain are accounted for). (Where keys are stored encrypted or wrapped under another key then this may need to be explained in order to allow the evaluator to confirm the consistency of the description of keys with the TOE functions).

The evaluator shall check that the TSS identifies any configurations or circumstances that may not conform to

the key destruction requirement (see further discussion in the AGD section below). Note that reference may be made to the AGD for description of the detail of such cases where destruction may be prevented or delayed.

Where the ST specifies the use of "a value that does not contain any sensitive data" to overwrite keys, the evaluator shall examine the TSS to ensure that it describes how that pattern is obtained and used, and that this justifies the claim that the pattern does not contain any sensitive data.

Guidance

The evaluator shall check that the guidance documentation for the TOE requires users to ensure that the TOE remains under the user's control while a session is active.

A TOE may be subject to situations that could prevent or delay data destruction in some cases. The evaluator shall check that the guidance documentation identifies configurations or circumstances that may not strictly conform to the key destruction requirement, and that this description is consistent with the relevant parts of the TSS (and KMD). The evaluator shall check that the guidance documentation provides guidance on situations where key destruction may be delayed at the physical layer, identifying any additional mitigation actions for the user (e.g. there might be some operation the user can invoke, or the user might be advised to retain control of the device for some particular time to maximise the probability that garbage collection will have occurred).

For example, when the TOE does not have full access to the physical memory, it is possible that the storage may implement wear-levelling and garbage collection. This may result in additional copies of the data that are logically inaccessible but persist physically. Where available, the TOE might then describe use of the TRIM command and garbage collection to destroy these persistent copies upon their deletion (this would be explained in TSS and guidance documentation).

Where TRIM is used then the TSS or guidance documentation is also expected to describe how the keys are stored such that they are not inaccessible to TRIM, (e.g. they would need not to be contained in a file less than 982 bytes which would be completely contained in the master file table.

KMD

The evaluator shall examine the KMD to verify that it identifies and describes the interfaces that are used to service commands to read/write memory. The evaluator shall examine the interface description for each different media type to ensure that the interface supports the selections made by the ST author.

45 The evaluator shall examine the KMD to ensure that all keys and keying material identified in the TSS and KMD have been accounted for.

46 Note that where selections include 'destruction of reference to the key directly followed by a request for garbage collection' (for volatile memory) then the evaluator shall examine the KMD to ensure that it explains the nature of the destruction of the reference, the request for garbage collection, and of the garbage collection process itself.

Tests

The following tests require the developer to provide access to a test platform that provides the evaluator with tools that are typically not found on factory products.

The evaluator shall perform the following tests:

- **Test 1:** Applied to each key or keying material held as plaintext in volatile memory and subject to destruction by overwrite by the TOE (whether or not the plaintext value is subsequently encrypted for storage in volatile or non-volatile memory).

  The evaluator shall:
  1. Record the value of the key or keying material.
  2. Cause the TOE to dump the SDO/SDE memory of the TOE into a binary file.
  3. Search the content of the binary file created in Step #2 to locate all instances of the known key value from Step #1.

     Note that the primary purpose of Step #3 is to demonstrate that appropriate search commands are being used for Steps #8 and #9.
  4. Cause the TOE to perform normal cryptographic processing with the key from Step #1.
  5. Cause the TOE to destroy the key.
  6. Cause the TOE to stop execution but not exit.
  7. Cause the TOE to dump the SDO/SDE memory of the TOE into a binary file.
  8. Search the content of the binary file created in Step #7 for instances of the known key value from Step #1.
  9. Break the key value from Step #1 into an evaluator-chosen set of fragments and perform a search using each fragment. (Note that the evaluator shall first confirm with the developer how the key is normally stored, in order to choose fragment sizes that are the same or smaller than any fragmentation of the data that may be implemented by the TOE. The endianness or byte-order should also be taken into account in the search.)

  Steps #1-8 ensure that the complete key does not exist anywhere in volatile memory. If a copy is found, then the test fails.

  Step #9 ensures that partial key fragments do not remain in memory. If the evaluator finds a 32-or-greater-consecutive-bit fragment, then fail immediately. Otherwise, there is a chance that it is not within the context of a key (e.g., some random bits that happen to match). If this is the case the test should be repeated with a different key in Step #1. If a fragment is also found in this repeated run then the test fails unless the developer provides a reasonable explanation for the collision, then the evaluator may give a pass on this test.

- **Test 2:** Applied to each key and keying material held in non-volatile memory and subject to destruction by overwrite by the TOE.
    1. Record the value of the key or keying material.
    2. Cause the TOE to perform normal cryptographic processing with the key from Step #1.
    3. Search the non-volatile memory the key was stored in for instances of the known key value from Step #1.

        Note that the primary purpose of Step #3 is to demonstrate that appropriate search commands are being used for Steps #5 and #6.
    4. Cause the TOE to clear the key.
    5. Search the non-volatile memory in which the key was stored for instances of the known key value from Step #1. If a copy is found, then the test fails.
    6. Break the key value from Step #1 into an evaluator-chosen set of fragments and perform a search using each fragment. (Note that the evaluator shall first confirm with the developer how the key is normally stored, in order to choose fragment sizes that are the same or smaller than any fragmentation of the data that may be implemented by the TOE. The endianness or byte-order should also be taken into account in the search).

    Step #6 ensures that partial key fragments do not remain in non-volatile memory. If the evaluator finds a 32-or-greater-consecutive-bit fragment, then fail immediately. Otherwise, there is a chance that it is not within the context of a key (e.g., some random bits that happen to match). If this is the case the test should be repeated with a different key in Step #1. If a fragment is also found in this repeated run then the test fails unless the developer provides a reasonable explanation for the collision, then the evaluator may give a pass on this test.
- **Test 3:** Applied to each key and keying material held in non-volatile memory and subject to destruction by overwrite by the TOE.
    1. Record memory of the key or keying material.
    2. Cause the TOE to perform normal cryptographic processing with the key from Step #1.
    3. Cause the TOE to clear the key. Record the value to be used for the overwrite of the key.
    4. Examine the memory from Step #1 to ensure the appropriate pattern (recorded in Step #3) is used.

    The test succeeds if correct pattern is found in the memory location. If the pattern is not found, then the test fails.

FCS_CKM_EXT.4 Cryptographic Key and Key Material Destruction Timing

TSS
The evaluator shall verify the TSS provides a high-level description of what it means for keys and key material to be no longer needed and when this data should be expected to be destroyed.

Guidance
There are no guidance evaluation activities for this component.

KMD
The evaluator shall verify that the KMD includes a description of the areas where keys and key material reside and when this data is no longer needed.

The evaluator shall verify that the KMD includes a key lifecycle that includes a description where key materials reside, how the key materials are used, how it is determined that keys and key material are no longer needed, and how the data is destroyed once it is no longer needed. The evaluator shall also verify that all key destruction operations are performed in a manner specified by FCS_CKM.4.

Tests
There are no test evaluation activities for this component

FCS_CKM_EXT.5 Cryptographic Key Derivation

TSS
The evaluator shall check that the TSS includes a description of the key derivation functions and shall check that this uses a key derivation algorithm and key sizes according to the specification selected in the ST out of the table as provided in the cPP table per row. The evaluator shall confirm that the TSS supports the selected methods.

If KeyDrv5 is selected, the evaluator shall verify that the TSS shows that the total length of the concatenated keys used as input to the KDF is greater than or equal to the length of the output from the KDF.

[conditional] If key combination is used to form a KEK, the evaluator shall verify that the TSS describes the method of combination and that this method is either an XOR, a KDF, or encryption.

[conditional] If a KDF is used to form a KEK, the evaluator shall ensure that the TSS includes a description of the key derivation function and shall verify the key derivation uses an approved derivation mode and key expansion algorithm according to SP 800-108.

[conditional] If key concatenation is used to derive KEKs (KeyDrv5), the evaluator shall ensure the TSS includes a description of the randomness extraction step, including the following:

- The description must include how an approved untruncated MAC function is being used for the randomness extraction step and the evaluator must verify the TSS describes that the output length (in bits) of the MAC function is at least as large as the targeted security strength (in bits) of the parameter set employed by the key establishment scheme (see Tables 1-3 of SP 800-56C).
- The description must include how the MAC function being used for the randomness extraction step is related to the PRF used in the key expansion and verify the TSS description includes the correct MAC function:
    - If an HMAC-hash is used in the randomness extraction step, then the same HMAC-hash (with the same hash function hash) is used as the PRF in the key expansion step.

- If an AES-CMAC (with key length 128, 192, or 256 bits) is used in the randomness extraction step, then AES-CMAC with a 128-bit key is used as the PRF in the key expansion step.
- The description must include the lengths of the salt values being used in the randomness extraction step and the evaluator shall verify the TSS description includes correct salt lengths:
    - If an HMAC-hash is being used as the MAC, the salt length can be any value up to the maximum bit length permitted for input to the hash function hash.
    - If an AES-CMAC is being used as the MAC, the salt length shall be the same length as the AES key (i.e. 128, 192, or 256 bits).

Guidance
The evaluator shall verify that the AGD guidance instructs the administrator how to configure the TOE to use the selected key types for all uses identified in the ST.

KMD
The evaluator shall examine the KMD to ensure that:

- The KMD describes the complete key derivation chain and the description must be consistent with the description in the TSS. For all key derivations the TOE must use a method as described in the cPP table. There should be no uncertainty about how a key is derived from another in the chain.
- The length of the key derivation key is defined by the PRF. The evaluator should check whether the key derivation key length is consistent with the length provided by the selected PRF.
- If a key is used as an input to several KDFs, each invocation must use a distinct context string. If the output of a KDF execution is used for multiple cryptographic keys, those keys must be disjoint segments of the output.

Tests
The following tests require the developer to provide access to a test platform that provides the evaluator with tools that are typically not found on factory products.

The evaluator shall perform one or more of the following tests to verify the correctness of the key derivation function, depending on the specific functions that are supported:

Preconditions for testing:

- Specification of input parameter to the key derivation function to be tested
- Specification of further required input parameters
- Access to derived keys

The following table maps the data fields in the tests below to the notations used in SP 800-108 and SP 800-56C

Data Fields	Notations	
	SP 800-108	SP 800-56C
Pseudorandom function	PRF	PRF
Counter length	r	r
Length of output of PRF	r	r
Length of derived keying material	L	L
Length of input values	I_length	I_length
Pseudorandom input values I	K1 (key derivation key)	Z (shared secret)
Pseudorandom salt values		S
Randomness extraction MAC	n/a	MAC

The below tests are derived from Key Derivation using Pseudorandom Functions (SP 800-108) Validation System (KBKDFVS), Updated 4 January 2016, Section 6.2, from the National Institute of Standards and Technology.

KeyDrv1: Counter Mode Tests:
The evaluator shall determine the following characteristics of the key derivation function:

- One or more pseudorandom functions that are supported by the implementation (PRF).
- One or more of the values {8, 16, 24, 32} that equal the length of the binary representation of the counter (r).
- The length (in bits) of the output of the PRF (h).
- Minimum and maximum values for the length (in bits) of the derived keying material (L). These values can be equal if only one value of L is supported. These must be evenly divisible by h.
- Up to two values of L that are NOT evenly divisible by h.
- Location of the counter relative to fixed input data: before, after, or in the middle.
    - Counter before fixed input data: fixed input data string length (in bytes), fixed input data string value.
    - Counter after fixed input data: fixed input data string length (in bytes), fixed input data string value.
    - Counter in the middle of fixed input data: length of data before counter (in bytes), length of data after counter (in bytes), value of string input before counter, value of string input after counter.
- The length (I_length) of the input values I.

For each supported combination of I_length, MAC, salt, PRF, counter location, value of r, and value of L, the evaluator shall generate 10 test vectors that include pseudorandom input values I, and pseudorandom salt values. If there is only one value of L that is evenly divisible by h, the evaluator shall generate 20 test vectors for it. For each test vector, the evaluator shall supply this data to the TOE in order to produce the keying material output.

The results from each test may either be obtained by the evaluator directly or by supplying the inputs to the implementer and receiving the results in response. To determine correctness, the evaluator shall compare the resulting values to those obtained by submitting the same inputs to a known good implementation.

KeyDrv2: Feedback Mode Tests:

The evaluator shall determine the following characteristics of the key derivation function:

- One or more pseudorandom functions that are supported by the implementation (PRF).
- The length (in bits) of the output of the PRF (h).
- Minimum and maximum values for the length (in bits) of the derived keying material (L). These values can be equal if only one value of L is supported. These must be evenly divisible by h.
- Up to two values of L that are NOT evenly divisible by h.
- Whether or not zero-length IVs are supported.
- Whether or not a counter is used, and if so:
  - One or more of the values {8, 16, 24, 32} that equal the length of the binary representation of the counter (r).
  - Location of the counter relative to fixed input data: before, after, or in the middle.
    - Counter before fixed input data: fixed input data string length (in bytes), fixed input data string value.
    - Counter after fixed input data: fixed input data string length (in bytes), fixed input data string value.
    - Counter in the middle of fixed input data: length of data before counter (in bytes), length of data after counter (in bytes), value of string input before counter, value of string input after counter.
- The length (I_length) of the input values L.

For each supported combination of I_length, MAC, salt, PRF, counter location (if a counter is used), value of r (if a counter is used), and value of L, the evaluator shall generate 10 test vectors that include pseudorandom input values I and pseudorandom salt values. If the KDF supports zero-length IVs, five of these test vectors will be accompanied by pseudorandom IVs and the other five will use zerolength IVs. If zero-length IVs are not supported, each test vector will be accompanied by an pseudorandom IV. If there is only one value of L that is evenly divisible by h, the evaluator shall generate 20 test vectors for it.

For each test vector, the evaluator shall supply this data to the TOE in order to produce the keying material output. The results from each test may either be obtained by the evaluator directly or by supplying the inputs to the implementer and receiving the results in response. To determine correctness, the evaluator shall compare the resulting values to those obtained by submitting the same inputs to a known good implementation.

KeyDrv3: Double Pipeline Iteration Mode Tests:

The evaluator shall determine the following characteristics of the key derivation function:

- One or more pseudorandom functions that are supported by the implementation (PRF).
- The length (in bits) of the output of the PRF (h).
- Minimum and maximum values for the length (in bits) of the derived keying material (L). These values can be equal if only one value of L is supported. These must be evenly divisible by h.
- Up to two values of L that are NOT evenly divisible by h.
- Whether or not a counter is used, and if so:
  - One or more of the values {8, 16, 24, 32} that equal the length of the binary representation of the counter (r).
  - Location of the counter relative to fixed input data: before, after, or in the middle.
    - Counter before fixed input data: fixed input data string length (in bytes), fixed input data string value.
    - Counter after fixed input data: fixed input data string length (in bytes), fixed input data string value.
    - Counter in the middle of fixed input data: length of data before counter (in bytes), length of data after counter (in bytes), value of string input before counter, value of string input after counter.
- The length (I_length) of the input values I.

For each supported combination of I_length, MAC, salt, PRF, counter location (if a counter is used), value of r (if a counter is used), and value of L, the evaluator shall generate 10 test vectors that include pseudorandom input values I, and pseudorandom salt values. If there is only one value of L that is evenly divisible by h, the evaluator shall generate 20 test vectors for it.

For each test vector, the evaluator shall supply this data to the TOE in order to produce the keying material output. The results from each test may either be obtained by the evaluator directly or by supplying the inputs to the implementer and receiving the results in response. To determine correctness, the evaluator shall compare the resulting values to those obtained by submitting the same inputs to a known good implementation.

KeyDrv4: Intermediate Keys Method

If the selected algorithm is a hash then the testing of the hash primitive is the only required Evaluation Activity. If the selected algorithm is XOR then no separate primitive testing is necessary.

KeyDrv5: Concatenated Keys Method

The evaluator should confirm that the combined length of the concatenated keys should be at least as long as the keysize of the selected methods. There are no other tests other than for the methods selected for this row performed for KeyDrv1, KeyDrv2, and KeyDrv3.

KeyDrv6: Two Keys Method

The evaluator should confirm that the combined length of the two keys should be at least as long as the keysize of the selected methods. There are no other tests other than for the methods selected for this row from FCD_COP.1/SK.

KeyDrv7: Shared Secret, Salt, Output Length, Fixed Information Method

For each supported selection of PRF, length of shared secret (Z) [selection: 128, 256] bits, length of salt (S) [selection: length of input block of PRF, one-half length of input block of PRF, 0] bits, output length (L) [selection: 128, 256] bits, and length of fixed information (FixedInfo) [selection: length of on input block of PRF, onehalf length of input block of PRF, 0] bits, the evaluator shall generate 10 test vectors that include pseudorandom input values for Z, salt values (for non-zero lengths, otherwise, omit) and fixed information (for non-zero lengths, otherwise, omit).

For each test vector, the evaluator shall supply this data to the TOE in order to produce the keying material output. The results from each test may either be obtained by the evaluator directly or by supplying the inputs to the implementer and receiving the results in response. To determine correctness, the evaluator shall compare the resulting values to those obtained by submitting the same inputs to a known good implementation.

KeyDrv8: Shared Secret, Salt, IV, Output Length, Fixed Information Method

For each supported selection of PRF, length of shared secret (Z), length of salt, length of initialization vector (IV), output length (L), and length of fixed information (FixedInfo), the evaluator shall generate 10 test vectors that include pseudorandom input values for Z, salt values (for non-zero lengths, otherwise, omit), IV (for non-zero lengths, otherwise, use a vector of length equal to length of input block of PRF and fill with zeros), and fixed information (for non-zero lengths, otherwise, omit).

For each test vector, the evaluator shall supply this data to the TOE in order to produce the keying material output. The results from each test may either be obtained by the evaluator directly or by supplying the inputs to the implementer and receiving the results in response. To determine correctness, the evaluator shall compare the resulting values to those obtained by submitting the same inputs to a known good implementation.

FCS_COP.1/Hash Cryptographic Operation (Hashing)

TSS
The evaluator shall check that the association of the hash function with other TSF cryptographic functions (for example, the digital signature verification function) is documented in the TSS. The evaluator shall also check that the TSS identifies whether the implementation is bit-oriented or byte-oriented.
Guidance
The evaluator checks the AGD documents to determine that any configuration that is required to configure the required hash sizes is present. The evaluator also checks the AGD documents to confirm that the instructions for establishing the evaluated configuration use only those hash algorithms selected in the ST.
KMD
There are no KMD evaluation activities for this component.
Tests
The following tests require the developer to provide access to a test platform that provides the evaluator with tools that are typically not found on factory products.

SHA-1 and SHA-2 Tests

The tests below are derived from the "The Secure Hash Algorithm Validation System (SHAVS), Updated: May 21, 2014" from the National Institute of Standards and Technology.

The TSF hashing functions can be implemented with one of two orientations. The first is a byte-oriented implementation: this hashes messages that are an integral number of bytes in length (i.e., the length (in bits) of the message to be hashed is divisible by 8). The second is a bit-oriented implementation: this hashes messages of arbitrary length. Separate tests for each orientation are given below.

The evaluator shall perform all of the following tests for each hash algorithm and orientation implemented by the TSF and used to satisfy the requirements of this PP. The evaluator shall compare digest values produced by a known-good SHA implementation against those generated by running the same values through the TSF.

Short Messages Test, Bit-oriented Implementation

The evaluators devise an input set consisting of m+1 messages, where m is the block length of the hash algorithm in bits (see SHA Properties Table). The length of the messages ranges sequentially from 0 to m bits. The message text shall be pseudorandomly generated. The evaluators compute the message digest for each of the messages and ensure that the correct result is produced when the messages are provided to the TSF.

Short Messages Test, Byte-oriented Implementation

The evaluators devise an input set consisting of m/8+1 messages, where m is the block length of the hash algorithm in bits (see SHA Properties Table). The length of the messages ranges sequentially from 0 to m/8 bytes, with each message being an integral number of bytes. The message text shall be pseudo-randomly generated. The evaluators compute the message digest for each of the messages and ensure that the correct result is produced when the messages are provided to the TSF.

Selected Long Messages Test, Bit-oriented Implementation

The evaluators devise an input set consisting of m messages, where m is the block length of the hash

algorithm in bits (see SHA Properties Table). The length of the ith message is m + 99*i, where 1 ≤ i ≤ m. The message text shall be pseudorandomly generated. The evaluators compute the message digest for each of the messages and ensure that the correct result is produced when the messages are provided to the TSF.

Selected Long Messages Test, Byte-oriented Implementation

The evaluators devise an input set consisting of m/8 messages, where m is the block length of the hash algorithm in bits (see SHA Properties Table). The length of the ith message is m + 8*99*i, where 1 ≤ i ≤ m/8. The message text shall be pseudorandomly generated. The evaluators compute the message digest for each of the messages and ensure that the correct result is produced when the messages are provided to the TSF.

Pseudo-randomly Generated Messages Test

The evaluators randomly generate a seed that is n bits long, where n is the length of the message digest produced by the hash function to be tested. The evaluators then formulate a set of 100 messages and associated digests by following the algorithm provided in Figure 1 of SHAVS, section 6.4. The evaluators then ensure that the correct result is produced when the messages are provided to the TSF.

SHA-3 Tests

The tests below are derived from the The Secure Hash Algorithm-3 Validation System (SHA3VS), Updated: April 7, 2016, from the National Institute of Standards and Technology.

For each SHA-3-XXX implementation, XXX represents d, the digest length in bits. The capacity, c, is equal to 2d bits. The rate is equal to 1600-c bits.

65 The TSF hashing functions can be implemented with one of two orientations. The first is a bit-oriented mode that hashes messages of arbitrary length. The second is a byte-oriented mode that hashes messages that are an integral number of bytes in length (i.e., the length (in bits) of the message to be hashed is divisible by 8). Separate tests for each orientation are given below.

The evaluator shall perform all of the following tests for each hash algorithm and orientation implemented by the TSF and used to satisfy the requirements of this PP. The evaluator shall compare digest values produced by a known-good SHA-3 implementation against those generated by running the same values through the TSF.

Short Messages Test, Bit-oriented Mode

The evaluators devise an input set consisting of rate+1 short messages. The length of the messages ranges sequentially from 0 to rate bits. The message text shall be pseudo-randomly generated. The evaluators compute the message digest for each of the messages and ensure that the correct result is produced when the messages are provided to the TSF. The message of length 0 is omitted if the TOE does not support zero-length messages.

Short Messages Test, Byte-oriented Mode

The evaluators devise an input set consisting of rate/8+1 short messages. The length of the messages ranges sequentially from 0 to rate/8 bytes, with each message being an integral number of bytes. The message text shall be pseudo-randomly generated. The evaluators compute the message digest for each of the messages and ensure that the correct result is produced when the messages are provided to the TSF. The message of length 0 is omitted if the TOE does not support zero-length messages.

Selected Long Messages Test, Bit-oriented Mode

The evaluators devise an input set consisting of 100 long messages ranging in size from rate+(rate+1) to rate+(100*(rate+1)), incrementing by rate+1. (For example, SHA-3-256 has a rate of 1088 bits. Therefore, 100 messages will be generated with lengths 2177, 3266, ..., 109988 bits.) The message text shall be pseudo-randomly generated. The evaluators compute the message digest for each of the messages and ensure that the correct result is produced when the messages are provided to the TSF.

Selected Long Messages Test, Byte-oriented Mode

The evaluators devise an input set consisting of 100 messages ranging in size from (rate+(rate+8)) to (rate+100*(rate+8)), incrementing by rate+8. (For example, SHA-3-256 has a rate of 1088 bits. Therefore 100 messages will be generated of lengths 2184, 3280, 4376, ..., 110688 bits.) The message text shall be pseudorandomly generated. The evaluators compute the message digest for each of the messages and ensure that the correct result is produced when the messages are provided to the TSF.

Pseudo-randomly Generated Messages Monte Carlo) Test, Byte-oriented Mode

The evaluators supply a seed of d bits (where d is the length of the message digest produced by the hash function to be tested. This seed is used by a pseudorandom function to generate 100,000 message digests. One hundred of the digests (every 1000th digest) are recorded as checkpoints. The TOE then uses the same procedure to generate the same 100,000 message digests and 100 checkpoint values. The evaluators then compare the results generated ensure that the correct result is produced when the messages are generated by the TSF.

FCS_COP.1/HMAC Cryptographic Operation (Keyed Hash)

TSS
The evaluator shall examine the TSS to ensure that it specifies the following values used by the HMAC and KMAC functions: output MAC length used.
Guidance

There are no guidance evaluation activities for this component.

KMD

There are no KMD evaluation activities for this component.

Tests

The following test requires the developer to provide access to a test platform that provides the evaluator with tools that are typically not found on factory products.

This test is derived from The Keyed-Hash Message Authentication Code Validation System (HMACVS), updated 6 May 2016.

The evaluator shall provide 15 sets of messages and keys for each selected hash algorithm and hash length/key size/MAC size combination. The evaluator shall have the TSF generate HMAC or KMAC tags for these sets of test data. The evaluator shall verify that the resulting HMAC or KMAC tags match the results from submitting the same inputs to a known-good implementation of the HMAC or KMAC function, having the same characteristics.

FCS_COP.1/KAT Cryptographic Operation (Key Agreement/Transport)

TSS

The evaluator shall ensure that the selected RSA and ECDH key agreement/transport schemes correspond to the key generation schemes selected in FCS_CKM.1/AK, and the key establishment schemes selected in FCS_CKM.2 If the ST selects DH, the TSS shall describe how the implementation meets the relevant sections of RFC 3526 (Section 3-7) and RFC 7919 (Appendices A.1-A.5). If the ST selects ECIES, the TSS shall describe the key sizes and algorithms (e.g. elliptic curve point multiplication, ECDH with either NIST or Brainpool curves, AES in a mode permitted by FCS_COP.1/SKC, a SHA-2 hash algorithm permitted by FCS_COP.1/Hash, and a MAC algorithm permitted by FCS_COP.1/HMAC) that are supported for the ECIES implementation.

The evaluator shall ensure that, for each key agreement/transport scheme, the size of the derived keying material is at least the same as the intended strength of the key agreement/transport scheme, and where feasible this should be twice the intended security strength of the key agreement/transport scheme.

Table 2 of NIST SP 800-57 identifies the key strengths for the different algorithms that can be used for the various key agreement/transport schemes.

Guidance

There are no guidance evaluation activities for this component.

KMD

There are no KMD evaluation activities for this component.

Tests

The following tests require the developer to provide access to a test platform that provides the evaluator with tools that are typically not found on factory products.

The evaluator shall verify the implementation of the key generation routines of the supported schemes using the following tests:

If ECDH-NIST or ECDH-BPC is claimed:

SP800-56A Key Agreement Schemes

The evaluator shall verify a TOE's implementation of SP800-56A key agreement schemes using the following Function and Validity tests. These validation tests for each key agreement scheme verify that a TOE has implemented the components of the key agreement scheme according to the specifications in the Recommendation. These components include the calculation of the DLC primitives (the shared secret value Z) and the calculation of the derived keying material (DKM) via the Key Derivation Function (KDF). If key confirmation is supported, the evaluator shall also verify that the components of key confirmation have been implemented correctly, using the test procedures described below. This includes the parsing of the DKM, the generation of MACdata and the calculation of MACtag.

Function Test

The Function test verifies the ability of the TOE to implement the key agreement schemes correctly. To conduct this test the evaluator shall generate or obtain test vectors from a known good implementation of the TOE supported schemes. For each supported key agreement scheme-key agreement role combination, KDF type, and, if supported, key confirmation role-key confirmation type combination, the tester shall generate 10 sets of test vectors. The data set consists of one set of domain parameter values (FFC) or the NIST approved curve (ECC) per 10 sets of public keys. These keys are static, ephemeral or both depending on the scheme being tested.

The evaluator shall obtain the DKM, the corresponding TOE's public keys (static or ephemeral), the MAC tags, and any inputs used in the KDF, such as the Other Information field OI and TOE id fields.

If the TOE does not use a KDF defined in SP 800-56A, the evaluator shall obtain only the public keys and the hashed value of the shared secret.

The evaluator shall verify the correctness of the TSF's implementation of a given scheme by using a known good implementation to calculate the shared secret value, derive the keying material DKM, and compare hashes or MAC tags generated from these values.

If key confirmation is supported, the TSF shall perform the above for each implemented approved MAC algorithm.

Validity Test

The Validity test verifies the ability of the TOE to recognize another party's valid and invalid key agreement results with or without key confirmation. To conduct this test, the evaluator shall obtain a list of the supporting cryptographic functions included in the SP800-56A key agreement implementation to determine which errors the TOE should be able to recognize. The evaluator generates a set of 24 (FFC) or 30 (ECC) test vectors consisting of data sets including domain parameter values or NIST approved curves, the evaluator's public keys, the TOE's public/private key pairs, MACTag, and any inputs used in the KDF, such as the other info and TOE id fields.

The evaluator shall inject an error in some of the test vectors to test that the TOE recognizes invalid key agreement results caused by the following fields being incorrect: the shared secret value Z, the DKM, the other information field OI, the data to be MACed, or the generated MACTag. If the TOE contains the full or partial (only ECC) public key validation, The evaluator shall also individually inject errors in both parties' static public keys, both parties' ephemeral public keys and the TOE's static private key to assure the TOE detects errors in the public key validation function or the partial key validation function (in ECC only). At least two of the test vectors shall remain unmodified and therefore should result in valid key agreement results (they should pass).

The TOE shall use these modified test vectors to emulate the key agreement scheme using the corresponding parameters. The evaluator shall compare the TOE's results with the results using a known good implementation verifying that the TOE detects these errors.

If KAS1, KAS2, KTS-OAEP, or RSAES-PKCS1-v1_5 is claimed:

SP800-56B and PKCS#1 Key Establishment Schemes

If the TOE acts as a sender, the following evaluation activity shall be performed to ensure the proper operation of every TOE supported combination of RSA-based key establishment scheme:

To conduct this test the evaluator shall generate or obtain test vectors from a known good implementation of the TOE supported schemes. For each combination of supported key establishment scheme and its options (with or without key confirmation if supported, for each supported key confirmation MAC function if key confirmation is supported, and for each supported mask generation function if KTS-OAEP is supported), the tester shall generate 10 sets of test vectors. Each test vector shall include the RSA public key, the plaintext keying material, any additional input parameters if applicable, the MacKey and MacTag if key confirmation is incorporated, and the outputted ciphertext. For each test vector, the evaluator shall perform a key establishment encryption operation on the TOE with the same inputs (in cases where key confirmation is incorporated, the test shall use the MacKey from the test vector instead of the randomly generated MacKey used in normal operation) and ensure that the outputted ciphertext is equivalent to the ciphertext in the test vector.

If the TOE acts as a receiver, the following evaluation activities shall be performed to ensure the proper operation of every TOE supported combination of RSA-based key establishment scheme:

To conduct this test the evaluator shall generate or obtain test vectors from a known good implementation of the TOE supported schemes. For each combination of supported key establishment scheme and its options (with our without key confirmation if supported, for each supported key confirmation MAC function if key confirmation is supported, and for each supported mask generation function if KTSOAEP is supported), the tester shall generate 10 sets of test vectors. Each test vector shall include the RSA private key, the plaintext keying material (KeyData), any additional input parameters if applicable, the MacTag in cases where key confirmation is incorporated, and the outputted ciphertext. For each test vector, the evaluator shall perform the key establishment decryption operation on the TOE and ensure that the outputted plaintext keying material (KeyData) is equivalent to the plain text keying material in the test vector. In cases where key confirmation is incorporated, the evaluator shall perform the key confirmation steps and ensure that the outputted MacTag is equivalent to the MacTag in the test vector.

The evaluator shall ensure that the TSS describes how the TOE handles decryption errors. In accordance with NIST Special Publication 800-56B, the TOE must not reveal the particular error that occurred, either through the contents of any outputted or logged error message or through timing variations. If KTS-OAEP is supported, the evaluator shall create separate contrived ciphertext values that trigger each of the three decryption error checks described in NIST Special Publication 800-56B section 7.2.2.3, ensure that each decryption attempt results in an error, and ensure that any outputted or logged error message is identical for each.

DH:

The evaluator shall verify the correctness of each TSF implementation of each supported Diffie-Hellman group by comparison with a known good implementation.

Curve25519:

The evaluator shall verify a TOE's implementation of the key agreement scheme using the following Function and Validity tests. These validation tests for each key agreement scheme verify that a TOE has implemented the components of the key agreement scheme according to the specification. These components include the calculation of the shared secret K and the hash of K.

Function Test

The Function test verifies the ability of the TOE to implement the key agreement schemes correctly. To conduct this test the evaluator shall generate or obtain test vectors from a known good implementation of the TOE supported schemes. For each supported key agreement role and hash function combination, the tester shall generate 10 sets of public keys. These keys are static, ephemeral or both depending on the scheme being tested.

The evaluator shall obtain the shared secret value K, and the hash of K. The evaluator shall verify the correctness of the TSF's implementation of a given scheme by using a known good implementation to calculate the shared secret value K and compare the hash generated from this value.

Validity Test

The Validity test verifies the ability of the TOE to recognize another party's valid and invalid key agreement results. To conduct this test, the evaluator generates a set of 30 test vectors consisting of data sets including the evaluator's public keys and the TOE's public/private key pairs.

The evaluator shall inject an error in some of the test vectors to test that the TOE recognizes invalid key agreement results caused by the following fields being incorrect: the shared secret value K or the hash of K. At least two of the test vectors shall remain unmodified and therefore should result in valid key agreement results (they should pass).

The TOE shall use these modified test vectors to emulate the key agreement scheme using the corresponding parameters. The evaluator shall compare the TOE's results with the results using a known good implementation verifying that the TOE detects these errors.

ECIES:

The evaluator shall verify the correctness of each TSF implementation of each supported use of ECIES by comparison with a known good implementation.

FCS_COP.1/KeyEnc Cryptographic Operation (Key Encryption)

TSS
The evaluator shall examine the TSS to ensure that it identifies whether the implementation of this cryptographic operation for key encryption (including key lengths and modes) is an implementation that is tested in FCS_COP.1/SKC.

The evaluator shall check that the TSS includes a description of the key wrap functions and shall check that this uses a key wrap algorithm and key sizes according to the specification selected in the ST out of the table as provided in the cPP table.

Guidance
The evaluator checks the AGD documents to confirm that the instructions for establishing the evaluated configuration use only those key wrap functions selected in the ST. If multiple key access modes are supported, the evaluator shall examine the guidance documentation to determine that the method of choosing a specific mode/key size by the end user is described.

KMD
The evaluator shall examine the KMD to ensure that it describes when the key wrapping occurs, that the KMD description is consistent with the description in the TSS, and that for all keys that are wrapped the TOE uses a method as described in the cPP table. No uncertainty should be left over which is the wrapping key and the key to be wrapped and where the wrapping key potentially comes from i.e. is derived from.

If "AES-GCM" or "AES-CCM" is used the evaluator shall examine the KMD to ensure that it describes how the IV is generated and that the same IV is never reused to encrypt different plaintext pairs under the same key. Moreover in the case of GCM, he must ensure that, at each invocation of GCM, the length of the plaintext is at most $(2^{32})-2$ blocks.

Tests
Refer to FCS_COP.1/SKC for the required testing for each symmetric key wrapping method selected from the table and to FCS_COP.1/KAT for the required testing for each asymmetric key wrapping method selected from the table. Each distinct implementation shall be tested separately.

If the implementation of the key encryption operation is the same implementation tested under FCS_COP.1/SKC or FCS_COP.1/KAT, and it has been tested with the same key lengths and modes, then no further testing is required. If key encryption uses a different implementation, (where "different implementation" includes the use of different key lengths or modes), then the evaluator shall additionally test the key encryption implementation using the corresponding tests specified for FCS_COP.1/SKC or FCS_COP.1/KAT.

FCS_COP.1/PBKDF Cryptographic Operation (Password-Based Key Derivation Functions)

TSS
The evaluator shall review the TSS to verify that it contains a description of the PBKDF. The evaluator shall also confirm the ST supports the selected hash function itself. The evaluator shall confirm that the TSS contains a description of how the TOE ensures that the output of the PBKDF is at least the same length as that specified in FCS_CKM.1/SK and for the KeyDrv4, KeyDrv5, or KeyDrv6 in FCS_CKM_EXT.5.

If the ST performs additional conditioning, whitening, or manipulation of the password or passphrase before applying the PBKDF, or to the output of the PBKDF, the evaluator shall ensure that the TSS describes the actions and provides assurance that the TSF does not negatively impact the entropy of the PBKDF output.

If any manipulation of the key is performed in forming the submask that will be used to form the KEK, that process shall be described in the TSS.
Guidance
There are no AGD evaluation activities for this component.
KMD
There are no KMD evaluation activities for this component.

Tests

No explicit testing of the formation of the submask from the input password is required.

For the NIST SP 800-132-based conditioning of the passphrase, the required evaluation activities will be performed when doing the evaluation activities for the appropriate requirements (FCS_COP.1/HMAC).

The evaluator shall verify that the iteration count for PBKDFs performed by the TOE comply with NIST SP 800-132 by ensuring that the TSS contains a description of the estimated time required to derive key material from passwords and how the TOE increases the computation time for password-based key derivation (including but not limited to increasing the iteration count).

FCS_COP.1/SigGen Cryptographic Operation (Signature Generation)

TSS

The evaluator shall examine the TSS to ensure that all signature generation functions use the approved algorithms and key sizes.

Guidance

There are no AGD evaluation activities for this component.

KMD

There are no KMD evaluation activities for this component.

Tests

The following tests require the developer to provide access to a test platform that provides the evaluator with tools that are typically not found on factory products.

Each section below contains tests the evaluators must perform for each selected digital signature scheme. Based on the assignments and selections in the requirement, the evaluators choose the specific activities that correspond to those selections.

The following tests require the developer to provide access to a test platform that provides the evaluator with tools that are not found on the TOE in its evaluated configuration.

If SigGen1: RSASSA-PKCS1-v1_5or SigGen4: RSASSA-PSS is claimed:

The below test is derived from The 186-4 RSA Validation System (RSA2VS). Updated 8 July 2014, Section 6.3, from the National Institute of Standards and Technology.

To test the implementation of RSA signature generation the evaluator uses the system under test to generate signatures for 10 messages for each combination of modulus size and SHA algorithm. The evaluator then uses a known-good implementation and the associated public keys to verify the signatures.

If SigGen2: Digital Signature Scheme 2 (DSS2) or SigGen3: Digital Signature Scheme 3 (DSS3):

To test the implementation of DSS2/3 signature generation the evaluator uses the system under test to generate signatures for 10 messages for each combination of SHA algorithm, hash size and key size. The evaluator them uses a known-good implementation and the associated public keys to verify the signatures.

If SigGen5: ECDSA is claimed:

The below test is derived from The FIPS 186-4 Elliptic Curve Digital Signature Algorithm Validation System (ECDSA2VS). Updated 18 March 2014, Section 6.4, from the National Institute of Standards and Technology.

To test the implementation of ECDSA signature generation the evaluator uses the system under test to generate signatures for 10 messages for each combination of curve, SHA algorithm, hash size, and key size. The evaluator then uses a known-good implementation and the associated public keys to verify the signatures.

FCS_COP.1/SigVer Cryptographic Operation (Signature Verification)

TSS

The evaluator shall check the TSS to ensure that it describes the overall flow of the signature verification. This should at least include identification of the format and general location (e.g., "firmware on the hard drive device" rather than "memory location 0x00007A4B") of the data to be used in verifying the digital signature; how the data received from the operational environment are brought onto the device; and any processing that is performed that is not part of the digital signature algorithm (for instance, checking of certificate revocation lists).

Guidance

There are no AGD evaluation activities for this component.

KMD

There are no KMD evaluation activities for this component.

Tests

The following tests require the developer to provide access to a test platform that provides the evaluator with tools that are typically not found on factory products.

Each section below contains tests the evaluators must perform for each selected digital signature scheme. Based on the assignments and selections in the requirement, the evaluators choose the specific activities that correspond to those selections.

The following tests require the developer to provide access to a test platform that provides the evaluator with tools that are not found on the TOE in its evaluated configuration.

SigVer1: RSASSA-PKCS1-v1_5 and SigVer4: RSASSA-PSS

These tests are derived from The 186-4 RSA Validation System (RSA2VS), updated 8 Jul 2014, Section 6.4.

The FIPS 186-4 RSA Signature Verification Test tests the ability of the TSF to recognize valid and invalid signatures. The evaluator shall provide a modulus and three associated key pairs (d, e) for each combination of selected SHA algorithm, modulus size and hash size. Each private key d is used to sign six pseudorandom messages each of 1024 bits. For five of the six messages, the public key (e), message, IR format, padding, or signature is altered so that signature verification should fail. The test passes only if all the signatures made using unaltered parameters result in successful signature verification, and all the signatures made using altered parameters result in unsuccessful signature verification.

SigVer5: ECDSA on NIST and Brainpool Curves

These tests are derived from The FIPS 186-4 Elliptic Curve Digital Signature Algorithm Validation System (ECDSA2VS), updated 18 Mar 2014, Section 6.5.

The FIPS 186-4 ECC Signature Verification Test tests the ability of the TSF to recognize valid and invalid signatures. The evaluator shall provide a modulus and associated key pair (x, y) for each combination of selected curve, SHA algorithm, modulus size, and hash size. Each private key (x) is used to sign 15 pseudorandom messages of 1024 bits. For eight of the fifteen messages, the message, IR format, padding, or signature is altered so that signature verification should fail. The test passes only if all the signatures made using unaltered parameters result in successful signature verification, and all the signatures made using altered parameters result in unsuccessful signature verification.

SigVer2: Digital Signature Scheme 2

The following or equivalent steps shall be taken to test the TSF.

For each supported modulus size, underlying hash algorithm, and length of the trailer field (1- or 2-byte), the evaluator shall generate NT sets of recoverable message (M1), non-recoverable message (M2), salt, public key and signature (Σ).

1. N_T shall be greater than or equal to 20.
2. The length of salts shall be selected from its supported length range of salt. The typical length of salt is equal to the output block length of underlying hash algorithm (see 9.2.2 of ISO/IEC 9796-2:2010).
3. The length of recoverable messages should be selected by considering modulus size, output block length of underlying hash algorithm, and length of salt (L_S). As described in Annex D of ISO/IEC 9796-2:2010, it is desirable to maximise the length of recoverable message. The following table shows the maximum bit-length of recoverable message that is divisible by 512, for some combinations of modulus size, underlying hash algorithm, and length of salt. None that 2-byte trailer field is assumed in calculating the maximum length of recoverable message

Maximum length of recoverable message divisible by 512 (bits)	Modulus size (bits)	Underlying hash algorithm (bits)	Length of salt L_S (bits)
1536	2048	SHA-256	128
1024			256
1024		SHA-512	128
1024			256
512			512
2560	3072	SHA-256	128
2048			256
2048		SHA-512	128
2048			256
1536			512

4. The length of non-recoverable messages should be selected by considering the underlying hash algorithm and usages. If the TSF is used for verifying the authenticity of software/firmware updates, the length of non-recoverable messages should be selected greater than or equal to 2048-bit. With this length range, it means that the underlying hash algorithm is also tested for two or more input blocks.
5. The evaluator shall select approximately one half of N_T sets and shall alter one of the values (non-recoverable message, public key exponent or signature) in the sets. In altering public key exponent, the evaluator shall alter the public key exponent while keeping the exponent odd. In altering signatures, the following ways should be considered:
   a. Altering a signature just by replacing a bit in the bit-string representation of the signature
   b. Altering a signature so that the trailer in the message representative cannot be interpreted. This can be achieved by following ways:
      - Setting the rightmost four bits of the message representative to the values other than '1100'.
      - In the case when 1-byte trailer is used, setting the rightmost byte of the message representative to the values other than '0xbc', while keeping the rightmost four bits to '1100'.
      - In the case when 2-byte trailer is used, setting the rightmost byte of the message representative to the values other than '0xcc', while keeping the rightmost four bits to '1100'.
   c. In the case when 2-byte trailer is used, altering a signature so that the hash algorithm identifier in the trailer (i.e. the left most byte of the trailer) does not correspond to hash algorithms identified in the SFR. The hash algorithm identifiers are 0x34 for SHA-256 (see Clause 10 of ISO/IEC 10118-3:2018), and 0x35 for SHA-512 (see Clause 11 of ISO/IEC 10118-3:2018).
   d. Let L_S be the length of salt, altering a signature so that the intermediate bit string D in the message representative is set to all zeroes except for the rightmost L_S bits of D.

e. (non-conformant signature length) Altering a signature so that the length of signature Σ is changed to modulus size and the most significant bit of signature Σ is set equal to '1'.

f. (non-conformant signature) Altering a signature so that the integer converted from signature Σ is greater than modulus n.

The evaluator shall supply the NT sets to the TSF and obtain in response a set of NT Verification-Success or Verification-Fail values. When the VerificationSuccess is obtained, the evaluator shall also obtain recovered message (M 1*).

The evaluator shall verify that Verification-Success results correspond to the unaltered sets and Verification-Fail results correspond to the altered sets.

For each recovered message, the evaluator shall compare the recovered message (M1*) with the corresponding recoverable message (M 1) in the unaltered sets.

The test passes only if all the signatures made using unaltered sets result in Verification-Success, each recovered message (M 1*) is equal to corresponding M 1 in the unaltered sets, and all the signatures made using altered sets result in Verification-Fail.

SigVer3: Digital Signature Scheme 3

The evaluator shall perform the test described in SigVer2: Digital Signature Scheme 2 while using a fixed salt for NT sets.

FCS_COP.1/SKC Cryptographic Operation (Symmetric Key Cryptography)

TSS
The evaluator shall check that the TSS includes a description of encryption functions used for symmetric key encryption. The evaluator should check that this description of the selected encryption function includes the key sizes and modes of operations as specified in the cPP table 9 "Supported Methods for Symmetric Key Cryptography Operation."

The evaluator shall check that the TSS describes the means by which the TOE satisfies constraints on algorithm parameters included in the selections made for 'cryptographic algorithm' and 'list of standards'.

Guidance
If the product supports multiple modes, the evaluator shall examine the vendor's documentation to determine that the method of choosing a specific mode/key size by the end user is described.

KMD
The evaluator shall examine the KMD to ensure that the points at which symmetric key encryption and decryption occurs are described, and that the complete data path for symmetric key encryption is described. The evaluator checks that this description is consistent with the relevant parts of the TSS.

Assessment of the complete data path for symmetric key encryption includes confirming that the KMD describes the data flow from the device's host interface to the device's non-volatile memory storing the data, and gives information enabling the user data datapath to be distinguished from those situations in which data bypasses the data encryption engine (e.g. read-write operations to an unencrypted Master Boot Record area). The evaluator shall ensure that the documentation of the data path is detailed enough that it thoroughly describes the parts of the TOE that the data passes through (e.g. different memory types, processors and co-processors), its encryption state (i.e. encrypted or unencrypted) in each part, and any places where the data is stored. For example, any caching or buffering of the data should be identified and distinguished from the final destination in non-volatile memory (the latter represents the location from which the host will expect to retrieve the data in future).

If support for AES-CTR is claimed and the counter value source is internal to the TOE, the evaluator shall verify that the KMD describes the internal counter mechanism used to ensure that it provides unique counter block values.

Tests
The following tests require the developer to provide access to a test platform that provides the evaluator with tools that are typically not found on factory products.

The following tests are conditional based upon the selections made in the SFR. The evaluator shall perform the following test or witness respective tests executed by the developer. The tests must be executed on a platform that is as close as practically possible to the operational platform (but which may be instrumented in terms of, for example, use of a debug mode). Where the test is not carried out on the TOE itself, the test platform shall be identified and the differences between test environment and TOE execution environment shall be described.

Preconditions for testing:

- Specification of keys as input parameter to the function to be tested
- specification of required input parameters such as modes
- Specification of user data (plaintext)
- Tapping of encrypted user data (ciphertext) directly in the non-volatile memory

AES-CBC:
For the AES-CBC tests described below, the plaintext, ciphertext, and IV values shall consist of 128-bit blocks. To determine correctness, the evaluator shall compare the resulting values to those obtained by submitting the same inputs to a known-good implementation.

These tests are intended to be equivalent to those described in NIST's AES Algorithm Validation Suite (AESAVS) (http://csrc.nist.gov/groups/STM/cavp/documents/aes/AESAVS.pdf). It is not recommended that

evaluators use values obtained from static sources such as the example NIST's AES Known Answer Test Values from the AESAVS document, or use values not generated expressly to exercise the AES-CBC implementation.

AES-CBC Known Answer Tests

<u>KAT-1 (GFSBox):</u> To test the encrypt functionality of AES-CBC, the evaluator shall supply a set of five different plaintext values for each selected key size and obtain the ciphertext value that results from AES-CBC encryption of the given plaintext using a key value of all zeros and an IV of all zeros.

To test the decrypt functionality of AES-CBC, the evaluator shall supply a set of five different ciphertext values for each selected key size and obtain the plaintext value that results from AES-CBC decryption of the given ciphertext using a key value of all zeros and an IV of all zeros.

<u>KAT-2 (KeySBox):</u> To test the encrypt functionality of AES-CBC, the evaluator shall supply a set of five different key values for each selected key size and obtain the ciphertext value that results from AES-CBC encryption of an all-zeros plaintext using the given key value and an IV of all zeros.

To test the decrypt functionality of AES-CBC, the evaluator shall supply a set of five different key values for each selected key size and obtain the plaintext that results from AES-CBC decryption of an all-zeros ciphertext using the given key and an IV of all zeros.

<u>KAT-3 (Variable Key):</u> To test the encrypt functionality of AES-CBC, the evaluator shall supply a set of keys for each selected key size (as described below) and obtain the ciphertext value that results from AES encryption of an all-zeros plaintext using each key and an IV of all zeros.

Key i in each set shall have the leftmost i bits set to ones and the remaining bits to zeros, for values of i from 1 to the key size. The keys and corresponding ciphertext are listed in AESAVS, Appendix E.

To test the decrypt functionality of AES-CBC, the evaluator shall use the same keys as above to decrypt the ciphertext results from above. Each decryption should result in an all-zeros plaintext.

<u>KAT-4 (Variable Text):</u> To test the encrypt functionality of AES-CBC, for each selected key size, the evaluator shall supply a set of 128-bit plaintext values (as described below) and obtain the ciphertext values that result from AES-CBC encryption of each plaintext value using a key of each size and IV consisting of all zeros.

Plaintext value i shall have the leftmost i bits set to ones and the remaining bits set to zeros, for values of i from 1 to 128. The plaintext values are listed in AESAVS, Appendix D.

To test the decrypt functionality of AES-CBC, for each selected key size, use the plaintext values from above as ciphertext input, and AES-CBC decrypt each ciphertext value using key of each size consisting of all zeros and an IV of all zeros.

AES-CBC Multi-Block Message Test

The evaluator shall test the encrypt functionality by encrypting nine i-block messages for each selected key size, for $2 \leq i \leq 10$. For each test, the evaluator shall supply a key, an IV, and a plaintext message of length i blocks, and encrypt the message using AES-CBC. The resulting ciphertext values shall be compared to the results of encrypting the plaintext messages using a known good implementation.

The evaluator shall test the decrypt functionality by decrypting nine i-block messages for each selected key size, for $2 \leq i \leq 10$. For each test, the evaluator shall supply a key, an IV, and a ciphertext message of length i blocks, and decrypt the message using AES-CBC. The resulting plaintext values shall be compared to the results of decrypting the ciphertext messages using a known good implementation.

AES-CBC Monte Carlo Tests

The evaluator shall test the encrypt functionality for each selected key size using 100 3-tuples of pseudo-random values for plaintext, IVs, and keys.

The evaluator shall supply a single 3-tuple of pseudo-random values for each selected key size. This 3-tuple of plaintext, IV, and key is provided as input to the below algorithm to generate the remaining 99 3-tuples, and to run each 3-tuple through 1000 iterations of AES-CBC encryption.

```
# Input: PT, IV, Key
Key[0] = Key
IV[0] = IV
PT[0] = PT
for i = 0 to 99 {
        Output Key[i], IV[i], PT[0]
        for j = 0 to 999 {
                if (j == 0) {
                        CT[j] = AES-CBC-Encrypt(Key[i], IV[i], PT[j])
                        PT[j+1] = IV[i]
                } else {
                        CT[j] = AES-CBC-Encrypt(Key[i], PT[j])
                        PT[j+1] = CT[j-1]
                }
        }
        Output CT[j]
        If (KeySize == 128) Key[i+1] = Key[i] xor CT[j]
        If (KeySize == 192) Key[i+1] = Key[i] xor (last 64 bits of CT[j-1] || C
        If (KeySize == 256) Key[i+1] = Key[i] xor ((CT[j-1] | CT[j]))
        IV[i+1] = CT[j]
        PT[0] = CT[j-1]
}
```

The ciphertext computed in the 1000th iteration (CT[999]) is the result for each of the 100 3-tuples for each selected key size. This result shall be compared to the result of running 1000 iterations with the same values using a known good implementation.

The evaluator shall test the decrypt functionality using the same test as above, exchanging CT and PT, and replacing AES-CBC-Encrypt with AES-CBC-Decrypt.

AES-CCM:

These tests are intended to be equivalent to those described in the NIST document, "The CCM Validation System (CCMVS)," updated 9 Jan 2012, found at http://csrc.nist.gov/groups/STM/cavp/documents/mac/CCMVS.pdf.

It is not recommended that evaluators use values obtained from static sources such as http://csrc.nist.gov/groups/STM/cavp/documents/mac/ccmtestvectors.zip or use values not generated expressly to exercise the AES-CCM implementation.

The evaluator shall test the generation-encryption and decryption-verification functionality of AES-CCM for the following input parameter and tag lengths:

- **Keys:** All supported and selected key sizes (e.g., 128, 192, or 256 bits).
- **Associated Data:** Two or three values for associated data length: The minimum (≥ 0 bytes) and maximum (≤ 32 bytes) supported associated data lengths, and 2^16 (65536) bytes, if supported.
- **Payload:** Two values for payload length: The minimum (≥ 0 bytes) and maximum (≤ 32 bytes) supported payload lengths.
- **Nonces:** All supported nonce lengths (e.g., 8, 9, 10, 11, 12, 13) in bytes.
- **Tag:** All supported tag lengths (e.g., 4, 6, 8, 10, 12, 14, 16) in bytes.

The testing for CCM consists of five tests. To determine correctness in each of the below tests, the evaluator shall compare the ciphertext with the result of encryption of the same inputs with a known good implementation.

Variable Associated Data Test: For each supported key size and associated data length, and any supported payload length, nonce length, and tag length, the evaluator shall supply one key value, one nonce value, and 10 pairs of associated data and payload values, and obtain the resulting ciphertext.

Variable Payload Text: For each supported key size and payload length, and any supported associated data length, nonce length, and tag length, the evaluator shall supply one key value, one nonce value, and 10 pairs of associated data and payload values, and obtain the resulting ciphertext.

Variable Nonce Test: For each supported key size and nonce length, and any supported associated data length, payload length, and tag length, the evaluator shall supply one key value, one nonce value, and 10 pairs of associated data and payload values, and obtain the resulting ciphertext.

Variable Tag Test: For each supported key size and tag length, and any supported associated data length, payload length, and nonce length, the evaluator shall supply one key value, one nonce value, and 10 pairs of associated data and payload values, and obtain the resulting ciphertext.

Decryption-Verification Process Test: To test the decryption-verification functionality of AES-CCM, for each combination of supported associated data length, payload length, nonce length, and tag length, the evaluator shall supply a key value and 15 sets of input plus ciphertext, and obtain the decrypted payload. Ten of the 15 input sets supplied should fail verification and five should pass.

AES-GCM: These tests are intended to be equivalent to those described in the NIST document, "The Galois/Counter Mode (GCM) and GMAC Validation System (GCMVS) with the Addition of XPN Validation Testing," rev. 15 Jun 2016, section 6.2, found at http://csrc.nist.gov/groups/STM/cavp/documents/mac/gcmvs.pdf.

It is not recommended that evaluators use values obtained from static sources such as http://csrc.nist.gov/groups/STM/cavp/documents/mac/gcmtestvectors.zip, or use values not generated expressly to exercise the AES-GCM implementation.

The evaluator shall test the authenticated encryption functionality of AES-GCM by supplying 15 sets of Key, Plaintext, AAD, IV, and Tag data for every combination of the following parameters as selected in the ST and supported by the implementation under test:

- **Key size in bits:** Each selected and supported key size (e.g., 128, 192, or 256 bits).
- **Plaintext length in bits:** Up to four values for plaintext length: Two values that are non-zero integer multiples of 128, if supported. And two values that are non-multiples of 128, if supported.
- **AAD length in bits:** Up to five values for AAD length: Zero-length, if supported. Two values that are non-zero integer multiples of 128, if supported. And two values that are integer non-multiples of 128, if supported.
- **IV length in bits:** Up to three values for IV length: 96 bits. Minimum and maximum supported lengths, if different.
- **MAC length in bits:** Each supported length (e.g., 128, 120, 112, 104, 96).

To determine correctness, the evaluator shall compare the resulting values to those obtained by submitting the same inputs to a known-good implementation.

The evaluator shall test the authenticated decrypt functionality of AES-GCM by supplying 15 Ciphertext-Tag pairs for every combination of the above parameters, replacing Plaintext length with Ciphertext length. For

each parameter combination the evaluator shall introduce an error into either the Ciphertext or the Tag such that approximately half of the cases are correct and half the cases contain errors. To determine correctness, the evaluator shall compare the resulting pass/fail status and Plaintext values to the results obtained by submitting the same inputs to a known-good implementation.

AES-CTR:

For the AES-CTR tests described below, the plaintext and ciphertext values shall consist of 128-bit blocks. To determine correctness, the evaluator shall compare the resulting values to those obtained by submitting the same inputs to a known-good implementation.

These tests are intended to be equivalent to those described in NIST's AES Algorithm Validation Suite (AESAVS) (http://csrc.nist.gov/groups/STM/cavp/documents/aes/AESAVS.pdf). It is not recommended that evaluators use values obtained from static sources such as the example NIST's AES Known Answer Test Values from the AESAVS document, or use values not generated expressly to exercise the AES-CTR implementation.

AES-CTR Known Answer Tests

KAT-1 (GFSBox): To test the encrypt functionality of AES-CTR, the evaluator shall supply a set of five different plaintext values for each selected key size and obtain the ciphertext value that results from AES-CTR encryption of the given plaintext using a key value of all zeros.

To test the decrypt functionality of AES-CTR, the evaluator shall supply a set of five different ciphertext values for each selected key size and obtain the plaintext value that results from AES-CTR decryption of the given ciphertext using a key value of all zeros.

KAT-2 (KeySBox): To test the encrypt functionality of AES-CTR, the evaluator shall supply a set of five different key values for each selected key size and obtain the ciphertext value that results from AES-CTR encryption of an all-zeros plaintext using the given key value.

To test the decrypt functionality of AES-CTR, the evaluator shall supply a set of five different key values for each selected key size and obtain the plaintext that results from AES-CTR decryption of an all-zeros ciphertext using the given key.

KAT-3 (Variable Key): To test the encrypt functionality of AES-CTR, the evaluator shall supply a set of keys for each selected key size (as described below) and obtain the ciphertext value that results from AES encryption of an all-zeros plaintext using each key.

Key i in each set shall have the leftmost i bits set to ones and the remaining bits to zeros, for values of i from 1 to the key size. The keys and corresponding ciphertext are listed in AESAVS, Appendix E.

To test the decrypt functionality of AES-CTR, the evaluator shall use the same keys as above to decrypt the ciphertext results from above. Each decryption should result in an all-zeros plaintext.

KAT-4 (Variable Text): To test the encrypt functionality of AES-CTR, for each selected key size, the evaluator shall supply a set of 128-bit plaintext values (as described below) and obtain the ciphertext values that result from AES-CTR encryption of each plaintext value using a key of each size.

Plaintext value i shall have the leftmost i bits set to ones and the remaining bits set to zeros, for values of i from 1 to 128. The plaintext values are listed in AESAVS, Appendix D.

To test the decrypt functionality of AES-CTR, for each selected key size, use the plaintext values from above as ciphertext input, and AES-CTR decrypt each ciphertext value using key of each size consisting of all zeros.

AES-CTR Multi-Block Message Test

The evaluator shall test the encrypt functionality by encrypting nine i-block messages for each selected key size, for $2 \leq i \leq 10$. For each test, the evaluator shall supply a key and a plaintext message of length i blocks, and encrypt the message using AES-CTR. The resulting ciphertext values shall be compared to the results of encrypting the plaintext messages using a known good implementation.

The evaluator shall test the decrypt functionality by decrypting nine i-block messages for each selected key size, for $2 \leq i \leq 10$. For each test, the evaluator shall supply a key and a ciphertext message of length i blocks, and decrypt the message using AES-CTR. The resulting plaintext values shall be compared to the results of decrypting the ciphertext messages using a known good implementation.

AES-CTR Monte Carlo Tests

The evaluator shall test the encrypt functionality for each selected key size using 100 2-tuples of pseudo-random values for plaintext and keys.

The evaluator shall supply a single 2-tuple of pseudo-random values for each selected key size. This 2-tuple of plaintext and key is provided as input to the below algorithm to generate the remaining 99 2-tuples, and to run each 2-tuple through 1000 iterations of AES-CTR encryption.

```
# Input: PT, Key
Key[0] = Key
PT[0] = PT
for i = 0 to 99 {
        Output Key[i], PT[0]
        for j = 0 to 999 {
                CT[j] = AES-CTR-Encrypt(Key[i], PT[j])
                PT[j+1] = CT[j]
        }
```

```
                                         Output CT[j]
                                         If (KeySize == 128) Key[i+1] = Key[i] xor CT[j]
                                         If (KeySize == 192) Key[i+1] = Key[i] xor (last 64 bits of CT[j-1] || C
                                         If (KeySize == 256) Key[i+1] = Key[i] xor ((CT[j-1] | CT[j])
                                         PT[0] = CT[j]
                        }
```

The ciphertext computed in the 1000th iteration (CT[999]) is the result for each of the 100 2-tuples for each selected key size. This result shall be compared to the result of running 1000 iterations with the same values using a known good implementation.

The evaluator shall test the decrypt functionality using the same test as above, exchanging CT and PT, and replacing AES-CTR-Encrypt with AES-CTR-Decrypt. 198 Note additional design considerations for this mode are addressed in the KMD requirements.

XTS-AES: These tests are intended to be equivalent to those described in the NIST document, "The XTS-AES Validation System (XTSVS)," updated 5 Sept 2013, found at
http://csrc.nist.gov/groups/STM/cavp/documents/aes/XTSVS.pdf

It is not recommended that evaluators use values obtained from static sources such as the XTS-AES test vectors at http://csrc.nist.gov/groups/STM/cavp/documents/aes/XTSTestVectors.zip or use values not generated expressly to exercise the XTS-AES implementation.

The evaluator shall generate test values as follows:

For each supported key size (256 bit (for AES-128) and 512 bit (for AES-256) keys), the evaluator shall provide up to five data lengths:

- Two data lengths divisible by the 128-bit block size, If data unit lengths of complete block sizes are supported.
- Two data lengths not divisible by the 128-bit block size, if data unit lengths of partial block sizes are supported.
- The largest data length supported by the implementation, or 2^{16} (65536), whichever is larger.

The evaluator shall specify whether the implementation supports tweak values of 128-bit hexadecimal strings or a data unit sequence numbers, or both.

For each combination of key size and data length, the evaluator shall provide 100 sets of input data and obtain the ciphertext that results from XTS-AES encryption. If both kinds of tweak values are supported then each type of tweak value shall be used in half of every 100 sets of input data, for all combinations of key size and data length. The evaluator shall verify that the resulting ciphertext matches the results from submitting the same inputs to a known-good implementation of XTS-AES.

The evaluator shall test the decrypt functionality of XTS-AES using the same test as for encrypt, replacing plaintext values with ciphertext values and XTS-AES encrypt with XTS- AES decrypt.

The evaluator shall check that the full length keys are created by methods that ensure that the two halves are different and independent.

AES-KWP:

The tests below are derived from "The Key Wrap Validation System (KWVS), Updated: June 20, 2014" from the National Institute of Standards and Technology.

The evaluator shall test the authenticated-encryption functionality of AES-KWP (KWP-AE) using the same test as for AES-KW authenticated-encryption with the following change in the five plaintext lengths:

- Four lengths that are multiples of 8 bits
- The largest supported length less than or equal to 4096 bits.

The evaluator shall test the authenticated-decryption (KWP-AD) functionality of AES-KWP using the same test as for AES-KWP authenticated-encryption, replacing plaintext values with ciphertext values and AES-KWP authenticatedencryption with AES-KWP authenticated-decryption. For the Authenticated Decryption test, 20 out of the 100 trials per plaintext length have ciphertext values that fail authentication.

Additionally, the evaluator shall perform the following negative tests:

Test 1: (invalid plaintext length):

Determine the valid plaintext lengths of the implementation from the TOE specification. Verify that the implementation of KWP-AE in the TOE rejects plaintexts of invalid length by testing plaintext of the following lengths: 1) plaintext with length greater than 64 semi-blocks, 2) plaintext with bit-length not divisible by 8, and 3) plaintext with length 0.

Test 2: (invalid ciphertext length): Determine the valid ciphertext lengths of the implementation from the TOE specification. Verify that the implementation of KWP-AD in the TOE rejects ciphertexts of invalid length by testing ciphertext of the following lengths: 1) ciphertext with length greater than 65 semi-blocks, 2) ciphertext with bit-length not divisible by 64, 3) ciphertext with length 0, and 4) ciphertext with length of one semi-block.

Test 3: (invalid ICV2): Test that the implementation detects invalid ICV2 values by encrypting any plaintext value four times using a different value for ICV2 each time as follows: Start with a base ICV2 of 0xA65959A6. For each of the four tests change a different byte of ICV2 to a different value, so that each of the four bytes is changed once. Verify that the implementation of KWP-AD in the TOE outputs FAIL for each test.

Test 4: (invalid padding length): Generate one ciphertext using algorithm KWP-AE with substring [len(P)/8]32 of S replaced by each of the following 32-bit values, where len(P) is the length of P in bits and [ ]32 denotes the representation of an integer in 32 bits:

- [0]32
- [len(P)/8-8]32
- [len(P)/8+8]32
- [513]32.

Verify that the implementation of KWP-AD in the TOE outputs FAIL on those inputs.

Test 5: (invalid padding bits):

If the implementation supports plaintext of length not a multiple of 64-bits, then

```
for each PAD length [1..7]
        for each byte in PAD set a zero PAD value;
                replace current byte by a non-zero value and use the resulting plaintex
                        input to algorithm KWP-AE to generate ciphertexts;
                verify that the implementation of KWP-AD in the TOE outputs FAIL on
                        this input.
```

AES-KW:

The tests below are derived from "The Key Wrap Validation System (KWVS), Updated: June 20, 2014" from the National Institute of Standards and Technology.

The evaluator shall test the authenticated-encryption functionality of AES-KW for each combination of the following input parameters:

- Supported key lengths selected in the ST (e.g. 128 bits, 256 bits)
- Five plaintext lengths:
    - Two lengths that are non-zero multiples of 128 bits (two semi-block lengths)
    - Two lengths that are odd multiples of the semi-block length (64 bits)
    - The largest supported plaintext length less than or equal to 4096 bits.

For each set of the above parameters the evaluator shall generate a set of 100 key and plaintext pairs and obtain the ciphertext that results from AES-KW authenticated encryption. To determine correctness, the evaluator shall compare the results with those obtained from the AES-KW authenticated-encryption function of a known good implementation.

The evaluator shall test the authenticated-decryption functionality of AES-KW using the same test as for authenticated-encryption, replacing plaintext values with ciphertext values and AES-KW authenticated-encryption (KW-AE) with AES-KW authenticated-decryption (KW-AD). For the authenticated-decryption test, 20 out of the 100 trials per plaintext length must have ciphertext values that are not authentic; that is, they fail authentication.

Additionally, the evaluator shall perform the following negative tests:

Test 1 (invalid plaintext length):

Determine the valid plaintext lengths of the implementation from the TOE specification. Verify that the implementation of KW-AE in the TOE rejects plaintexts of invalid length by testing plaintext of the following lengths: 1) plaintext length greater than 64 semi-blocks, 2) plaintext bit-length not divisible by 64, 3) plaintext with length 0, and 4) plaintext with one semi-block.

Test 2 (invalid ciphertext length):

Determine the valid ciphertext lengths of the implementation from the TOE specification. Verify that the implementation of KW-AD in the TOE rejects ciphertexts of invalid length by testing ciphertext of the following lengths: 1) ciphertext with length greater than 65 semi-blocks, 2) ciphertext with bit-length not divisible by 64, 3) ciphertext with length 0, 4) ciphertext with length of one semiblock, and 5) ciphertext with length of two semi-blocks.

Test 3 (invalid ICV1):

222 Test that the implementation detects invalid ICV1 values by encrypting any plaintext value eight times using a different value for ICV1 each time as follows: Start with a base ICV1 of 0xA6A6A6A6A6A6A6A6. For each of the eight tests change a different byte to a different value, so that each of the eight bytes is changed once. Verify that the implementation of KW-AD in the TOE outputs FAIL for each test.

CAM-CBC:

To test the encrypt and decrypt functionality of Camellia in CBC mode, the evaluator shall perform the tests as specified in 10.2.1.2 of ISO/IEC 18367:2016.

CAM-CCM:

To test the encrypt functionality of Camellia in CCM mode, the evaluator shall perform the tests as specified in 10.6.1.1 of ISO/IEC 18367:2016.

To test the decrypt functionality of Camellia in CCM mode, the evaluator shall perform the tests as specified in 10.6.1.2 of ISO/IEC 18367:2016.

As a prerequisite for these tests, the evaluator shall perform the test for encrypt functionality of Camellia in ECB mode as specified in 10.2.1.2 of ISO/IEC 18367:2016.

CAM-GCM:

To test the encrypt functionality of Camellia in GCM, the evaluator shall perform the tests as specified in 10.6.1.1 of ISO/IEC 18367:2016.

To test the decrypt functionality of Camellia in GCM, the evaluator shall perform the tests as specified in 10.6.1.2 of ISO/IEC 18367:2016.

As a prerequisite for these tests, the evaluator shall perform the test for encrypt functionality of Camellia in ECB mode as specified in 10.2.1.2 of ISO/IEC 18367:2016.

XTS-CAM:

These tests are intended to be equivalent to those described in the IPA document, ATR-01-B, "Specifications of Cryptographic Algorithm Implementation Testing — Symmetric-Key Cryptography", found at https://www.ipa.go.jp/security/jcmvp/jcmvp_e/documents/atr/atr01b_en.pdf.

The evaluator shall generate test values as follows:

For each supported key size (256 bit (for Camellia-128) and 512 bit (for Camellia256) keys), the evaluator shall provide up to five data lengths:

- Two data lengths divisible by the 128-bit block size, If data unit lengths of complete block sizes are supported.
- Two data lengths not divisible by the 128-bit block size, if data unit lengths of partial block sizes are supported.
- The largest data length supported by the implementation, or 2^{16} (65536), whichever is larger.

The evaluator shall specify whether the implementation supports tweak values of 128-bit hexadecimal strings or a data unit sequence numbers, or both.

For each combination of key size and data length, the evaluator shall provide 100 sets of input data and obtain the ciphertext that results from XTS-Camellia encryption. If both kinds of tweak values are supported, 50 of each 100 sets of input data shall use each type of tweak value. The resulting ciphertext shall be compared to the results of a known-good implementation.

As a prerequisite for this test, the evaluator shall perform the test for encrypt functionality of Camellia in ECB mode as specified in 10.2.1.2 of ISO/IEC 18367:2016.

The evaluator shall test the decrypt functionality of XTS-Camellia using the same test as for encrypt, replacing plaintext values with ciphertext values and XTSCamellia encrypt with XTS- Camellia decrypt.

As a prerequisite for this test, the evaluator shall perform the test for decrypt functionality of Camellia in ECB mode as specified in 10.2.1.2 of ISO/IEC 18367:2016.

FCS_RBG_EXT.1 Random Bit Generation

TSS
The evaluator shall examine the TSS to determine that it specifies the DRBG type, identifies the entropy sources seeding the DRBG, and state the assumed or calculated min-entropy supplied either separately by each source or the min-entropy contained in the combined seed value.

In addition to the materials below, documentation shall be produced—and the evaluator shall perform the activities—in accordance with Appendix D of [DSCcPP].

Guidance
There are no AGD evaluation activities for this component.
KMD
There are no KMD evaluation activities for this component.
Tests
The following tests require the developer to provide access to a test platform that provides the evaluator with tools that are typically not found on factory products.

The evaluator shall perform 15 trials for the RNG implementation. If the RNG is configurable, the evaluator shall perform 15 trials for each configuration.

If the RNG has prediction resistance enabled, each trial consists of (1) instantiate DRBG, (2) generate the first block of random bits (3) generate a second block of random bits (4) uninstantiate. The evaluator verifies that the second block of random bits is the expected value. The evaluator shall generate eight input values for each trial. The first is a count (0 – 14). The next three are entropy input, nonce, and personalization string for the instantiate operation. The next two are additional input and entropy input for the first call to generate. The final two are additional input and entropy input for the second call to generate. These values are randomly generated. "generate one block of random bits" means to generate random bits with number of returned bits equal to the Output Block Length (as defined in NIST SP800-90A).

If the RNG does not have prediction resistance, each trial consists of (1) instantiate DRBG, (2) generate the first block of random bits (3) reseed, (4) generate a second block of random bits (5) uninstantiate. The evaluator verifies that the second block of random bits is the expected value. The evaluator shall generate eight input values for each trial. The first is a count (0 – 14). The next three are entropy input, nonce, and personalization string for the instantiate operation. The fifth value is additional input to the first call to

generate. The sixth and seventh are additional input and entropy input to the call to reseed. The final value is additional input to the second generate call.

The following paragraphs contain more information on some of the input values to be generated/selected by the evaluator.

- **Entropy input:** the length of the entropy input value must equal the seed length.
- **Nonce:** If a nonce is supported (CTR_DRBG with no Derivation Function does not use a nonce), the nonce bit length is one-half the seed length.
- **Personalization string:** The length of the personalization string must be ≤ seed length. If the implementation only supports one personalization string length, then the same length can be used for both values. If more than one string length is support, the evaluator shall use personalization strings of two different lengths. If the implementation does not use a personalization string, no value needs to be supplied.
- **Additional input:** the additional input bit lengths have the same defaults and restrictions as the personalization string lengths.

FCS_SLT_EXT.1 Cryptographic Salt Generation

TSS
The evaluator shall ensure the TSS describes how salts are generated using the RBG.
Guidance
There are no AGD evaluation activities for this component.
KMD
There are no KMD evaluation activities for this component.
Tests
The evaluator shall confirm by testing that the salts obtained in the cryptographic operations that use the salts are of the length specified in FCS_SLT_EXT.1, are obtained from the RBG, and are fresh on each invocation.

Note: in general these tests may be carried out as part of the tests of the relevant cryptographic operations.

FCS_STG_EXT.1 Protected Storage

TSS
The evaluator shall review the TSS to determine that the TOE implements the required protected storage. The evaluator shall ensure that the TSS contains a description of the protected storage mechanism that justifies the selection of mutable hardware-based or software-based.

Guidance
The evaluator shall examine the operational guidance to ensure that it describes the process for generating keys, importing keys, or both, based on what is claimed by the ST. The evaluator shall also examine the operational guidance to ensure that it describes the process for destroying keys that have been imported or generated.

KMD
There are no KMD evaluation activities for this component.
Tests
The evaluator shall test the functionality of each security function as described below. If the TOE supports both import and generation of keys, the evaluator shall repeat the testing as needed to demonstrate that the keys resulting from both operations are treated in the same manner. The devices used with the tooling may need to be non-production devices in order to enable the execution and gathering of evidence.

- **Test 1:** The evaluator shall import or generate keys/secrets of each supported type according to the operational guidance. The evaluator shall write, or the developer shall provide access to, an application that generates a key/secret of each supported type and calls the import functions. The evaluator shall verify that no errors occur during import.
- **Test 2:** The evaluator shall write, or the developer shall provide access to, an application that uses a generated or imported key/secret:
    - For RSA, the secret shall be used to sign data.
    - For ECDSA, the secret shall be used to sign data.
  The evaluator shall repeat this test with the application-imported or applicationgenerated keys/secrets and a different application's imported keys/secrets or generated keys/secrets. The evaluator shall verify that the TOE requires approval before allowing the application to use the key/secret imported or generated by the user or by a different application:
    - The evaluator shall deny the approvals to verify that the application is not able to use the key/secret as described.
    - The evaluator shall repeat the test, allowing the approvals to verify that the application is able to use the key/secret as described.
  If the ST author has selected common application developer, this test is performed by either using applications from different developers or appropriately (according to API documentation) not authorizing sharing.

- **Test 3:** The evaluator shall destroy keys/secrets of each supported type according to the operational guidance. The evaluator shall write, or the developer shall provide access to, an application that destroys an imported or generated key/secret. The evaluator shall repeat this test with the application-imported or applicationgenerated keys/secrets and a different application's imported or generated keys/secrets. The evaluator shall verify that the TOE requires approval before allowing the application to destroy the key/secret imported by the administrator or by a different application:
    - The evaluator shall deny the approvals and verify that the application is still able to use the key/secret as described.

- The evaluator shall repeat the test, allowing the approvals and verifying that the application is no longer able to use the key/secret as described.

If the ST author has selected common application developer, this test is performed by either using applications from different developers or appropriately (according to API documentation) not authorizing sharing.

FCS_STG_EXT.2 Key Storage Encryption

TSS

The evaluator shall review the TSS to determine that the TSS describes the protection of symmetric keys, KEKs, long-term trusted channel key material, and software-based key storage as claimed in FCS_STG_EXT.2.1.

Guidance

There are no AGD evaluation activities for this component.

KMD

There are no KMD evaluation activities for this component

Tests

There are no test evaluation activities for this component.

FCS_STG_EXT.3 Key Integrity Protection

TSS

The evaluator shall examine the TSS and ensure that it contains a description of how the TOE protects the integrity of its keys.

Guidance

There are no AGD evaluation activities for this component.

KMD

There are no KMD evaluation activities for this component.

Tests

There are no test evaluation activities for this component.

2.1.2 User Data Protection

FDP_ACC.1 Subset Access Control

TSS

The evaluator shall confirm that the TSS contain the access control policy implemented by the TOE. I.e., the ST author lists each object and identifies for each object, which operations the TSF permits for each subject (i.e. what can "admins" do vs "users").

Guidance

There are no guidance evaluation activities for this component.

KMD

There are no KMD evaluation activities for this component.

Tests

Testing for FDP_ACF includes testing this component.

FDP_ACF.1 Security Attribute Based Access Control

TSS

The evaluator shall examine the TSS to verify that it describes the policy rules for the Access Control SFP. Specifically, the evaluator should be able to identify, for any arbitrary subject-object-operation pairing, which of the following is true:

a. The subject can always perform the desired operation.
b. The subject can never perform the desired operation, either because they lack sufficient permission or because the TSF includes no interface to support the operation.
c. The subject can only perform the desired operation under certain conditions (which the evaluator shall verify are described in the TSS). For example, "the S.CA subject may only perform the OP.Destroy operation on an OB.SDO object if it was the subject that originally created or imported the SDO."
d. The subject can only perform the desired operation on one or more attributes of the object as opposed to the entire object itself (which the evaluator shall verify are identified in the TSS).
e. Whether the subject can perform the desired operation depends on TSF configuration (which the evaluator shall verify is described in the TSS as part of the evaluation of FMT_SMF.1).
f. Some combination of c, d, and e.

Given that this SFR requires a large number of potential subject-object-operation pairings to be identified, it is not the expectation that the TSS contain an exhaustive list of these pairings. It is possible that large numbers of pairings are addressed by blanket statements of policy rules, such as "the subjects S.DSC and S.CA are never able to perform any operation on the OB.AntiReplay object." For any rules that are not addressed in this manner, the evaluator shall verify the TSS includes sufficient data for the evaluator to determine how the TSF will evaluate the action. This can be presented in the form of a table, flowchart, list, or other manner that the ST author finds suitable.

Note that the DSC developer may not use the same terminology for its subjects, objects, and operations as the PP. If this is the case, the evaluator shall verify that the TSS includes a mapping that unambiguously shows how the vendor's preferred terminology corresponds to what the PP defines.

Guidance

For any access control policy enforcement behavior that is configurable, the evaluator shall ensure that the

operational guidance describes how to perform the configuration, including any restrictions on permissible configurable settings.

KMD
There are no KMD evaluation activities for this component.

Tests
The following testing may require the TOE developer to make a test harness available to the evaluator that allows the evaluator to interface directly with the DSC. Due to the large volume of potential testing that this requires, this test may require the use of an automated script. If a test script is made available, the evaluator shall verify that it includes sufficient detail to validate the claims made in the TSS.

For each subject/object/operation/attribute combination, the evaluator shall attempt to perform the operation or determine that no interface is present to attempt the operation, consistent with the limitations described in the TSS.

For each case where an operation is always permitted or never permitted, both positive and negative testing will be conducted implicitly by attempting the operation with all possible subjects and determining that the intended results occur in each case.

For each case where the operation succeeds or fails based on the target object attribute, the evaluator shall ensure that both positive and negative testing is performed such that only the correct target attributes can be operated upon.

For each case where the operation succeeds or fails based on one or more specific conditions, the evaluator shall ensure that both positive and negative testing is performed such that the presence of the conditions causes the test to succeed while the absence of the conditions causes the test to fail.

For each case where the operation succeeds or fails based on an administratively configured setting, the evaluator shall ensure that both positive and negative testing is performed such that the configuration setting can be shown to affect whether or not the operation succeeds.

FDP_ETC_EXT.2 Propagation of SDOs

TSS
The evaluator shall examine the TSS to ensure that it describes how it protects the SDO references, authorization data, against access from unauthorized entities. If the TSF is selected, then it should describe how it provides confidentiality of the data while it resides outside the TOE.

Guidance
There are no guidance evaluation activities for this component.

KMD
There are no KMD evaluation activities for this component.

Tests
There are no test evaluation activities for this component.

FDP_FRS_EXT.1 Factory Reset

TSS
The evaluator shall examine the TSS to determine that it describes each of the conditions which will lead to a factory reset.

Guidance
The evaluator shall examine the operational guidance to ensure that it describes the ways the administrator can set the conditions to initiate a factory reset.

KMD
There are no KMD evaluation activities for this component.

Tests
The evaluator shall identify all functions that resets the DSC to factory setting. For each function, the evaluator shall identify all methods for authorizing the factory reset. For each function and for each authorization method, the evaluator shall create an SDE or SDO. The evaluator shall then verify the presence of the item just created. The evaluator shall initiate a factory reset using the selected function and authorization method and verify the item no longer exists.

FDP_ITC_EXT.1 Parsing of SDEs

TSS
The evaluator shall confirm the TSS contains descriptions of the supported methods the TSF uses to import SDEs into the TOE. For each import method selected, the TSS shall describe integrity verification schemes employed. The TSS shall also list the ways the TSF generates and binds security attributes to the SDEs.

Guidance
There are no AGD evaluation activities for this component.

KMD
There are no KMD evaluation activities for this component.

Tests
For each supported import method selected in FDP_ITC_EXT.1.1 and for each supported integrity verification method selected in FDP_ITC_EXT.1.2. used by the selected import method, provide one SDE with valid integrity credentials, one with invalid integrity credentials (e.g. hash). The operations with invalid integrity credentials must result in error. The operations with valid integrity credentials must return an SDO with valid security attributes in accordance with FDP_ITC_EXT.1.4.

FDP_ITC_EXT.2 Parsing of SDOs

TSS

The evaluator shall confirm the TSS contains descriptions of the supported methods the TSF uses to import SDOs into the TOE. For each import method selected, the TSS shall describe integrity verification schemes employed. The TSS shall also list the ways the TSF generates and binds security attributes to the SDOs.

Guidance
There are no AGD evaluation activities for this component.

KMD
There are no KMD evaluation activities for this component.

Tests
For each supported import method selected in FDP_ITC_EXT.2.1 and for each supported integrity verification method selected in FDP_ITC_EXT.2.2 used by the selected import method, provide one SDO with valid integrity credentials, one with invalid integrity credentials (e.g. hash). The operations with invalid integrity credentials must result in error. The operations with valid integrity credentials must return an SDO with valid security attributes in accordance with FDP_ITC_EXT.2.3, FDP_ITC_EXT.2.4, and FDP_ITC_EXT.2.5.

FDP_MFW_EXT.1 Mutable/Immutable Firmware

TSS
The evaluator shall examine the TSS and ensure that details of which firmware components are considered mutable and which firmware components are considered immutable, as well as how these firmware components can/cannot be modified or altered, are described. For example, DSC firmware components that are stored in ROM would be considered immutable.

Guidance
If the TOE has mutable firmware, the evaluator shall examine the operational guidance to ensure that it describes how to modify the firmware.

KMD
There are no KMD evaluation activities for this component.

Tests
If the TOE has mutable firmware, the evaluator shall perform the activities described in the operational guidance to modify the firmware.

FDP_RIP.1 Subset Residual Information Protection

TSS
The evaluator shall check to ensure that the TSS describes resource deallocation to the extent that they can determine that no data will be reused when reallocating resources following the destruction of an SDE or SDO. The evaluator shall ensure that this description at a minimum describes how the previous data is destroyed. The evaluator shall also ensure that this destruction method is consistent with FCS_CKM.4.

Guidance
There are no AGD evaluation activities for this component.

KMD
There are no KMD evaluation activities for this component.

Tests
Testing for FCS_CKM.4 is sufficient to address this component.

FDP_SDC_EXT.1 Confidentiality of SDEs

TSS
The evaluator shall examine the TSS to determine that it describes the protection for SDEs and authorization data and the methods of protection (e.g. protected storage, symmetric encryption, key wrapping, etc.).

The evaluator shall also examine the TSS to determine whether the TSF stores this data inside the TOE boundary or in its operational environment. If the TSF stores this data inside the TOE boundary, the evaluator shall ensure that TSF uses one of the listed methods to provide confidentiality. If the data is stored in the TOE's operational environment, the evaluator shall ensure that the TSF uses key wrapping to provide confidentiality.

The evaluator shall examine the TSS to confirm is sufficiently describes each method used to provide confidentiality for SDEs. The evaluator shall also confirm that the TOE supports all encryption methods selected.

Guidance
There are no AGD evaluation activities for this component.

KMD
There are no KMD evaluation activities for this component.

Tests
If the TOE stores SDEs and authorization data inside the TSF, the evaluator shall ensure that external interfaces cannot extract this data in plaintext.

In this case, use the evaluation activities of the FPT_PHP.3 if protected storage is selected, FCS_COP.1/SK if symmetric encryption using… is selected, and FCS_COP.1/KAT if key wrapping using… is selected.

If the TOE stores authentication data inside the operational environment, the evaluator shall ensure that plaintext data is not visible on the interface between the TOE and the operational environment.

FDP_SDI.2 Stored Data Integrity Monitoring and Action

TSS
The evaluator shall confirm that the ST author describes the methods for protecting the integrity of SDOs stored with the TOE, and shall identify the iteration of FCS_COP.1/Hash or FCS_COP.1/HMAC that covers any cryptographic algorithm used. The evaluator shall also confirm that the TSS describes the response upon the

detection of an integrity error.

The evaluator shall confirm that the TSS describes the actions the TSF takes when the integrity verification fails for an SDO, including the circumstances that cause a notification to be sent when this occurs.

The evaluator shall confirm that TSS describes how integrity of SDOs is protected in FMT_MSA.3 during initialization, and how the integrity of SDOs are verified during parsing (import) in FDP_ITC_EXT.

Guidance
The evaluator shall examine the operational guidance to verify that it describes the conditions that cause a notification to be sent when an integrity error is detected, and what the contents of the notification are.

KMD
There are no KMD evaluation activities for this component.

Tests
The tests for FDP_ITC_EXT and FMT_MSA.3 shall suffice for this component.

2.1.3 Identification and Authentication

FIA_AFL_EXT.1 Authorization Failure Handling

TSS
The evaluator shall examine the TSS to determine that it contains a description for how successive unsuccessful authorization attempts are detected and tracked. The evaluator shall examine the TSS to determine that is contains a description of the actions in the event that the authorization attempt threshold is met or exceeded.

The evaluator shall also examine the TSS to determine that it describes how the failed authorization attempt counter is incremented before the authorization is verified.

The evaluator shall also examine the TSS to determine the behaviour that will occur if there are excessive failed authorization attempts, specifically whether future attempts are prevented for a static or configurable amount of time, future attempts are prevented indefinitely, or a factory reset is triggered.

Guidance
The evaluator shall examine the guidance documentation to ensure that instructions for configuring the number of successive unsuccessful authentication attempts and time period (if implemented) are provided, and that the process of unlocking the SDOs is described for each "action" specified (if that option is chosen).

The evaluator shall examine the guidance documentation to confirm that it describes, and identifies the importance of, any actions that are required in order to ensure that access to SDOs can be maintained, unless it is made permanently unavailable due to a factory reset.

KMD
There are no KMD evaluation activities for this component.

Tests
The evaluator shall perform the following tests for each method by which the TSF authorizes access the SDOs (e.g. any passwords entered as part of establishing authorization):

- **Test 1:** The evaluator shall use the operational guidance to configure the number of successive unsuccessful authorization attempts allowed by the TOE (and, if the time period selection in FIA_AFL_EXT.1.3 is included in the ST, then the evaluator shall also use the operational guidance to configure the time period after which access is re-enabled). The evaluator shall test that once the authorization attempts limit is reached, authorization attempts with valid credentials are no longer successful.
- **Test 2:** After reaching the limit for unsuccessful authorization attempts as in Test 1 above, the evaluator shall proceed as follows. If the action selected in FIA_AFL_EXT.1.3 is included in the ST then the evaluator shall confirm by testing that following the operational guidance and performing each action specified in the ST to re-enable access results in successful access. If the time period selection in FIA_AFL_EXT.1.3 is included in the ST, then the evaluator shall wait for just less than the time period configured in Test 1 and show that an authorization attempt using valid credentials does not result in successful access. The evaluator shall then wait until just after the time period configured in Test 1 and show that an authorization attempt using valid credentials results in successful access.
- **Test 3:** [conditional]: If <u>factory reset the TOE wiping out all non-persistent SDOs, as described by FDP_FRS_EXT.2</u> is selected in FIA_AFL_EXT.1.3, the evaluator shall perform the test required by FDP_FRS_EXT.2 with step 5 replaced with "The evaluator shall initiate a factory reset by deliberately meeting or surpassing the threshold for unsuccessful authorization attempts, depending on whether meets or surpasses is selected in FIA_AFL_EXT.1.3."

FIA_SOS.2 TSF Generation of Secrets

TSS
The evaluator shall ensure that the TSS describes for each of the TSF functions listed in FIA_SOS.2.2, if the available key space is configurable, and the size (or range) of the key space employed to generate authorization values.

The evaluator shall ensure that the TSS states that the quality metrics provided is based on the assumption of sufficient entropy being provided in accordance with the information given in [DSCcPP] Annex D.

The evaluator shall ensure that the TSS describes the restrictions implemented in order to restrict consecutive authentication attempts. (Authentication throttling)

The evaluator shall ensure that the TSS describes the mechanism used to generate authorization values and documents the quality metric that the mechanism provides. The information provided in the TSS shall demonstrate that:

a. The probability that a random single authentication attempt will be successful is less than one in 1,000,000; and
b. The probability that random multiple authentication attempts during a one (1) minute period will be successful is less than one in 100,000.

Guidance

The evaluator shall examine the guidance documentation to determine that it describes any configuration necessary to enforce the use of TSF generated authorization values listed in FIA_SOS.2.2.

The evaluator shall ensure that the guidance documentation provides any instructions needed to set parameters affecting the available key spaces.

KMD

There are no KMD evaluation activities for this component.

Tests

The evaluator shall perform the following tests.

- **Test 1:** The evaluator shall compose a set of 50 authorization values that meet the requirements, and 50 authorization values that fail to meet the requirements.
    a. For each authentication value that meets the requirements, the evaluator shall verify that the TOE supports the authentication value.
    b. For each authentication value that does not meet the requirements, the evaluator shall verify that the TOE does not support the authentication value.
    While the evaluator is not required (nor is it feasible) to test all possible compositions of authentication values, the evaluator shall ensure that the key space identified in the TSS is valid.
- **Test 2:** For each TSF function listed in FIA_SOS.2.2 the TOE shall be configured to generate the authentication values; the evaluator shall check that the TOE produces the authentication values.

FIA_UAU.2 User Authentication before Any Action

TSS

The evaluator shall examine the TSS to determine that it describes the identification and authentication process for each supported method (PIN/try-PIN, salted hash, etc.), the circumstances in which each supported method is used, and constitutes "successful authentication."

The evaluator shall examine the TSS to determine that it describes which actions are allowed before user identification and authentication. The evaluator shall also determine that the TSS describes, for each action that does require identification and authentication, the method and circumstances by which the authentication is performed (e.g., as per the application note, the TSF may authenticate a user once rather than each time access to an SDO is attempted; the TSS shall describe when authentication is or is not required in order to perform a TSF-mediated action).

Guidance

The evaluator shall examine the guidance documentation to determine that any necessary preparatory steps (e.g., establishing valid credential material such as PIN) to logging in are described. For each supported the login method, the evaluator shall ensure the guidance documentation provides clear instructions for successfully logging on.

KMD

There are no KMD evaluation activities for this component.

Tests

The evaluator shall use the guidance documentation to configure the appropriate credentials supported for each authentication method. For that authentication method, the evaluator shall attempt to perform TSF-mediated actions that require successful use of that authentication method and subsequently show that providing correct I&A information results in the ability to perform the requested action, while providing incorrect information results in denial of access.

FIA_UAU.5 Multiple Authentication Mechanisms

TSS

The evaluator shall examine the TSS and ensure that it describes the authentication mechanisms used to support user authentication for the Prove service as well as how each authentication mechanism provides authentication for the Prove service.

Guidance

If the supported authentication mechanisms are configurable, the evaluator shall examine the operational guidance to verify that it describes how to configure the authentication mechanisms used to provide authentication for the Prove service.

KMD

There are no KMD evaluation activities for this component.

Tests

For each supported authentication mechanism, the evaluator shall verify that valid credentials result in successful authentication and invalid credentials result in a rejected authentication attempt. If the supported authentication mechanisms are configurable, the evaluator shall follow the operational guidance to enable/disable the various mechanisms and ensure that valid credentials do not result in successful authentication if that mechanism is disabled, or that there is no interface to provide authentication credentials over an external interface when that mechanism is disabled.

FIA-UAU.6 Re-Authenticating

TSS

The evaluator shall examine the TSS to determine that it describes each of the options for reauthorization.

Guidance

There are no AGD evaluation activities for this component.

KMD

There are no KMD evaluation activities for this component.

Tests

The evaluator shall use the configuration guidance to create an SDO with each of the options for reauthorization, then identify functions to exercise each of these options, then execute these options providing the correct authorization confirming that the operation succeeded with respect to the reauthorization option chosen. The evaluator shall then attempt to execute these functions while providing the incorrect authorization and confirming that the operation fails.

2.1.4 Security Management (FMT)

FMT_MOF_EXT.1 Management of Security Functions Behavior

FMT_MSA.1 Management of Security Attributes

FMT_MSA.3 Static Attribute Initialization

2.2 Evaluation Activities for Optional SFRs

2.2.1 Cryptographic Support (FCS)

FCS_CKM.1 Cryptographic Key Generation

TSS

The evaluator shall examine the TSS to verify that it describes how the TOE obtains a cryptographic key through importation of keys from external sources as specified in FDP_ITC_EXT.1 and FDP_ITC_EXT.2. The evaluator shall also examine the TSS to determine whether it describes any supported asymmetric or symmetric key generation functionality consistent with the claims made in FCS_CKM.1.1.

Guidance

The evaluator shall verify that the guidance instructs the administrator how to configure the TOE to use the selected key types for all uses identified in the ST.

KMD

The evaluator shall confirm that the KMD describes:

- The parsing interface and how the TSF imports keys for internal use
- The asymmetric key generation interfaces and how the TSF internally creates asymmetric keys, if claimed
- The symmetric key generation interfaces and how the TSF internally creates symmetric keys, if claimed

If the TOE uses the generated key in a key chain/hierarchy then the KMD shall describe how the key is used as part of the key chain/hierarchy.

Tests

Testing for this function is performed in conjunction with FDP_ITC_EXT.1 and FDP_ITC_EXT.2. If asymmetric or symmetric key generation functionality is claimed, testing for this function is also performed in conjunction with FCS_CKM.1/AK or FCS_CKM.1/SK.

FCS_CKM.1/AK Cryptographic Key Generation (Asymmetric Keys)

TSS

The evaluator shall examine the TSS to verify that it describes how the TOE generates an asymmetric key based on the methods selected from cPP Table 13: "Supported Methods for Asymmetric Key Generation". The evaluator shall examine the TSS to verify that it describes how the TOE invokes the methods selected in the ST from the same table. The evaluator shall examine the TSS to verify that it identifies the usage for each row identifier (key type, key size, and list of standards) selected in the ST.

Guidance

The evaluator shall verify that the AGD guidance instructs the administrator how to configure the TOE to use the selected key types for all uses identified in the ST.

KMD

If the TOE uses the generated key in a key chain/hierarchy then the evaluator shall confirm that the KMD describes:

- If <u>AK1</u> is selected, then the KMD describes which methods for generating p and q are used
- How the key is used as part of the key chain/hierarchy.

Tests

The following tests require the developer to provide access to a test platform that provides the evaluator with tools that are typically not found on factory products.

AK1: RSA Key Generation

The below tests are derived from The 186-4 RSA Validation System (RSA2VS), Updated 8 July 2014, Section 6.2, from the National Institute of Standards and Technology.

The evaluator shall verify the implementation of RSA Key Generation by the TOE using the Key Generation test. This test verifies the ability of the TSF to correctly produce values for the key components including the public verification exponent e, the private prime factors p and q, the public modulus n and the calculation of the private signature exponent d.

FIPS 186-4 Key Pair generation specifies 5 methods for generating the primes p and q.

These are:

1. Random Primes:
   - Provable primes
   - Probable primes
2. Primes with Conditions:
   - Primes p1, p2, q1, q2, p and q shall all be provable primes.
   - Primes p1, p2, q1, and q2 shall be provable primes and p and q shall be probable primes
   - Primes p1, p2, q1, q2, p and q shall all be probable primes.

To test the key generation method for the Random Provable primes method and for all the Primes with Conditions methods, the evaluator must seed the TSF key generation routine with sufficient data to deterministically generate the RSA key pair.

For each key length supported, the evaluator shall have the TSF generate 25 key pairs. The evaluator shall verify the correctness of the TSF's implementation by comparing values generated by the TSF with those generated by a known good implementation using the same input parameters.

If the TOE generates Random Probable Primes then if possible, the Random Probable primes method should also be verified against a known good implementation as described above. If verification against a known good implementation is not possible, the evaluator shall have the TSF generate 25 key pairs for each supported key length nlen and verify that all of the following are true:

- $n = p*q$
- p and q are probably prime according to Miller-Rabin tests with error probability $<2^{-125}$
- $2^{16} < e < 2^{256}$ and e is an odd integer
- $GCD(p-1,e) = 1$
- $GCD(q-1,e) = 1$
- $|p-q| > 2^{(nlen/2 - 100)}$
- $p >= squareroot(2)*(2^{(nlen/2 -1)})$
- $q >= squareroot(2)*(2^{(nlen/2 -1)})$
- $2^{(nlen/2)} < d < LCM(p-1,q-1)$
- $e*d = 1 \mod LCM(p-1,q-1)$

AK2 & AK3: ECC Key Generation with NIST and Brainpool Curves

These tests are derived from The FIPS 186-4 Elliptic Curve Digital Signature Algorithm Validation System (ECDSA2VS), Updated 18 Mar 2014, Section 6.

ECC Key Generation Test

For each selected curve, and for each key pair generation method as described in FIPS 186-4, section B.4, the evaluator shall require the implementation under test to generate 10 private/public key pairs (d, Q). The private key, d, shall be generated using a random bit generator as specified in FCS_RBG_EXT.1. The private key, d, is used to compute the public key, Q'. The evaluator shall confirm that $0<d<n$ (where n is the order of the group), and the computed value Q' is then compared to the generated public/private key pairs' public key, Q, to confirm that Q is equal to Q'.

Public Key Validation (PKV) Test

For each supported curve, the evaluator shall generate 12 private/public key pairs using the key generation function of a known good implementation and modify six of the public key values so that they are incorrect, leaving six values unchanged (i.e., correct). To determine correctness, the evaluator shall submit the 12 key pairs to the public key validation (PKV) function of the TOE and shall confirm that the results correspond as expected to the modified and unmodified values.

AK4: DSA Key Generation using Finite-Field Cryptography (FFC)

The evaluator shall verify the implementation of the Parameters Generation and the Key Generation for FFC by the TOE using the Parameter Generation and Key Generation test. This test verifies the ability of the TSF to correctly produce values for the field prime p, the cryptographic prime q (dividing p-1), the cryptographic group generator g, and the calculation of the private key x and public key y.

The Parameter generation specifies 2 ways (or methods) to generate the cryptographic prime q and the field prime p:

- Primes q and p shall both be provable primes
- Primes q and field prime p shall both be probable primes

and two ways to generate the cryptographic group generator g:

- Generator g constructed through a verifiable process
- Generator g constructed through an unverifiable process.

The Key generation specifies 2 ways to generate the private key x:

- len(q) bit output of RBG where $1 \leq x \leq q-1$
- len(q) + 64 bit output of RBG, followed by a mod q-1 operation and a +1 operation, where $1 \leq x \leq q-1$.

The security strength of the RBG must be at least that of the security offered by the FFC parameter set.

To test the cryptographic and field prime generation method for the provable primes method or the group generator g for a verifiable process, the evaluator must seed the TSF parameter generation routine with sufficient data to deterministically generate the parameter set.

For each key length supported, the evaluator shall have the TSF generate 25 parameter sets and key pairs. The evaluator shall verify the correctness of the TSF's implementation by comparing values generated by the TSF with those generated from a known good implementation. Verification must also confirm

- g != 0,1
- q divides p-1
- g^q mod p = 1
- g^x mod p = y

for each FFC parameter set and key pair.

AK5: Curve25519 Key Generation

The evaluator shall require the implementation under test (IUT) to generate 10 private/public key pairs. The private key shall be generated as specified in RFC 7748 using an approved random bit generator (RBG) and shall be written in littleendian order (least significant byte first. To determine correctness, the evaluator shall submit the generated key pairs to the public key verification (PKV) function of a known good implementation.

Note: Assuming the PKV function of the good implementation will (using little-endian order):

- Confirm the private and public keys are 32-byte values
- Confirm the three least significant bits of the first byte of the private key are zero
- Confirm the most significant bit of the last byte is zero
- Confirm the second most significant bit of the last byte is one
- Calculate the expected public key from the private key and confirm it matches the supplied public key

The evaluator shall generate 10 private/public key pairs using the key generation function of a known good implementation and modify 5 of the public key values so that they are incorrect, leaving five values unchanged (i.e. correct). The evaluator shall obtain in response a set of 10 PASS/FAIL values.

FCS_CKM.1/SK Cryptographic Key Generation (Symmetric Encryption Key)

TSS
The evaluator shall examine the TSS to verify that it describes how the TOE obtains an SK through direct generation as specified in FCS_RBG_EXT.1, FCS_COP.1/KDF, or FCS_COP.1/PBKDF. The evaluator shall review the TSS to verify that it describes how the ST invokes the functionality described by FCS_RBG_EXT.1 and FCS_COP.1/PBKDF where applicable.

[conditional] If the symmetric key is generated by an RBG, the evaluator shall review the TSS to determine that it describes how the functionality described by FCS_RBG_EXT.1 is invoked. The evaluator uses the description of the RBG functionality in FCS_RBG_EXT.1 or documentation available for the operational environment to determine that the key size being requested is greater than or equal to the key size and mode to be used for the encryption/decryption of the data.
Guidance
The evaluator shall verify that the AGD guidance instructs the administrator how to configure the TOE to use the selected key types for all uses identified in the ST.
KMD
The evaluator shall confirm that the KMD describes, as applicable:

- The RBG interface and how the ST uses it in symmetric key generation
- The KDF interface and how the ST uses it in symmetric key generation
- The PBKDF interface and how the ST uses it in symmetric key generation
- If the TOE uses the generated key in a key chain/hierarchy then the KMD shall describe how the ST uses the key as part of the key chain/hierarchy.

Tests
For each selected key generation method, the evaluator shall configure the selected generation capability. The evaluator shall use the description of the RBG interface to verify that the TOE requests and receives an amount of RBG output greater than or equal to the requested key size. The evaluator shall perform the tests as described for FCS_COP.1/KDF and FCS_COP.1/PBKDF.

FCS_CKM.1/KEK Cryptographic Key Generation (Key Encryption Key)

TSS
The evaluator shall examine the key hierarchy section of the TSS to ensure that the formation of all KEKs is described and that the key sizes match that described by the ST author. The evaluator shall examine the key hierarchy section of the TSS to ensure that each KEK encrypts keys of equal or lesser security strength using one of the selected methods.

[conditional] If the KEK is generated according to an asymmetric key scheme, the evaluator shall review the TSS to determine that it describes how the functionality described by FCS_CKM.1/AK is invoked. The evaluator uses the description of the key generation functionality in FCS_CKM.1/AK or documentation available for the operational environment to determine that the key strength being requested is greater than or equal to 112 bits.

[conditional] If the KEK is generated according to a symmetric key scheme, the evaluator shall review the TSS to determine that it describes how the functionality described by FCS_CKM.1/SK is invoked. The evaluator uses the description of the RBG functionality in FCS_RBG_EXT.1, or the key derivation functionality in either FCS_CKM_EXT.5 or FCS_COP.1/PBKDF, depending on the key generation method claimed, to determine that the key size being requested is greater than or equal to the key size and mode to be used for the encryption/decryption of the data.

[conditional] If the KEK is formed from derivation, the evaluator shall verify that the TSS describes the method of derivation and that this method is consistent with FCS_CKM_EXT.5.

Guidance
There are no guidance evaluation activities for this component.
KMD
The evaluator shall iterate through each of the methods selected by the ST and confirm that the KMD describes the applicable selected methods.
Tests
The evaluator shall iterate through each of the methods selected by the ST and perform all applicable tests from the selected methods.

FCS_CKM.2 Cryptographic Key Establishment

TSS
The evaluator shall examine the TSS to ensure that ST supports at least one key establishment scheme. The evaluator also ensures that for each key establishment scheme selected by the ST in FCS_CKM.2.1 it also supports one or more corresponding methods selected in FCS_COP.1/KAT. If the ST selects RSA in FCS_CKM.2.1, then the TOE must support one or more of "KAS1," or "KAS2," "KTS-OAEP," from FCS_COP.1/KAT. If the ST selects elliptic curve-based, then the TOE must support one or more of "ECDH-NIST" or "ECDH-BPC" from FCS_COP.1/KAT. If the ST selects Diffie-Hellman-based key establishment, then the TOE must support "DH" from FCS_COP.1/KAT.

Guidance
The evaluator shall verify that the guidance instructs the administrator how to configure the TOE to use the selected key establishment scheme.
KMD
There are no KMD evaluation activities for this component.
Tests
Testing for this SFR is performed under the corresponding functions in FCS_COP.1/KAT.

FCS_CKM.4 Cryptographic Key Destruction

TSS
The evaluator shall examine the TSS to ensure it lists all relevant keys and keying material (describing the source of the data, all memory types in which the data is stored (covering storage both during and outside of a session, and both plaintext and non-plaintext forms of the data)), all relevant destruction situations (including the point in time at which the destruction occurs; e.g. factory reset or device wipe function, change of authorization data, change of DEK, completion of use of an intermediate key) and the destruction method used in each case. The evaluator shall confirm that the description of the data and storage locations is consistent with the functions carried out by the TOE (e.g. that all keys in the key chain are accounted for). (Where keys are stored encrypted or wrapped under another key then this may need to be explained in order to allow the evaluator to confirm the consistency of the description of keys with the TOE functions).

The evaluator shall check that the TSS identifies any configurations or circumstances that may not conform to the key destruction requirement (see further discussion in the AGD section below). Note that reference may be made to the AGD for description of the detail of such cases where destruction may be prevented or delayed.

Where the ST specifies the use of "a value that does not contain any sensitive data" to overwrite keys, the evaluator shall examine the TSS to ensure that it describes how that pattern is obtained and used, and that this justifies the claim that the pattern does not contain any sensitive data.

Guidance
The evaluator shall check that the guidance documentation for the TOE requires users to ensure that the TOE remains under the user's control while a session is active.

A TOE may be subject to situations that could prevent or delay data destruction in some cases. The evaluator shall check that the guidance documentation identifies configurations or circumstances that may not strictly conform to the key destruction requirement, and that this description is consistent with the relevant parts of the TSS (and KMD). The evaluator shall check that the guidance documentation provides guidance on situations where key destruction may be delayed at the physical layer, identifying any additional mitigation actions for the user (e.g. there might be some operation the user can invoke, or the user might be advised to retain control of the device for some particular time to maximise the probability that garbage collection will have occurred).

For example, when the TOE does not have full access to the physical memory, it is possible that the storage may implement wear-levelling and garbage collection. This may result in additional copies of the data that are logically inaccessible but persist physically. Where available, the TOE might then describe use of the TRIM command and garbage collection to destroy these persistent copies upon their deletion (this would be explained in TSS and guidance documentation).

Where TRIM is used then the TSS or guidance documentation is also expected to describe how the keys are stored such that they are not inaccessible to TRIM, (e.g. they would need not to be contained in a file less than 982 bytes which would be completely contained in the master file table.

KMD
The evaluator shall examine the KMD to verify that it identifies and describes the interfaces that are used to service commands to read/write memory. The evaluator shall examine the interface description for each different media type to ensure that the interface supports the selections made by the ST author.

45 The evaluator shall examine the KMD to ensure that all keys and keying material identified in the TSS and KMD have been accounted for.

46 Note that where selections include 'destruction of reference to the key directly followed by a request for garbage collection' (for volatile memory) then the evaluator shall examine the KMD to ensure that it explains the nature of the destruction of the reference, the request for garbage collection, and of the garbage collection process itself.

Tests
The following tests require the developer to provide access to a test platform that provides the evaluator with tools that are typically not found on factory products.

The evaluator shall perform the following tests:

- **Test 1:** Applied to each key or keying material held as plaintext in volatile memory and subject to destruction by overwrite by the TOE (whether or not the plaintext value is subsequently encrypted for storage in volatile or non-volatile memory).

  The evaluator shall:
  1. Record the value of the key or keying material.
  2. Cause the TOE to dump the SDO/SDE memory of the TOE into a binary file.
  3. Search the content of the binary file created in Step #2 to locate all instances of the known key value from Step #1.

     Note that the primary purpose of Step #3 is to demonstrate that appropriate search commands are being used for Steps #8 and #9.
  4. Cause the TOE to perform normal cryptographic processing with the key from Step #1.
  5. Cause the TOE to destroy the key.
  6. Cause the TOE to stop execution but not exit.
  7. Cause the TOE to dump the SDO/SDE memory of the TOE into a binary file.
  8. Search the content of the binary file created in Step #7 for instances of the known key value from Step #1.
  9. Break the key value from Step #1 into an evaluator-chosen set of fragments and perform a search using each fragment. (Note that the evaluator shall first confirm with the developer how the key is normally stored, in order to choose fragment sizes that are the same or smaller than any fragmentation of the data that may be implemented by the TOE. The endianness or byte-order should also be taken into account in the search.)

  Steps #1-8 ensure that the complete key does not exist anywhere in volatile memory. If a copy is found, then the test fails.

  Step #9 ensures that partial key fragments do not remain in memory. If the evaluator finds a 32-or-greater-consecutive-bit fragment, then fail immediately. Otherwise, there is a chance that it is not within the context of a key (e.g., some random bits that happen to match). If this is the case the test should be repeated with a different key in Step #1. If a fragment is also found in this repeated run then the test fails unless the developer provides a reasonable explanation for the collision, then the evaluator may give a pass on this test.

- **Test 2:** Applied to each key and keying material held in non-volatile memory and subject to destruction by overwrite by the TOE.
  1. Record the value of the key or keying material.
  2. Cause the TOE to perform normal cryptographic processing with the key from Step #1.
  3. Search the non-volatile memory the key was stored in for instances of the known key value from Step #1.

     Note that the primary purpose of Step #3 is to demonstrate that appropriate search commands are being used for Steps #5 and #6.
  4. Cause the TOE to clear the key.
  5. Search the non-volatile memory in which the key was stored for instances of the known key value from Step #1. If a copy is found, then the test fails.
  6. Break the key value from Step #1 into an evaluator-chosen set of fragments and perform a search using each fragment. (Note that the evaluator shall first confirm with the developer how the key is normally stored, in order to choose fragment sizes that are the same or smaller than any fragmentation of the data that may be implemented by the TOE. The endianness or byte-order should also be taken into account in the search).

  Step #6 ensures that partial key fragments do not remain in non-volatile memory. If the evaluator finds a 32-or-greater-consecutive-bit fragment, then fail immediately. Otherwise, there is a chance that it is not within the context of a key (e.g., some random bits that happen to match). If this is the case the test should be repeated with a different key in Step #1. If a fragment is also found in this repeated run then the test fails unless the developer provides a reasonable explanation for the collision, then the evaluator may give a pass on this test.

- **Test 3:** Applied to each key and keying material held in non-volatile memory and subject to destruction by overwrite by the TOE.
  1. Record memory of the key or keying material.
  2. Cause the TOE to perform normal cryptographic processing with the key from Step #1.
  3. Cause the TOE to clear the key. Record the value to be used for the overwrite of the key.
  4. Examine the memory from Step #1 to ensure the appropriate pattern (recorded in Step #3) is used.

  The test succeeds if correct pattern is found in the memory location. If the pattern is not found, then the test fails.

FCS_CKM_EXT.4 Cryptographic Key and Key Material Destruction Timing

TSS
The evaluator shall verify the TSS provides a high-level description of what it means for keys and key material to be no longer needed and when this data should be expected to be destroyed.
Guidance
There are no guidance evaluation activities for this component.
KMD
The evaluator shall verify that the KMD includes a description of the areas where keys and key material reside and when this data is no longer needed.

The evaluator shall verify that the KMD includes a key lifecycle that includes a description where key materials reside, how the key materials are used, how it is determined that keys and key material are no longer needed, and how the data is destroyed once it is no longer needed. The evaluator shall also verify that all key destruction operations are performed in a manner specified by FCS_CKM.4.

Tests
There are no test evaluation activities for this component

FCS_CKM_EXT.5 Cryptographic Key Derivation

TSS
The evaluator shall check that the TSS includes a description of the key derivation functions and shall check that this uses a key derivation algorithm and key sizes according to the specification selected in the ST out of the table as provided in the cPP table per row. The evaluator shall confirm that the TSS supports the selected methods.

If KeyDrv5 is selected, the evaluator shall verify that the TSS shows that the total length of the concatenated keys used as input to the KDF is greater than or equal to the length of the output from the KDF.

[conditional] If key combination is used to form a KEK, the evaluator shall verify that the TSS describes the method of combination and that this method is either an XOR, a KDF, or encryption.

[conditional] If a KDF is used to form a KEK, the evaluator shall ensure that the TSS includes a description of the key derivation function and shall verify the key derivation uses an approved derivation mode and key expansion algorithm according to SP 800-108.

[conditional] If key concatenation is used to derive KEKs (KeyDrv5), the evaluator shall ensure the TSS includes a description of the randomness extraction step, including the following:

- The description must include how an approved untruncated MAC function is being used for the randomness extraction step and the evaluator must verify the TSS describes that the output length (in bits) of the MAC function is at least as large as the targeted security strength (in bits) of the parameter set employed by the key establishment scheme (see Tables 1-3 of SP 800-56C).
- The description must include how the MAC function being used for the randomness extraction step is related to the PRF used in the key expansion and verify the TSS description includes the correct MAC function:
    - If an HMAC-hash is used in the randomness extraction step, then the same HMAC-hash (with the same hash function hash) is used as the PRF in the key expansion step.
    - If an AES-CMAC (with key length 128, 192, or 256 bits) is used in the randomness extraction step, then AES-CMAC with a 128-bit key is used as the PRF in the key expansion step.
- The description must include the lengths of the salt values being used in the randomness extraction step and the evaluator shall verify the TSS description includes correct salt lengths:
    - If an HMAC-hash is being used as the MAC, the salt length can be any value up to the maximum bit length permitted for input to the hash function hash.
    - If an AES-CMAC is being used as the MAC, the salt length shall be the same length as the AES key (i.e. 128, 192, or 256 bits).

Guidance
The evaluator shall verify that the AGD guidance instructs the administrator how to configure the TOE to use the selected key types for all uses identified in the ST.
KMD
The evaluator shall examine the KMD to ensure that:

- The KMD describes the complete key derivation chain and the description must be consistent with the description in the TSS. For all key derivations the TOE must use a method as described in the cPP table. There should be no uncertainty about how a key is derived from another in the chain.
- The length of the key derivation key is defined by the PRF. The evaluator should check whether the key derivation key length is consistent with the length provided by the selected PRF.
- If a key is used as an input to several KDFs, each invocation must use a distinct context string. If the output of a KDF execution is used for multiple cryptographic keys, those keys must be disjoint segments of the output.

Tests
The following tests require the developer to provide access to a test platform that provides the evaluator with tools that are typically not found on factory products.

The evaluator shall perform one or more of the following tests to verify the correctness of the key derivation function, depending on the specific functions that are supported:

Preconditions for testing:

- Specification of input parameter to the key derivation function to be tested
- Specification of further required input parameters

- Access to derived keys

The following table maps the data fields in the tests below to the notations used in SP 800-108 and SP 800-56C

Data Fields	Notations	
	SP 800-108	SP 800-56C
Pseudorandom function	PRF	PRF
Counter length	r	r
Length of output of PRF	r	r
Length of derived keying material	L	L
Length of input values	I_length	I_length
Pseudorandom input values I	K1 (key derivation key)	Z (shared secret)
Pseudorandom salt values		S
Randomness extraction MAC	n/a	MAC

The below tests are derived from Key Derivation using Pseudorandom Functions (SP 800-108) Validation System (KBKDFVS), Updated 4 January 2016, Section 6.2, from the National Institute of Standards and Technology.

KeyDrv1: Counter Mode Tests:
The evaluator shall determine the following characteristics of the key derivation function:

- One or more pseudorandom functions that are supported by the implementation (PRF).
- One or more of the values {8, 16, 24, 32} that equal the length of the binary representation of the counter (r).
- The length (in bits) of the output of the PRF (h).
- Minimum and maximum values for the length (in bits) of the derived keying material (L). These values can be equal if only one value of L is supported. These must be evenly divisible by h.
- Up to two values of L that are NOT evenly divisible by h.
- Location of the counter relative to fixed input data: before, after, or in the middle.
  - Counter before fixed input data: fixed input data string length (in bytes), fixed input data string value.
  - Counter after fixed input data: fixed input data string length (in bytes), fixed input data string value.
  - Counter in the middle of fixed input data: length of data before counter (in bytes), length of data after counter (in bytes), value of string input before counter, value of string input after counter.
- The length (I_length) of the input values I.

For each supported combination of I_length, MAC, salt, PRF, counter location, value of r, and value of L, the evaluator shall generate 10 test vectors that include pseudorandom input values I, and pseudorandom salt values. If there is only one value of L that is evenly divisible by h, the evaluator shall generate 20 test vectors for it. For each test vector, the evaluator shall supply this data to the TOE in order to produce the keying material output.

The results from each test may either be obtained by the evaluator directly or by supplying the inputs to the implementer and receiving the results in response. To determine correctness, the evaluator shall compare the resulting values to those obtained by submitting the same inputs to a known good implementation.

KeyDrv2: Feedback Mode Tests:

The evaluator shall determine the following characteristics of the key derivation function:

- One or more pseudorandom functions that are supported by the implementation (PRF).
- The length (in bits) of the output of the PRF (h).
- Minimum and maximum values for the length (in bits) of the derived keying material (L). These values can be equal if only one value of L is supported. These must be evenly divisible by h.
- Up to two values of L that are NOT evenly divisible by h.
- Whether or not zero-length IVs are supported.
- Whether or not a counter is used, and if so:
  - One or more of the values {8, 16, 24, 32} that equal the length of the binary representation of the counter (r).
  - Location of the counter relative to fixed input data: before, after, or in the middle.
    - Counter before fixed input data: fixed input data string length (in bytes), fixed input data string value.
    - Counter after fixed input data: fixed input data string length (in bytes), fixed input data string value.
    - Counter in the middle of fixed input data: length of data before counter (in bytes), length of data after counter (in bytes), value of string input before counter, value of string input after counter.
- The length (I_length) of the input values L.

For each supported combination of I_length, MAC, salt, PRF, counter location (if a counter is used), value of r (if a counter is used), and value of L, the evaluator shall generate 10 test vectors that include pseudorandom input values I and pseudorandom salt values. If the KDF supports zero-length IVs, five of these test vectors will be accompanied by pseudorandom IVs and the other five will use zerolength IVs. If zero-length IVs are not supported, each test vector will be accompanied by an pseudorandom IV. If there is only one value of L that is evenly divisible by h, the evaluator shall generate 20 test vectors for it.

For each test vector, the evaluator shall supply this data to the TOE in order to produce the keying material output. The results from each test may either be obtained by the evaluator directly or by supplying the inputs to the implementer and receiving the results in response. To determine correctness, the evaluator shall compare the resulting values to those obtained by submitting the same inputs to a known good implementation.

KeyDrv3: Double Pipeline Iteration Mode Tests:

The evaluator shall determine the following characteristics of the key derivation function:

- One or more pseudorandom functions that are supported by the implementation (PRF).
- The length (in bits) of the output of the PRF (h).
- Minimum and maximum values for the length (in bits) of the derived keying material (L). These values can be equal if only one value of L is supported. These must be evenly divisible by h.
- Up to two values of L that are NOT evenly divisible by h.
- Whether or not a counter is used, and if so:
    - One or more of the values {8, 16, 24, 32} that equal the length of the binary representation of the counter (r).
    - Location of the counter relative to fixed input data: before, after, or in the middle.
        - Counter before fixed input data: fixed input data string length (in bytes), fixed input data string value.
        - Counter after fixed input data: fixed input data string length (in bytes), fixed input data string value.
        - Counter in the middle of fixed input data: length of data before counter (in bytes), length of data after counter (in bytes), value of string input before counter, value of string input after counter.
- The length (I_length) of the input values I.

For each supported combination of I_length, MAC, salt, PRF, counter location (if a counter is used), value of r (if a counter is used), and value of L, the evaluator shall generate 10 test vectors that include pseudorandom input values I, and pseudorandom salt values. If there is only one value of L that is evenly divisible by h, the evaluator shall generate 20 test vectors for it.

For each test vector, the evaluator shall supply this data to the TOE in order to produce the keying material output. The results from each test may either be obtained by the evaluator directly or by supplying the inputs to the implementer and receiving the results in response. To determine correctness, the evaluator shall compare the resulting values to those obtained by submitting the same inputs to a known good implementation.

KeyDrv4: Intermediate Keys Method

If the selected algorithm is a hash then the testing of the hash primitive is the only required Evaluation Activity. If the selected algorithm is XOR then no separate primitive testing is necessary.

KeyDrv5: Concatenated Keys Method

The evaluator should confirm that the combined length of the concatenated keys should be at least as long as the keysize of the selected methods. There are no other tests other than for the methods selected for this row performed for KeyDrv1, KeyDrv2, and KeyDrv3.

KeyDrv6: Two Keys Method

The evaluator should confirm that the combined length of the two keys should be at least as long as the keysize of the selected methods. There are no other tests other than for the methods selected for this row from FCD_COP.1/SK.

KeyDrv7: Shared Secret, Salt, Output Length, Fixed Information Method

For each supported selection of PRF, length of shared secret (Z) [selection: 128, 256] bits, length of salt (S) [selection: length of input block of PRF, one-half length of input block of PRF, 0] bits, output length (L) [selection: 128, 256] bits, and length of fixed information (FixedInfo) [selection: length of on input block of PRF, onehalf length of input block of PRF, 0] bits, the evaluator shall generate 10 test vectors that include pseudorandom input values for Z, salt values (for non-zero lengths, otherwise, omit) and fixed information (for non-zero lengths, otherwise, omit).

For each test vector, the evaluator shall supply this data to the TOE in order to produce the keying material output. The results from each test may either be obtained by the evaluator directly or by supplying the inputs to the implementer and receiving the results in response. To determine correctness, the evaluator shall compare the resulting values to those obtained by submitting the same inputs to a known good implementation.

KeyDrv8: Shared Secret, Salt, IV, Output Length, Fixed Information Method

For each supported selection of PRF, length of shared secret (Z), length of salt, length of initialization vector (IV), output length (L), and length of fixed information (FixedInfo), the evaluator shall generate 10 test vectors that include pseudorandom input values for Z, salt values (for non-zero lengths, otherwise, omit), IV (for non-zero lengths, otherwise, use a vector of length equal to length of input block of PRF and fill with zeros), and fixed information (for non-zero lengths, otherwise, omit).

For each test vector, the evaluator shall supply this data to the TOE in order to produce the keying material output. The results from each test may either be obtained by the evaluator directly or by supplying the inputs to the implementer and receiving the results in response. To determine correctness, the evaluator shall compare the resulting values to those obtained by submitting the same inputs to a known good

implementation.

FCS_COP.1/Hash Cryptographic Operation (Hashing)

TSS
The evaluator shall check that the association of the hash function with other TSF cryptographic functions (for example, the digital signature verification function) is documented in the TSS. The evaluator shall also check that the TSS identifies whether the implementation is bit-oriented or byte-oriented.

Guidance
The evaluator checks the AGD documents to determine that any configuration that is required to configure the required hash sizes is present. The evaluator also checks the AGD documents to confirm that the instructions for establishing the evaluated configuration use only those hash algorithms selected in the ST.

KMD
There are no KMD evaluation activities for this component.

Tests
The following tests require the developer to provide access to a test platform that provides the evaluator with tools that are typically not found on factory products.

SHA-1 and SHA-2 Tests

The tests below are derived from the "The Secure Hash Algorithm Validation System (SHAVS), Updated: May 21, 2014" from the National Institute of Standards and Technology.

The TSF hashing functions can be implemented with one of two orientations. The first is a byte-oriented implementation: this hashes messages that are an integral number of bytes in length (i.e., the length (in bits) of the message to be hashed is divisible by 8). The second is a bit-oriented implementation: this hashes messages of arbitrary length. Separate tests for each orientation are given below.

The evaluator shall perform all of the following tests for each hash algorithm and orientation implemented by the TSF and used to satisfy the requirements of this PP. The evaluator shall compare digest values produced by a known-good SHA implementation against those generated by running the same values through the TSF.

Short Messages Test, Bit-oriented Implementation

The evaluators devise an input set consisting of m+1 messages, where m is the block length of the hash algorithm in bits (see SHA Properties Table). The length of the messages ranges sequentially from 0 to m bits. The message text shall be pseudorandomly generated. The evaluators compute the message digest for each of the messages and ensure that the correct result is produced when the messages are provided to the TSF.

Short Messages Test, Byte-oriented Implementation

The evaluators devise an input set consisting of m/8+1 messages, where m is the block length of the hash algorithm in bits (see SHA Properties Table). The length of the messages ranges sequentially from 0 to m/8 bytes, with each message being an integral number of bytes. The message text shall be pseudo-randomly generated. The evaluators compute the message digest for each of the messages and ensure that the correct result is produced when the messages are provided to the TSF.

Selected Long Messages Test, Bit-oriented Implementation

The evaluators devise an input set consisting of m messages, where m is the block length of the hash algorithm in bits (see SHA Properties Table). The length of the ith message is $m + 99*i$, where $1 \leq i \leq m$. The message text shall be pseudorandomly generated. The evaluators compute the message digest for each of the messages and ensure that the correct result is produced when the messages are provided to the TSF.

Selected Long Messages Test, Byte-oriented Implementation

The evaluators devise an input set consisting of m/8 messages, where m is the block length of the hash algorithm in bits (see SHA Properties Table). The length of the ith message is $m + 8*99*i$, where $1 \leq i \leq m/8$. The message text shall be pseudorandomly generated. The evaluators compute the message digest for each of the messages and ensure that the correct result is produced when the messages are provided to the TSF.

Pseudo-randomly Generated Messages Test

The evaluators randomly generate a seed that is n bits long, where n is the length of the message digest produced by the hash function to be tested. The evaluators then formulate a set of 100 messages and associated digests by following the algorithm provided in Figure 1 of SHAVS, section 6.4. The evaluators then ensure that the correct result is produced when the messages are provided to the TSF.

SHA-3 Tests

The tests below are derived from the The Secure Hash Algorithm-3 Validation System (SHA3VS), Updated: April 7, 2016, from the National Institute of Standards and Technology.

For each SHA-3-XXX implementation, XXX represents d, the digest length in bits. The capacity, c, is equal to 2d bits. The rate is equal to 1600-c bits.

65 The TSF hashing functions can be implemented with one of two orientations. The first is a bit-oriented mode that hashes messages of arbitrary length. The second is a byte-oriented mode that hashes messages that are an integral number of bytes in length (i.e., the length (in bits) of the message to be hashed is divisible by 8). Separate tests for each orientation are given below.

The evaluator shall perform all of the following tests for each hash algorithm and orientation implemented by the TSF and used to satisfy the requirements of this PP. The evaluator shall compare digest values produced

by a known-good SHA-3 implementation against those generated by running the same values through the TSF.

Short Messages Test, Bit-oriented Mode

The evaluators devise an input set consisting of rate+1 short messages. The length of the messages ranges sequentially from 0 to rate bits. The message text shall be pseudo-randomly generated. The evaluators compute the message digest for each of the messages and ensure that the correct result is produced when the messages are provided to the TSF. The message of length 0 is omitted if the TOE does not support zero-length messages.

Short Messages Test, Byte-oriented Mode

The evaluators devise an input set consisting of rate/8+1 short messages. The length of the messages ranges sequentially from 0 to rate/8 bytes, with each message being an integral number of bytes. The message text shall be pseudo-randomly generated. The evaluators compute the message digest for each of the messages and ensure that the correct result is produced when the messages are provided to the TSF. The message of length 0 is omitted if the TOE does not support zero-length messages.

Selected Long Messages Test, Bit-oriented Mode

The evaluators devise an input set consisting of 100 long messages ranging in size from rate+(rate+1) to rate+(100*(rate+1)), incrementing by rate+1. (For example, SHA-3-256 has a rate of 1088 bits. Therefore, 100 messages will be generated with lengths 2177, 3266, ..., 109988 bits.) The message text shall be pseudo-randomly generated. The evaluators compute the message digest for each of the messages and ensure that the correct result is produced when the messages are provided to the TSF.

Selected Long Messages Test, Byte-oriented Mode

The evaluators devise an input set consisting of 100 messages ranging in size from (rate+(rate+8)) to (rate+100*(rate+8)), incrementing by rate+8. (For example, SHA-3-256 has a rate of 1088 bits. Therefore 100 messages will be generated of lengths 2184, 3280, 4376, ..., 110688 bits.) The message text shall be pseudorandomly generated. The evaluators compute the message digest for each of the messages and ensure that the correct result is produced when the messages are provided to the TSF.

Pseudo-randomly Generated Messages Monte Carlo) Test, Byte-oriented Mode

The evaluators supply a seed of d bits (where d is the length of the message digest produced by the hash function to be tested. This seed is used by a pseudorandom function to generate 100,000 message digests. One hundred of the digests (every 1000th digest) are recorded as checkpoints. The TOE then uses the same procedure to generate the same 100,000 message digests and 100 checkpoint values. The evaluators then compare the results generated ensure that the correct result is produced when the messages are generated by the TSF.

FCS_COP.1/HMAC Cryptographic Operation (Keyed Hash)

TSS
The evaluator shall examine the TSS to ensure that it specifies the following values used by the HMAC and KMAC functions: output MAC length used.
Guidance
There are no guidance evaluation activities for this component.
KMD
There are no KMD evaluation activities for this component.
Tests
The following test requires the developer to provide access to a test platform that provides the evaluator with tools that are typically not found on factory products.

This test is derived from The Keyed-Hash Message Authentication Code Validation System (HMACVS), updated 6 May 2016.

The evaluator shall provide 15 sets of messages and keys for each selected hash algorithm and hash length/key size/MAC size combination. The evaluator shall have the TSF generate HMAC or KMAC tags for these sets of test data. The evaluator shall verify that the resulting HMAC or KMAC tags match the results from submitting the same inputs to a known-good implementation of the HMAC or KMAC function, having the same characteristics.

FCS_COP.1/KAT Cryptographic Operation (Key Agreement/Transport)

TSS
The evaluator shall ensure that the selected RSA and ECDH key agreement/transport schemes correspond to the key generation schemes selected in FCS_CKM.1/AK, and the key establishment schemes selected in FCS_CKM.2 If the ST selects DH, the TSS shall describe how the implementation meets the relevant sections of RFC 3526 (Section 3-7) and RFC 7919 (Appendices A.1-A.5). If the ST selects ECIES, the TSS shall describe the key sizes and algorithms (e.g. elliptic curve point multiplication, ECDH with either NIST or Brainpool curves, AES in a mode permitted by FCS_COP.1/SKC, a SHA-2 hash algorithm permitted by FCS_COP.1/Hash, and a MAC algorithm permitted by FCS_COP.1/HMAC) that are supported for the ECIES implementation.

The evaluator shall ensure that, for each key agreement/transport scheme, the size of the derived keying material is at least the same as the intended strength of the key agreement/transport scheme, and where feasible this should be twice the intended security strength of the key agreement/transport scheme.

Table 2 of NIST SP 800-57 identifies the key strengths for the different algorithms that can be used for the

various key agreement/transport schemes.

Guidance
There are no guidance evaluation activities for this component.
KMD
There are no KMD evaluation activities for this component.
Tests
The following tests require the developer to provide access to a test platform that provides the evaluator with tools that are typically not found on factory products.

The evaluator shall verify the implementation of the key generation routines of the supported schemes using the following tests:

If ECDH-NIST or ECDH-BPC is claimed:

SP800-56A Key Agreement Schemes

The evaluator shall verify a TOE's implementation of SP800-56A key agreement schemes using the following Function and Validity tests. These validation tests for each key agreement scheme verify that a TOE has implemented the components of the key agreement scheme according to the specifications in the Recommendation. These components include the calculation of the DLC primitives (the shared secret value Z) and the calculation of the derived keying material (DKM) via the Key Derivation Function (KDF). If key confirmation is supported, the evaluator shall also verify that the components of key confirmation have been implemented correctly, using the test procedures described below. This includes the parsing of the DKM, the generation of MACdata and the calculation of MACtag.

Function Test

The Function test verifies the ability of the TOE to implement the key agreement schemes correctly. To conduct this test the evaluator shall generate or obtain test vectors from a known good implementation of the TOE supported schemes. For each supported key agreement scheme-key agreement role combination, KDF type, and, if supported, key confirmation role-key confirmation type combination, the tester shall generate 10 sets of test vectors. The data set consists of one set of domain parameter values (FFC) or the NIST approved curve (ECC) per 10 sets of public keys. These keys are static, ephemeral or both depending on the scheme being tested.

The evaluator shall obtain the DKM, the corresponding TOE's public keys (static or ephemeral), the MAC tags, and any inputs used in the KDF, such as the Other Information field OI and TOE id fields.

If the TOE does not use a KDF defined in SP 800-56A, the evaluator shall obtain only the public keys and the hashed value of the shared secret.

The evaluator shall verify the correctness of the TSF's implementation of a given scheme by using a known good implementation to calculate the shared secret value, derive the keying material DKM, and compare hashes or MAC tags generated from these values.

If key confirmation is supported, the TSF shall perform the above for each implemented approved MAC algorithm.

Validity Test

The Validity test verifies the ability of the TOE to recognize another party's valid and invalid key agreement results with or without key confirmation. To conduct this test, the evaluator shall obtain a list of the supporting cryptographic functions included in the SP800-56A key agreement implementation to determine which errors the TOE should be able to recognize. The evaluator generates a set of 24 (FFC) or 30 (ECC) test vectors consisting of data sets including domain parameter values or NIST approved curves, the evaluator's public keys, the TOE's public/private key pairs, MACTag, and any inputs used in the KDF, such as the other info and TOE id fields.

The evaluator shall inject an error in some of the test vectors to test that the TOE recognizes invalid key agreement results caused by the following fields being incorrect: the shared secret value Z, the DKM, the other information field OI, the data to be MACed, or the generated MACTag. If the TOE contains the full or partial (only ECC) public key validation, The evaluator shall also individually inject errors in both parties' static public keys, both parties' ephemeral public keys and the TOE's static private key to assure the TOE detects errors in the public key validation function or the partial key validation function (in ECC only). At least two of the test vectors shall remain unmodified and therefore should result in valid key agreement results (they should pass).

The TOE shall use these modified test vectors to emulate the key agreement scheme using the corresponding parameters. The evaluator shall compare the TOE's results with the results using a known good implementation verifying that the TOE detects these errors.

If KAS1, KAS2, KTS-OAEP, or RSAES-PKCS1-v1_5 is claimed:

SP800-56B and PKCS#1 Key Establishment Schemes

If the TOE acts as a sender, the following evaluation activity shall be performed to ensure the proper operation of every TOE supported combination of RSA-based key establishment scheme:

To conduct this test the evaluator shall generate or obtain test vectors from a known good implementation of the TOE supported schemes. For each combination of supported key establishment scheme and its options (with or without key confirmation if supported, for each supported key confirmation MAC function if key confirmation is supported, and for each supported mask generation function if KTS-OAEP is supported), the

tester shall generate 10 sets of test vectors. Each test vector shall include the RSA public key, the plaintext keying material, any additional input parameters if applicable, the MacKey and MacTag if key confirmation is incorporated, and the outputted ciphertext. For each test vector, the evaluator shall perform a key establishment encryption operation on the TOE with the same inputs (in cases where key confirmation is incorporated, the test shall use the MacKey from the test vector instead of the randomly generated MacKey used in normal operation) and ensure that the outputted ciphertext is equivalent to the ciphertext in the test vector.

If the TOE acts as a receiver, the following evaluation activities shall be performed to ensure the proper operation of every TOE supported combination of RSA-based key establishment scheme:

To conduct this test the evaluator shall generate or obtain test vectors from a known good implementation of the TOE supported schemes. For each combination of supported key establishment scheme and its options (with our without key confirmation if supported, for each supported key confirmation MAC function if key confirmation is supported, and for each supported mask generation function if KTSOAEP is supported), the tester shall generate 10 sets of test vectors. Each test vector shall include the RSA private key, the plaintext keying material (KeyData), any additional input parameters if applicable, the MacTag in cases where key confirmation is incorporated, and the outputted ciphertext. For each test vector, the evaluator shall perform the key establishment decryption operation on the TOE and ensure that the outputted plaintext keying material (KeyData) is equivalent to the plain text keying material in the test vector. In cases where key confirmation is incorporated, the evaluator shall perform the key confirmation steps and ensure that the outputted MacTag is equivalent to the MacTag in the test vector.

The evaluator shall ensure that the TSS describes how the TOE handles decryption errors. In accordance with NIST Special Publication 800-56B, the TOE must not reveal the particular error that occurred, either through the contents of any outputted or logged error message or through timing variations. If KTS-OAEP is supported, the evaluator shall create separate contrived ciphertext values that trigger each of the three decryption error checks described in NIST Special Publication 800-56B section 7.2.2.3, ensure that each decryption attempt results in an error, and ensure that any outputted or logged error message is identical for each.

DH:

The evaluator shall verify the correctness of each TSF implementation of each supported Diffie-Hellman group by comparison with a known good implementation.

Curve25519:

The evaluator shall verify a TOE's implementation of the key agreement scheme using the following Function and Validity tests. These validation tests for each key agreement scheme verify that a TOE has implemented the components of the key agreement scheme according to the specification. These components include the calculation of the shared secret K and the hash of K.

Function Test

The Function test verifies the ability of the TOE to implement the key agreement schemes correctly. To conduct this test the evaluator shall generate or obtain test vectors from a known good implementation of the TOE supported schemes. For each supported key agreement role and hash function combination, the tester shall generate 10 sets of public keys. These keys are static, ephemeral or both depending on the scheme being tested.

The evaluator shall obtain the shared secret value K, and the hash of K. The evaluator shall verify the correctness of the TSF's implementation of a given scheme by using a known good implementation to calculate the shared secret value K and compare the hash generated from this value.

Validity Test

The Validity test verifies the ability of the TOE to recognize another party's valid and invalid key agreement results. To conduct this test, the evaluator generates a set of 30 test vectors consisting of data sets including the evaluator's public keys and the TOE's public/private key pairs.

The evaluator shall inject an error in some of the test vectors to test that the TOE recognizes invalid key agreement results caused by the following fields being incorrect: the shared secret value K or the hash of K. At least two of the test vectors shall remain unmodified and therefore should result in valid key agreement results (they should pass).

The TOE shall use these modified test vectors to emulate the key agreement scheme using the corresponding parameters. The evaluator shall compare the TOE's results with the results using a known good implementation verifying that the TOE detects these errors.

ECIES:

The evaluator shall verify the correctness of each TSF implementation of each supported use of ECIES by comparison with a known good implementation.

FCS_COP.1/KeyEnc Cryptographic Operation (Key Encryption)

TSS
The evaluator shall examine the TSS to ensure that it identifies whether the implementation of this cryptographic operation for key encryption (including key lengths and modes) is an implementation that is tested in FCS_COP.1/SKC.

The evaluator shall check that the TSS includes a description of the key wrap functions and shall check that

this uses a key wrap algorithm and key sizes according to the specification selected in the ST out of the table as provided in the cPP table.

Guidance
The evaluator checks the AGD documents to confirm that the instructions for establishing the evaluated configuration use only those key wrap functions selected in the ST. If multiple key access modes are supported, the evaluator shall examine the guidance documentation to determine that the method of choosing a specific mode/key size by the end user is described.

KMD
The evaluator shall examine the KMD to ensure that it describes when the key wrapping occurs, that the KMD description is consistent with the description in the TSS, and that for all keys that are wrapped the TOE uses a method as described in the cPP table. No uncertainty should be left over which is the wrapping key and the key to be wrapped and where the wrapping key potentially comes from i.e. is derived from.

If "AES-GCM" or "AES-CCM" is used the evaluator shall examine the KMD to ensure that it describes how the IV is generated and that the same IV is never reused to encrypt different plaintext pairs under the same key. Moreover in the case of GCM, he must ensure that, at each invocation of GCM, the length of the plaintext is at most $(2^{32})-2$ blocks.

Tests
Refer to FCS_COP.1/SKC for the required testing for each symmetric key wrapping method selected from the table and to FCS_COP.1/KAT for the required testing for each asymmetric key wrapping method selected from the table. Each distinct implementation shall be tested separately.

If the implementation of the key encryption operation is the same implementation tested under FCS_COP.1/SKC or FCS_COP.1/KAT, and it has been tested with the same key lengths and modes, then no further testing is required. If key encryption uses a different implementation, (where "different implementation" includes the use of different key lengths or modes), then the evaluator shall additionally test the key encryption implementation using the corresponding tests specified for FCS_COP.1/SKC or FCS_COP.1/KAT.

FCS_COP.1/PBKDF Cryptographic Operation (Password-Based Key Derivation Functions)

TSS
The evaluator shall review the TSS to verify that it contains a description of the PBKDF. The evaluator shall also confirm the ST supports the selected hash function itself. The evaluator shall confirm that the TSS contains a description of how the TOE ensures that the output of the PBKDF is at least the same length as that specified in FCS_CKM.1/SK and for the KeyDrv4, KeyDrv5, or KeyDrv6 in FCS_CKM_EXT.5.

If the ST performs additional conditioning, whitening, or manipulation of the password or passphrase before applying the PBKDF, or to the output of the PBKDF, the evaluator shall ensure that the TSS describes the actions and provides assurance that the TSF does not negatively impact the entropy of the PBKDF output.

If any manipulation of the key is performed in forming the submask that will be used to form the KEK, that process shall be described in the TSS.
Guidance
There are no AGD evaluation activities for this component.
KMD
There are no KMD evaluation activities for this component.
Tests
No explicit testing of the formation of the submask from the input password is required.

For the NIST SP 800-132-based conditioning of the passphrase, the required evaluation activities will be performed when doing the evaluation activities for the appropriate requirements (FCS_COP.1/HMAC).

The evaluator shall verify that the iteration count for PBKDFs performed by the TOE comply with NIST SP 800-132 by ensuring that the TSS contains a description of the estimated time required to derive key material from passwords and how the TOE increases the computation time for password-based key derivation (including but not limited to increasing the iteration count).

FCS_COP.1/SigGen Cryptographic Operation (Signature Generation)

TSS
The evaluator shall examine the TSS to ensure that all signature generation functions use the approved algorithms and key sizes.
Guidance
There are no AGD evaluation activities for this component.
KMD
There are no KMD evaluation activities for this component.
Tests
The following tests require the developer to provide access to a test platform that provides the evaluator with tools that are typically not found on factory products.

Each section below contains tests the evaluators must perform for each selected digital signature scheme. Based on the assignments and selections in the requirement, the evaluators choose the specific activities that correspond to those selections.

The following tests require the developer to provide access to a test platform that provides the evaluator with tools that are not found on the TOE in its evaluated configuration.

If SigGen1: RSASSA-PKCS1-v1_5or SigGen4: RSASSA-PSS is claimed:

The below test is derived from The 186-4 RSA Validation System (RSA2VS). Updated 8 July 2014, Section 6.3, from the National Institute of Standards and Technology.

To test the implementation of RSA signature generation the evaluator uses the system under test to generate signatures for 10 messages for each combination of modulus size and SHA algorithm. The evaluator then uses a known-good implementation and the associated public keys to verify the signatures.

If SigGen2: Digital Signature Scheme 2 (DSS2) or SigGen3: Digital Signature Scheme 3 (DSS3):

To test the implementation of DSS2/3 signature generation the evaluator uses the system under test to generate signatures for 10 messages for each combination of SHA algorithm, hash size and key size. The evaluator them uses a known-good implementation and the associated public keys to verify the signatures.

If SigGen5: ECDSA is claimed:

The below test is derived from The FIPS 186-4 Elliptic Curve Digital Signature Algorithm Validation System (ECDSA2VS). Updated 18 March 2014, Section 6.4, from the National Institute of Standards and Technology.

To test the implementation of ECDSA signature generation the evaluator uses the system under test to generate signatures for 10 messages for each combination of curve, SHA algorithm, hash size, and key size. The evaluator then uses a known-good implementation and the associated public keys to verify the signatures.

FCS_COP.1/SigVer Cryptographic Operation (Signature Verification)

TSS
The evaluator shall check the TSS to ensure that it describes the overall flow of the signature verification. This should at least include identification of the format and general location (e.g., "firmware on the hard drive device" rather than "memory location 0x00007A4B") of the data to be used in verifying the digital signature; how the data received from the operational environment are brought onto the device; and any processing that is performed that is not part of the digital signature algorithm (for instance, checking of certificate revocation lists).
Guidance
There are no AGD evaluation activities for this component.
KMD
There are no KMD evaluation activities for this component.
Tests
The following tests require the developer to provide access to a test platform that provides the evaluator with tools that are typically not found on factory products.

Each section below contains tests the evaluators must perform for each selected digital signature scheme. Based on the assignments and selections in the requirement, the evaluators choose the specific activities that correspond to those selections.

The following tests require the developer to provide access to a test platform that provides the evaluator with tools that are not found on the TOE in its evaluated configuration.

SigVer1: RSASSA-PKCS1-v1_5 and SigVer4: RSASSA-PSS

These tests are derived from The 186-4 RSA Validation System (RSA2VS), updated 8 Jul 2014, Section 6.4.

The FIPS 186-4 RSA Signature Verification Test tests the ability of the TSF to recognize valid and invalid signatures. The evaluator shall provide a modulus and three associated key pairs (d, e) for each combination of selected SHA algorithm, modulus size and hash size. Each private key d is used to sign six pseudorandom messages each of 1024 bits. For five of the six messages, the public key (e), message, IR format, padding, or signature is altered so that signature verification should fail. The test passes only if all the signatures made using unaltered parameters result in successful signature verification, and all the signatures made using altered parameters result in unsuccessful signature verification.

SigVer5: ECDSA on NIST and Brainpool Curves

These tests are derived from The FIPS 186-4 Elliptic Curve Digital Signature Algorithm Validation System (ECDSA2VS), updated 18 Mar 2014, Section 6.5.

The FIPS 186-4 ECC Signature Verification Test tests the ability of the TSF to recognize valid and invalid signatures. The evaluator shall provide a modulus and associated key pair (x, y) for each combination of selected curve, SHA algorithm, modulus size, and hash size. Each private key (x) is used to sign 15 pseudorandom messages of 1024 bits. For eight of the fifteen messages, the message, IR format, padding, or signature is altered so that signature verification should fail. The test passes only if all the signatures made using unaltered parameters result in successful signature verification, and all the signatures made using altered parameters result in unsuccessful signature verification.

SigVer2: Digital Signature Scheme 2

The following or equivalent steps shall be taken to test the TSF.

For each supported modulus size, underlying hash algorithm, and length of the trailer field (1- or 2-byte), the evaluator shall generate NT sets of recoverable message (M1), non-recoverable message (M2), salt, public key and signature (Σ).

1. N_T shall be greater than or equal to 20.
2. The length of salts shall be selected from its supported length range of salt. The typical length of salt is

equal to the output block length of underlying hash algorithm (see 9.2.2 of ISO/IEC 9796-2:2010).

3. The length of recoverable messages should be selected by considering modulus size, output block length of underlying hash algorithm, and length of salt (L_S). As described in Annex D of ISO/IEC 9796-2:2010, it is desirable to maximise the length of recoverable message. The following table shows the maximum bit-length of recoverable message that is divisible by 512, for some combinations of modulus size, underlying hash algorithm, and length of salt. None that 2-byte trailer field is assumed in calculating the maximum length of recoverable message

Maximum length of recoverable message divisible by 512 (bits)	Modulus size (bits)	Underlying hash algorithm (bits)	Length of salt L_S (bits)
1536	2048	SHA-256	128
1024		SHA-256	256
1024		SHA-512	128
1024		SHA-512	256
512		SHA-512	512
2560	3072	SHA-256	128
2048		SHA-256	256
2048		SHA-512	128
2048		SHA-512	256
1536		SHA-512	512

4. The length of non-recoverable messages should be selected by considering the underlying hash algorithm and usages. If the TSF is used for verifying the authenticity of software/firmware updates, the length of non-recoverable messages should be selected greater than or equal to 2048-bit. With this length range, it means that the underlying hash algorithm is also tested for two or more input blocks.

5. The evaluator shall select approximately one half of N_T sets and shall alter one of the values (non-recoverable message, public key exponent or signature) in the sets. In altering public key exponent, the evaluator shall alter the public key exponent while keeping the exponent odd. In altering signatures, the following ways should be considered:
    a. Altering a signature just by replacing a bit in the bit-string representation of the signature
    b. Altering a signature so that the trailer in the message representative cannot be interpreted. This can be achieved by following ways:
        - Setting the rightmost four bits of the message representative to the values other than '1100'.
        - In the case when 1-byte trailer is used, setting the rightmost byte of the message representative to the values other than '0xbc', while keeping the rightmost four bits to '1100'.
        - In the case when 2-byte trailer is used, setting the rightmost byte of the message representative to the values other than '0xcc', while keeping the rightmost four bits to '1100'.
    c. In the case when 2-byte trailer is used, altering a signature so that the hash algorithm identifier in the trailer (i.e. the left most byte of the trailer) does not correspond to hash algorithms identified in the SFR. The hash algorithm identifiers are 0x34 for SHA-256 (see Clause 10 of ISO/IEC 10118-3:2018), and 0x35 for SHA-512 (see Clause 11 of ISO/IEC 10118-3:2018).
    d. Let L_S be the length of salt, altering a signature so that the intermediate bit string D in the message representative is set to all zeroes except for the rightmost L_S bits of D.
    e. (non-conformant signature length) Altering a signature so that the length of signature Σ is changed to modulus size and the most significant bit of signature Σ is set equal to '1'.
    f. (non-conformant signature) Altering a signature so that the integer converted from signature Σ is greater than modulus n.

The evaluator shall supply the NT sets to the TSF and obtain in response a set of NT Verification-Success or Verification-Fail values. When the VerificationSuccess is obtained, the evaluator shall also obtain recovered message (M 1*).

The evaluator shall verify that Verification-Success results correspond to the unaltered sets and Verification-Fail results correspond to the altered sets.

For each recovered message, the evaluator shall compare the recovered message (M1*) with the corresponding recoverable message (M 1) in the unaltered sets.

The test passes only if all the signatures made using unaltered sets result in Verification-Success, each recovered message (M 1*) is equal to corresponding M 1 in the unaltered sets, and all the signatures made using altered sets result in Verification-Fail.

SigVer3: Digital Signature Scheme 3

The evaluator shall perform the test described in SigVer2: Digital Signature Scheme 2 while using a fixed salt for NT sets.

FCS_COP.1/SKC Cryptographic Operation (Symmetric Key Cryptography)

TSS
The evaluator shall check that the TSS includes a description of encryption functions used for symmetric key encryption. The evaluator should check that this description of the selected encryption function includes the key sizes and modes of operations as specified in the cPP table 9 "Supported Methods for Symmetric Key Cryptography Operation."

The evaluator shall check that the TSS describes the means by which the TOE satisfies constraints on algorithm parameters included in the selections made for 'cryptographic algorithm' and 'list of standards'.

Guidance
If the product supports multiple modes, the evaluator shall examine the vendor's documentation to determine that the method of choosing a specific mode/key size by the end user is described.

KMD
The evaluator shall examine the KMD to ensure that the points at which symmetric key encryption and decryption occurs are described, and that the complete data path for symmetric key encryption is described. The evaluator checks that this description is consistent with the relevant parts of the TSS.

Assessment of the complete data path for symmetric key encryption includes confirming that the KMD describes the data flow from the device's host interface to the device's non-volatile memory storing the data, and gives information enabling the user data datapath to be distinguished from those situations in which data bypasses the data encryption engine (e.g. read-write operations to an unencrypted Master Boot Record area). The evaluator shall ensure that the documentation of the data path is detailed enough that it thoroughly describes the parts of the TOE that the data passes through (e.g. different memory types, processors and co-processors), its encryption state (i.e. encrypted or unencrypted) in each part, and any places where the data is stored. For example, any caching or buffering of the data should be identified and distinguished from the final destination in non-volatile memory (the latter represents the location from which the host will expect to retrieve the data in future).

If support for AES-CTR is claimed and the counter value source is internal to the TOE, the evaluator shall verify that the KMD describes the internal counter mechanism used to ensure that it provides unique counter block values.

Tests
The following tests require the developer to provide access to a test platform that provides the evaluator with tools that are typically not found on factory products.

The following tests are conditional based upon the selections made in the SFR. The evaluator shall perform the following test or witness respective tests executed by the developer. The tests must be executed on a platform that is as close as practically possible to the operational platform (but which may be instrumented in terms of, for example, use of a debug mode). Where the test is not carried out on the TOE itself, the test platform shall be identified and the differences between test environment and TOE execution environment shall be described.

Preconditions for testing:

- Specification of keys as input parameter to the function to be tested
- specification of required input parameters such as modes
- Specification of user data (plaintext)
- Tapping of encrypted user data (ciphertext) directly in the non-volatile memory

AES-CBC:
For the AES-CBC tests described below, the plaintext, ciphertext, and IV values shall consist of 128-bit blocks. To determine correctness, the evaluator shall compare the resulting values to those obtained by submitting the same inputs to a known-good implementation.

These tests are intended to be equivalent to those described in NIST's AES Algorithm Validation Suite (AESAVS) (http://csrc.nist.gov/groups/STM/cavp/documents/aes/AESAVS.pdf). It is not recommended that evaluators use values obtained from static sources such as the example NIST's AES Known Answer Test Values from the AESAVS document, or use values not generated expressly to exercise the AES-CBC implementation.

AES-CBC Known Answer Tests

KAT-1 (GFSBox): To test the encrypt functionality of AES-CBC, the evaluator shall supply a set of five different plaintext values for each selected key size and obtain the ciphertext value that results from AES-CBC encryption of the given plaintext using a key value of all zeros and an IV of all zeros.

To test the decrypt functionality of AES-CBC, the evaluator shall supply a set of five different ciphertext values for each selected key size and obtain the plaintext value that results from AES-CBC decryption of the given ciphertext using a key value of all zeros and an IV of all zeros.

KAT-2 (KeySBox): To test the encrypt functionality of AES-CBC, the evaluator shall supply a set of five different key values for each selected key size and obtain the ciphertext value that results from AES-CBC encryption of an all-zeros plaintext using the given key value and an IV of all zeros.

To test the decrypt functionality of AES-CBC, the evaluator shall supply a set of five different key values for each selected key size and obtain the plaintext that results from AES-CBC decryption of an all-zeros ciphertext using the given key and an IV of all zeros.

KAT-3 (Variable Key): To test the encrypt functionality of AES-CBC, the evaluator shall supply a set of keys for each selected key size (as described below) and obtain the ciphertext value that results from AES encryption of an all-zeros plaintext using each key and an IV of all zeros.

Key i in each set shall have the leftmost i bits set to ones and the remaining bits to zeros, for values of i from 1 to the key size. The keys and corresponding ciphertext are listed in AESAVS, Appendix E.

To test the decrypt functionality of AES-CBC, the evaluator shall use the same keys as above to decrypt the ciphertext results from above. Each decryption should result in an all-zeros plaintext.

<u>KAT-4 (Variable Text):</u> To test the encrypt functionality of AES-CBC, for each selected key size, the evaluator shall supply a set of 128-bit plaintext values (as described below) and obtain the ciphertext values that result from AES-CBC encryption of each plaintext value using a key of each size and IV consisting of all zeros.

Plaintext value i shall have the leftmost i bits set to ones and the remaining bits set to zeros, for values of i from 1 to 128. The plaintext values are listed in AESAVS, Appendix D.

To test the decrypt functionality of AES-CBC, for each selected key size, use the plaintext values from above as ciphertext input, and AES-CBC decrypt each ciphertext value using key of each size consisting of all zeros and an IV of all zeros.

AES-CBC Multi-Block Message Test

The evaluator shall test the encrypt functionality by encrypting nine i-block messages for each selected key size, for $2 \leq i \leq 10$. For each test, the evaluator shall supply a key, an IV, and a plaintext message of length i blocks, and encrypt the message using AES-CBC. The resulting ciphertext values shall be compared to the results of encrypting the plaintext messages using a known good implementation.

The evaluator shall test the decrypt functionality by decrypting nine i-block messages for each selected key size, for $2 \leq i \leq 10$. For each test, the evaluator shall supply a key, an IV, and a ciphertext message of length i blocks, and decrypt the message using AES-CBC. The resulting plaintext values shall be compared to the results of decrypting the ciphertext messages using a known good implementation.

AES-CBC Monte Carlo Tests

The evaluator shall test the encrypt functionality for each selected key size using 100 3-tuples of pseudo-random values for plaintext, IVs, and keys.

The evaluator shall supply a single 3-tuple of pseudo-random values for each selected key size. This 3-tuple of plaintext, IV, and key is provided as input to the below algorithm to generate the remaining 99 3-tuples, and to run each 3-tuple through 1000 iterations of AES-CBC encryption.

```
# Input: PT, IV, Key
Key[0] = Key
IV[0] = IV
PT[0] = PT
for i = 0 to 99 {
        Output Key[i], IV[i], PT[0]
        for j = 0 to 999 {
                if (j == 0) {
                        CT[j] = AES-CBC-Encrypt(Key[i], IV[i], PT[j])
                        PT[j+1] = IV[i]
                } else {
                        CT[j] = AES-CBC-Encrypt(Key[i], PT[j])
                        PT[j+1] = CT[j-1]
                }
        }
        Output CT[j]
        If (KeySize == 128) Key[i+1] = Key[i] xor CT[j]
        If (KeySize == 192) Key[i+1] = Key[i] xor (last 64 bits of CT[j-1] || C
        If (KeySize == 256) Key[i+1] = Key[i] xor ((CT[j-1] | CT[j])
        IV[i+1] = CT[j]
        PT[0] = CT[j-1]
}
```

The ciphertext computed in the 1000th iteration (CT[999]) is the result for each of the 100 3-tuples for each selected key size. This result shall be compared to the result of running 1000 iterations with the same values using a known good implementation.

The evaluator shall test the decrypt functionality using the same test as above, exchanging CT and PT, and replacing AES-CBC-Encrypt with AES-CBC-Decrypt.

AES-CCM:

These tests are intended to be equivalent to those described in the NIST document, "The CCM Validation System (CCMVS)," updated 9 Jan 2012, found at
http://csrc.nist.gov/groups/STM/cavp/documents/mac/CCMVS.pdf.

It is not recommended that evaluators use values obtained from static sources such as http://csrc.nist.gov/groups/STM/cavp/documents/mac/ccmtestvectors.zip or use values not generated expressly to exercise the AES-CCM implementation.

The evaluator shall test the generation-encryption and decryption-verification functionality of AES-CCM for the following input parameter and tag lengths:

- **Keys:** All supported and selected key sizes (e.g., 128, 192, or 256 bits).
- **Associated Data:** Two or three values for associated data length: The minimum (≥ 0 bytes) and maximum (≤ 32 bytes) supported associated data lengths, and 2^{16} (65536) bytes, if supported.
- **Payload:** Two values for payload length: The minimum (≥ 0 bytes) and maximum (≤ 32 bytes) supported payload lengths.
- **Nonces:** All supported nonce lengths (e.g., 8, 9, 10, 11, 12, 13) in bytes.
- **Tag:** All supported tag lengths (e.g., 4, 6, 8, 10, 12, 14, 16) in bytes.

The testing for CCM consists of five tests. To determine correctness in each of the below tests, the evaluator shall compare the ciphertext with the result of encryption of the same inputs with a known good

implementation.

Variable Associated Data Test: For each supported key size and associated data length, and any supported payload length, nonce length, and tag length, the evaluator shall supply one key value, one nonce value, and 10 pairs of associated data and payload values, and obtain the resulting ciphertext.

Variable Payload Text: For each supported key size and payload length, and any supported associated data length, nonce length, and tag length, the evaluator shall supply one key value, one nonce value, and 10 pairs of associated data and payload values, and obtain the resulting ciphertext.

Variable Nonce Test: For each supported key size and nonce length, and any supported associated data length, payload length, and tag length, the evaluator shall supply one key value, one nonce value, and 10 pairs of associated data and payload values, and obtain the resulting ciphertext.

Variable Tag Test: For each supported key size and tag length, and any supported associated data length, payload length, and nonce length, the evaluator shall supply one key value, one nonce value, and 10 pairs of associated data and payload values, and obtain the resulting ciphertext.

Decryption-Verification Process Test: To test the decryption-verification functionality of AES-CCM, for each combination of supported associated data length, payload length, nonce length, and tag length, the evaluator shall supply a key value and 15 sets of input plus ciphertext, and obtain the decrypted payload. Ten of the 15 input sets supplied should fail verification and five should pass.

AES-GCM: These tests are intended to be equivalent to those described in the NIST document, "The Galois/Counter Mode (GCM) and GMAC Validation System (GCMVS) with the Addition of XPN Validation Testing," rev. 15 Jun 2016, section 6.2, found at
http://csrc.nist.gov/groups/STM/cavp/documents/mac/gcmvs.pdf.

It is not recommended that evaluators use values obtained from static sources such as http://csrc.nist.gov/groups/STM/cavp/documents/mac/gcmtestvectors.zip, or use values not generated expressly to exercise the AES-GCM implementation.

The evaluator shall test the authenticated encryption functionality of AES-GCM by supplying 15 sets of Key, Plaintext, AAD, IV, and Tag data for every combination of the following parameters as selected in the ST and supported by the implementation under test:

- **Key size in bits:** Each selected and supported key size (e.g., 128, 192, or 256 bits).
- **Plaintext length in bits:** Up to four values for plaintext length: Two values that are non-zero integer multiples of 128, if supported. And two values that are non-multiples of 128, if supported.
- **AAD length in bits:** Up to five values for AAD length: Zero-length, if supported. Two values that are non-zero integer multiples of 128, if supported. And two values that are integer non-multiples of 128, if supported.
- **IV length in bits:** Up to three values for IV length: 96 bits. Minimum and maximum supported lengths, if different.
- **MAC length in bits:** Each supported length (e.g., 128, 120, 112, 104, 96).

To determine correctness, the evaluator shall compare the resulting values to those obtained by submitting the same inputs to a known-good implementation.

The evaluator shall test the authenticated decrypt functionality of AES-GCM by supplying 15 Ciphertext-Tag pairs for every combination of the above parameters, replacing Plaintext length with Ciphertext length. For each parameter combination the evaluator shall introduce an error into either the Ciphertext or the Tag such that approximately half of the cases are correct and half the cases contain errors. To determine correctness, the evaluator shall compare the resulting pass/fail status and Plaintext values to the results obtained by submitting the same inputs to a known-good implementation.

AES-CTR:

For the AES-CTR tests described below, the plaintext and ciphertext values shall consist of 128-bit blocks. To determine correctness, the evaluator shall compare the resulting values to those obtained by submitting the same inputs to a known-good implementation.

These tests are intended to be equivalent to those described in NIST's AES Algorithm Validation Suite (AESAVS) (http://csrc.nist.gov/groups/STM/cavp/documents/aes/AESAVS.pdf). It is not recommended that evaluators use values obtained from static sources such as the example NIST's AES Known Answer Test Values from the AESAVS document, or use values not generated expressly to exercise the AES-CTR implementation.

AES-CTR Known Answer Tests

KAT-1 (GFSBox): To test the encrypt functionality of AES-CTR, the evaluator shall supply a set of five different plaintext values for each selected key size and obtain the ciphertext value that results from AES-CTR encryption of the given plaintext using a key value of all zeros.

To test the decrypt functionality of AES-CTR, the evaluator shall supply a set of five different ciphertext values for each selected key size and obtain the plaintext value that results from AES-CTR decryption of the given ciphertext using a key value of all zeros.

KAT-2 (KeySBox): To test the encrypt functionality of AES-CTR, the evaluator shall supply a set of five different key values for each selected key size and obtain the ciphertext value that results from AES-CTR encryption of an all-zeros plaintext using the given key value.

To test the decrypt functionality of AES-CTR, the evaluator shall supply a set of five different key values for

each selected key size and obtain the plaintext that results from AES-CTR decryption of an all-zeros ciphertext using the given key.

KAT-3 (Variable Key): To test the encrypt functionality of AES-CTR, the evaluator shall supply a set of keys for each selected key size (as described below) and obtain the ciphertext value that results from AES encryption of an all-zeros plaintext using each key.

Key i in each set shall have the leftmost i bits set to ones and the remaining bits to zeros, for values of i from 1 to the key size. The keys and corresponding ciphertext are listed in AESAVS, Appendix E.

To test the decrypt functionality of AES-CTR, the evaluator shall use the same keys as above to decrypt the ciphertext results from above. Each decryption should result in an all-zeros plaintext.

KAT-4 (Variable Text): To test the encrypt functionality of AES-CTR, for each selected key size, the evaluator shall supply a set of 128-bit plaintext values (as described below) and obtain the ciphertext values that result from AES-CTR encryption of each plaintext value using a key of each size.

Plaintext value i shall have the leftmost i bits set to ones and the remaining bits set to zeros, for values of i from 1 to 128. The plaintext values are listed in AESAVS, Appendix D.

To test the decrypt functionality of AES-CTR, for each selected key size, use the plaintext values from above as ciphertext input, and AES-CTR decrypt each ciphertext value using key of each size consisting of all zeros.

AES-CTR Multi-Block Message Test

The evaluator shall test the encrypt functionality by encrypting nine i-block messages for each selected key size, for $2 \leq i \leq 10$. For each test, the evaluator shall supply a key and a plaintext message of length i blocks, and encrypt the message using AES-CTR. The resulting ciphertext values shall be compared to the results of encrypting the plaintext messages using a known good implementation.

The evaluator shall test the decrypt functionality by decrypting nine i-block messages for each selected key size, for $2 \leq i \leq 10$. For each test, the evaluator shall supply a key and a ciphertext message of length i blocks, and decrypt the message using AES-CTR. The resulting plaintext values shall be compared to the results of decrypting the ciphertext messages using a known good implementation.

AES-CTR Monte Carlo Tests

The evaluator shall test the encrypt functionality for each selected key size using 100 2-tuples of pseudo-random values for plaintext and keys.

The evaluator shall supply a single 2-tuple of pseudo-random values for each selected key size. This 2-tuple of plaintext and key is provided as input to the below algorithm to generate the remaining 99 2-tuples, and to run each 2-tuple through 1000 iterations of AES-CTR encryption.

```
# Input: PT, Key
Key[0] = Key
PT[0] = PT
for i = 0 to 99 {
        Output Key[i], PT[0]
        for j = 0 to 999 {
                CT[j] = AES-CTR-Encrypt(Key[i], PT[j])
                PT[j+1] = CT[j]
        }
        Output CT[j]
        If (KeySize == 128) Key[i+1] = Key[i] xor CT[j]
        If (KeySize == 192) Key[i+1] = Key[i] xor (last 64 bits of CT[j-1] || C
        If (KeySize == 256) Key[i+1] = Key[i] xor ((CT[j-1] | CT[j])
        PT[0] = CT[j]
}
```

The ciphertext computed in the 1000th iteration (CT[999]) is the result for each of the 100 2-tuples for each selected key size. This result shall be compared to the result of running 1000 iterations with the same values using a known good implementation.

The evaluator shall test the decrypt functionality using the same test as above, exchanging CT and PT, and replacing AES-CTR-Encrypt with AES-CTR-Decrypt. 198 Note additional design considerations for this mode are addressed in the KMD requirements.

XTS-AES: These tests are intended to be equivalent to those described in the NIST document, "The XTS-AES Validation System (XTSVS)," updated 5 Sept 2013, found at
http://csrc.nist.gov/groups/STM/cavp/documents/aes/XTSVS.pdf

It is not recommended that evaluators use values obtained from static sources such as the XTS-AES test vectors at http://csrc.nist.gov/groups/STM/cavp/documents/aes/XTSTestVectors.zip or use values not generated expressly to exercise the XTS-AES implementation.

The evaluator shall generate test values as follows:

For each supported key size (256 bit (for AES-128) and 512 bit (for AES-256) keys), the evaluator shall provide up to five data lengths:

- Two data lengths divisible by the 128-bit block size, If data unit lengths of complete block sizes are supported.
- Two data lengths not divisible by the 128-bit block size, if data unit lengths of partial block sizes are supported.

- The largest data length supported by the implementation, or 2^16 (65536), whichever is larger.

The evaluator shall specify whether the implementation supports tweak values of 128-bit hexadecimal strings or a data unit sequence numbers, or both.

For each combination of key size and data length, the evaluator shall provide 100 sets of input data and obtain the ciphertext that results from XTS-AES encryption. If both kinds of tweak values are supported then each type of tweak value shall be used in half of every 100 sets of input data, for all combinations of key size and data length. The evaluator shall verify that the resulting ciphertext matches the results from submitting the same inputs to a known-good implementation of XTS-AES.

The evaluator shall test the decrypt functionality of XTS-AES using the same test as for encrypt, replacing plaintext values with ciphertext values and XTS-AES encrypt with XTS- AES decrypt.

The evaluator shall check that the full length keys are created by methods that ensure that the two halves are different and independent.

AES-KWP:

The tests below are derived from "The Key Wrap Validation System (KWVS), Updated: June 20, 2014" from the National Institute of Standards and Technology.

The evaluator shall test the authenticated-encryption functionality of AES-KWP (KWP-AE) using the same test as for AES-KW authenticated-encryption with the following change in the five plaintext lengths:

- Four lengths that are multiples of 8 bits
- The largest supported length less than or equal to 4096 bits.

The evaluator shall test the authenticated-decryption (KWP-AD) functionality of AES-KWP using the same test as for AES-KWP authenticated-encryption, replacing plaintext values with ciphertext values and AES-KWP authenticatedencryption with AES-KWP authenticated-decryption. For the Authenticated Decryption test, 20 out of the 100 trials per plaintext length have ciphertext values that fail authentication.

Additionally, the evaluator shall perform the following negative tests:

Test 1: (invalid plaintext length):

Determine the valid plaintext lengths of the implementation from the TOE specification. Verify that the implementation of KWP-AE in the TOE rejects plaintexts of invalid length by testing plaintext of the following lengths: 1) plaintext with length greater than 64 semi-blocks, 2) plaintext with bit-length not divisible by 8, and 3) plaintext with length 0.

Test 2: (invalid ciphertext length): Determine the valid ciphertext lengths of the implementation from the TOE specification. Verify that the implementation of KWP-AD in the TOE rejects ciphertexts of invalid length by testing ciphertext of the following lengths: 1) ciphertext with length greater than 65 semi-blocks, 2) ciphertext with bit-length not divisible by 64, 3) ciphertext with length 0, and 4) ciphertext with length of one semi-block.

Test 3: (invalid ICV2): Test that the implementation detects invalid ICV2 values by encrypting any plaintext value four times using a different value for ICV2 each time as follows: Start with a base ICV2 of 0xA65959A6. For each of the four tests change a different byte of ICV2 to a different value, so that each of the four bytes is changed once. Verify that the implementation of KWP-AD in the TOE outputs FAIL for each test.

Test 4: (invalid padding length): Generate one ciphertext using algorithm KWP-AE with substring [len(P)/8]32 of S replaced by each of the following 32-bit values, where len(P) is the length of P in bits and [ ]32 denotes the representation of an integer in 32 bits:

- [0]32
- [len(P)/8-8]32
- [len(P)/8+8]32
- [513]32.

Verify that the implementation of KWP-AD in the TOE outputs FAIL on those inputs.

Test 5: (invalid padding bits):

If the implementation supports plaintext of length not a multiple of 64-bits, then

```
for each PAD length [1..7]
        for each byte in PAD set a zero PAD value;
                replace current byte by a non-zero value and use the resulting plaintex
                        input to algorithm KWP-AE to generate ciphertexts;
                verify that the implementation of KWP-AD in the TOE outputs FAIL on
                        this input.
```

AES-KW:

The tests below are derived from "The Key Wrap Validation System (KWVS), Updated: June 20, 2014" from the National Institute of Standards and Technology.

The evaluator shall test the authenticated-encryption functionality of AES-KW for each combination of the following input parameters:

- Supported key lengths selected in the ST (e.g. 128 bits, 256 bits)

- Five plaintext lengths:
  - Two lengths that are non-zero multiples of 128 bits (two semi-block lengths)
  - Two lengths that are odd multiples of the semi-block length (64 bits)
  - The largest supported plaintext length less than or equal to 4096 bits.

For each set of the above parameters the evaluator shall generate a set of 100 key and plaintext pairs and obtain the ciphertext that results from AES-KW authenticated encryption. To determine correctness, the evaluator shall compare the results with those obtained from the AES-KW authenticated-encryption function of a known good implementation.

The evaluator shall test the authenticated-decryption functionality of AES-KW using the same test as for authenticated-encryption, replacing plaintext values with ciphertext values and AES-KW authenticated-encryption (KW-AE) with AES-KW authenticated-decryption (KW-AD). For the authenticated-decryption test, 20 out of the 100 trials per plaintext length must have ciphertext values that are not authentic; that is, they fail authentication.

Additionally, the evaluator shall perform the following negative tests:

Test 1 (invalid plaintext length):

Determine the valid plaintext lengths of the implementation from the TOE specification. Verify that the implementation of KW-AE in the TOE rejects plaintexts of invalid length by testing plaintext of the following lengths: 1) plaintext length greater than 64 semi-blocks, 2) plaintext bit-length not divisible by 64, 3) plaintext with length 0, and 4) plaintext with one semi-block.

Test 2 (invalid ciphertext length):

Determine the valid ciphertext lengths of the implementation from the TOE specification. Verify that the implementation of KW-AD in the TOE rejects ciphertexts of invalid length by testing ciphertext of the following lengths: 1) ciphertext with length greater than 65 semi-blocks, 2) ciphertext with bit-length not divisible by 64, 3) ciphertext with length 0, 4) ciphertext with length of one semiblock, and 5) ciphertext with length of two semi-blocks.

Test 3 (invalid ICV1):

222 Test that the implementation detects invalid ICV1 values by encrypting any plaintext value eight times using a different value for ICV1 each time as follows: Start with a base ICV1 of 0xA6A6A6A6A6A6A6A6. For each of the eight tests change a different byte to a different value, so that each of the eight bytes is changed once. Verify that the implementation of KW-AD in the TOE outputs FAIL for each test.

CAM-CBC:

To test the encrypt and decrypt functionality of Camellia in CBC mode, the evaluator shall perform the tests as specified in 10.2.1.2 of ISO/IEC 18367:2016.

CAM-CCM:

To test the encrypt functionality of Camellia in CCM mode, the evaluator shall perform the tests as specified in 10.6.1.1 of ISO/IEC 18367:2016.

To test the decrypt functionality of Camellia in CCM mode, the evaluator shall perform the tests as specified in 10.6.1.2 of ISO/IEC 18367:2016.

As a prerequisite for these tests, the evaluator shall perform the test for encrypt functionality of Camellia in ECB mode as specified in 10.2.1.2 of ISO/IEC 18367:2016.

CAM-GCM:

To test the encrypt functionality of Camellia in GCM, the evaluator shall perform the tests as specified in 10.6.1.1 of ISO/IEC 18367:2016.

To test the decrypt functionality of Camellia in GCM, the evaluator shall perform the tests as specified in 10.6.1.2 of ISO/IEC 18367:2016.

As a prerequisite for these tests, the evaluator shall perform the test for encrypt functionality of Camellia in ECB mode as specified in 10.2.1.2 of ISO/IEC 18367:2016.

XTS-CAM:

These tests are intended to be equivalent to those described in the IPA document, ATR-01-B, "Specifications of Cryptographic Algorithm Implementation Testing — Symmetric-Key Cryptography", found at https://www.ipa.go.jp/security/jcmvp/jcmvp_e/documents/atr/atr01b_en.pdf.

The evaluator shall generate test values as follows:

For each supported key size (256 bit (for Camellia-128) and 512 bit (for Camellia256) keys), the evaluator shall provide up to five data lengths:

- Two data lengths divisible by the 128-bit block size, If data unit lengths of complete block sizes are supported.
- Two data lengths not divisible by the 128-bit block size, if data unit lengths of partial block sizes are supported.
- The largest data length supported by the implementation, or 2^{16} (65536), whichever is larger.

The evaluator shall specify whether the implementation supports tweak values of 128-bit hexadecimal strings or a data unit sequence numbers, or both.

For each combination of key size and data length, the evaluator shall provide 100 sets of input data and obtain the ciphertext that results from XTS-Camellia encryption. If both kinds of tweak values are supported, 50 of each 100 sets of input data shall use each type of tweak value. The resulting ciphertext shall be compared to the results of a known-good implementation.

As a prerequisite for this test, the evaluator shall perform the test for encrypt functionality of Camellia in ECB mode as specified in 10.2.1.2 of ISO/IEC 18367:2016.

The evaluator shall test the decrypt functionality of XTS-Camellia using the same test as for encrypt, replacing plaintext values with ciphertext values and XTSCamellia encrypt with XTS- Camellia decrypt.

As a prerequisite for this test, the evaluator shall perform the test for decrypt functionality of Camellia in ECB mode as specified in 10.2.1.2 of ISO/IEC 18367:2016.

FCS_RBG_EXT.1 Random Bit Generation

TSS
The evaluator shall examine the TSS to determine that it specifies the DRBG type, identifies the entropy sources seeding the DRBG, and state the assumed or calculated min-entropy supplied either separately by each source or the min-entropy contained in the combined seed value.

In addition to the materials below, documentation shall be produced—and the evaluator shall perform the activities—in accordance with Appendix D of [DSCcPP].

Guidance
There are no AGD evaluation activities for this component.
KMD
There are no KMD evaluation activities for this component.
Tests
The following tests require the developer to provide access to a test platform that provides the evaluator with tools that are typically not found on factory products.

The evaluator shall perform 15 trials for the RNG implementation. If the RNG is configurable, the evaluator shall perform 15 trials for each configuration.

If the RNG has prediction resistance enabled, each trial consists of (1) instantiate DRBG, (2) generate the first block of random bits (3) generate a second block of random bits (4) uninstantiate. The evaluator verifies that the second block of random bits is the expected value. The evaluator shall generate eight input values for each trial. The first is a count (0 – 14). The next three are entropy input, nonce, and personalization string for the instantiate operation. The next two are additional input and entropy input for the first call to generate. The final two are additional input and entropy input for the second call to generate. These values are randomly generated. "generate one block of random bits" means to generate random bits with number of returned bits equal to the Output Block Length (as defined in NIST SP800-90A).

If the RNG does not have prediction resistance, each trial consists of (1) instantiate DRBG, (2) generate the first block of random bits (3) reseed, (4) generate a second block of random bits (5) uninstantiate. The evaluator verifies that the second block of random bits is the expected value. The evaluator shall generate eight input values for each trial. The first is a count (0 – 14). The next three are entropy input, nonce, and personalization string for the instantiate operation. The fifth value is additional input to the first call to generate. The sixth and seventh are additional input and entropy input to the call to reseed. The final value is additional input to the second generate call.

The following paragraphs contain more information on some of the input values to be generated/selected by the evaluator.

- **Entropy input:** the length of the entropy input value must equal the seed length.
- **Nonce:** If a nonce is supported (CTR_DRBG with no Derivation Function does not use a nonce), the nonce bit length is one-half the seed length.
- **Personalization string:** The length of the personalization string must be ≤ seed length. If the implementation only supports one personalization string length, then the same length can be used for both values. If more than one string length is support, the evaluator shall use personalization strings of two different lengths. If the implementation does not use a personalization string, no value needs to be supplied.
- **Additional input:** the additional input bit lengths have the same defaults and restrictions as the personalization string lengths.

FCS_SLT_EXT.1 Cryptographic Salt Generation

TSS
The evaluator shall ensure the TSS describes how salts are generated using the RBG.
Guidance
There are no AGD evaluation activities for this component.
KMD
There are no KMD evaluation activities for this component.
Tests
The evaluator shall confirm by testing that the salts obtained in the cryptographic operations that use the salts are of the length specified in FCS_SLT_EXT.1, are obtained from the RBG, and are fresh on each invocation.

Note: in general these tests may be carried out as part of the tests of the relevant cryptographic operations.

FCS_STG_EXT.1 Protected Storage

TSS
The evaluator shall review the TSS to determine that the TOE implements the required protected storage. The evaluator shall ensure that the TSS contains a description of the protected storage mechanism that justifies the selection of mutable hardware-based or software-based.

Guidance
The evaluator shall examine the operational guidance to ensure that it describes the process for generating keys, importing keys, or both, based on what is claimed by the ST. The evaluator shall also examine the operational guidance to ensure that it describes the process for destroying keys that have been imported or generated.

KMD
There are no KMD evaluation activities for this component.
Tests
The evaluator shall test the functionality of each security function as described below. If the TOE supports both import and generation of keys, the evaluator shall repeat the testing as needed to demonstrate that the keys resulting from both operations are treated in the same manner. The devices used with the tooling may need to be non-production devices in order to enable the execution and gathering of evidence.

- **Test 1:** The evaluator shall import or generate keys/secrets of each supported type according to the operational guidance. The evaluator shall write, or the developer shall provide access to, an application that generates a key/secret of each supported type and calls the import functions. The evaluator shall verify that no errors occur during import.
- **Test 2:** The evaluator shall write, or the developer shall provide access to, an application that uses a generated or imported key/secret:
    - For RSA, the secret shall be used to sign data.
    - For ECDSA, the secret shall be used to sign data.
  The evaluator shall repeat this test with the application-imported or applicationgenerated keys/secrets and a different application's imported keys/secrets or generated keys/secrets. The evaluator shall verify that the TOE requires approval before allowing the application to use the key/secret imported or generated by the user or by a different application:
    - The evaluator shall deny the approvals to verify that the application is not able to use the key/secret as described.
    - The evaluator shall repeat the test, allowing the approvals to verify that the application is able to use the key/secret as described.
  If the ST author has selected common application developer, this test is performed by either using applications from different developers or appropriately (according to API documentation) not authorizing sharing.

- **Test 3:** The evaluator shall destroy keys/secrets of each supported type according to the operational guidance. The evaluator shall write, or the developer shall provide access to, an application that destroys an imported or generated key/secret. The evaluator shall repeat this test with the application-imported or applicationgenerated keys/secrets and a different application's imported or generated keys/secrets. The evaluator shall verify that the TOE requires approval before allowing the application to destroy the key/secret imported by the administrator or by a different application:
    - The evaluator shall deny the approvals and verify that the application is still able to use the key/secret as described.
    - The evaluator shall repeat the test, allowing the approvals and verifying that the application is no longer able to use the key/secret as described.
  If the ST author has selected common application developer, this test is performed by either using applications from different developers or appropriately (according to API documentation) not authorizing sharing.

FCS_STG_EXT.2 Key Storage Encryption

TSS
The evaluator shall review the TSS to determine that the TSS describes the protection of symmetric keys, KEKs, long-term trusted channel key material, and software-based key storage as claimed in FCS_STG_EXT.2.1.
Guidance
There are no AGD evaluation activities for this component.
KMD
There are no KMD evaluation activities for this component
Tests
There are no test evaluation activities for this component.

FCS_STG_EXT.3 Key Integrity Protection

TSS
The evaluator shall examine the TSS and ensure that it contains a description of how the TOE protects the integrity of its keys.
Guidance
There are no AGD evaluation activities for this component.
KMD
There are no KMD evaluation activities for this component.
Tests

There are no test evaluation activities for this component.

2.2.2 User Data Protection

FDP_ACC.1 Subset Access Control

TSS
The evaluator shall confirm that the TSS contain the access control policy implemented by the TOE. I.e., the ST author lists each object and identifies for each object, which operations the TSF permits for each subject (i.e. what can "admins" do vs "users").

Guidance
There are no guidance evaluation activities for this component.

KMD
There are no KMD evaluation activities for this component.

Tests
Testing for FDP_ACF includes testing this component.

FDP_ACF.1 Security Attribute Based Access Control

TSS
The evaluator shall examine the TSS to verify that it describes the policy rules for the Access Control SFP. Specifically, the evaluator should be able to identify, for any arbitrary subject-object-operation pairing, which of the following is true:

a. The subject can always perform the desired operation.
b. The subject can never perform the desired operation, either because they lack sufficient permission or because the TSF includes no interface to support the operation.
c. The subject can only perform the desired operation under certain conditions (which the evaluator shall verify are described in the TSS). For example, "the S.CA subject may only perform the OP.Destroy operation on an OB.SDO object if it was the subject that originally created or imported the SDO."
d. The subject can only perform the desired operation on one or more attributes of the object as opposed to the entire object itself (which the evaluator shall verify are identified in the TSS).
e. Whether the subject can perform the desired operation depends on TSF configuration (which the evaluator shall verify is described in the TSS as part of the evaluation of FMT_SMF.1).
f. Some combination of c, d, and e.

Given that this SFR requires a large number of potential subject-object-operation pairings to be identified, it is not the expectation that the TSS contain an exhaustive list of these pairings. It is possible that large numbers of pairings are addressed by blanket statements of policy rules, such as "the subjects S.DSC and S.CA are never able to perform any operation on the OB.AntiReplay object." For any rules that are not addressed in this manner, the evaluator shall verify the TSS includes sufficient data for the evaluator to determine how the TSF will evaluate the action. This can be presented in the form of a table, flowchart, list, or other manner that the ST author finds suitable.

Note that the DSC developer may not use the same terminology for its subjects, objects, and operations as the PP. If this is the case, the evaluator shall verify that the TSS includes a mapping that unambiguously shows how the vendor's preferred terminology corresponds to what the PP defines.

Guidance
For any access control policy enforcement behavior that is configurable, the evaluator shall ensure that the operational guidance describes how to perform the configuration, including any restrictions on permissible configurable settings.

KMD
There are no KMD evaluation activities for this component.

Tests
The following testing may require the TOE developer to make a test harness available to the evaluator that allows the evaluator to interface directly with the DSC. Due to the large volume of potential testing that this requires, this test may require the use of an automated script. If a test script is made available, the evaluator shall verify that it includes sufficient detail to validate the claims made in the TSS.

For each subject/object/operation/attribute combination, the evaluator shall attempt to perform the operation or determine that no interface is present to attempt the operation, consistent with the limitations described in the TSS.

For each case where an operation is always permitted or never permitted, both positive and negative testing will be conducted implicitly by attempting the operation with all possible subjects and determining that the intended results occur in each case.

For each case where the operation succeeds or fails based on the target object attribute, the evaluator shall ensure that both positive and negative testing is performed such that only the correct target attributes can be operated upon.

For each case where the operation succeeds or fails based on one or more specific conditions, the evaluator shall ensure that both positive and negative testing is performed such that the presence of the conditions causes the test to succeed while the absence of the conditions causes the test to fail.

For each case where the operation succeeds or fails based on an administratively configured setting, the evaluator shall ensure that both positive and negative testing is performed such that the configuration setting can be shown to affect whether or not the operation succeeds.

FDP_ETC_EXT.2 Propagation of SDOs

The evaluator shall examine the TSS to ensure that it describes how it protects the SDO references, authorization data, against access from unauthorized entities. If the TSF is selected, then it should describe how it provides confidentiality of the data while it resides outside the TOE.

Guidance
There are no guidance evaluation activities for this component.

KMD
There are no KMD evaluation activities for this component.

Tests
There are no test evaluation activities for this component.

FDP_FRS_EXT.1 Factory Reset

TSS
The evaluator shall examine the TSS to determine that it describes each of the conditions which will lead to a factory reset.

Guidance
The evaluator shall examine the operational guidance to ensure that it describes the ways the administrator can set the conditions to initiate a factory reset.

KMD
There are no KMD evaluation activities for this component.

Tests
The evaluator shall identify all functions that resets the DSC to factory setting. For each function, the evaluator shall identify all methods for authorizing the factory reset. For each function and for each authorization method, the evaluator shall create an SDE or SDO. The evaluator shall then verify the presence of the item just created. The evaluator shall initiate a factory reset using the selected function and authorization method and verify the item no longer exists.

FDP_ITC_EXT.1 Parsing of SDEs

TSS
The evaluator shall confirm the TSS contains descriptions of the supported methods the TSF uses to import SDEs into the TOE. For each import method selected, the TSS shall describe integrity verification schemes employed. The TSS shall also list the ways the TSF generates and binds security attributes to the SDEs.

Guidance
There are no AGD evaluation activities for this component.

KMD
There are no KMD evaluation activities for this component.

Tests
For each supported import method selected in FDP_ITC_EXT.1.1 and for each supported integrity verification method selected in FDP_ITC_EXT.1.2. used by the selected import method, provide one SDE with valid integrity credentials, one with invalid integrity credentials (e.g. hash). The operations with invalid integrity credentials must result in error. The operations with valid integrity credentials must return an SDO with valid security attributes in accordance with FDP_ITC_EXT.1.4.

FDP_ITC_EXT.2 Parsing of SDOs

TSS
The evaluator shall confirm the TSS contains descriptions of the supported methods the TSF uses to import SDOs into the TOE. For each import method selected, the TSS shall describe integrity verification schemes employed. The TSS shall also list the ways the TSF generates and binds security attributes to the SDOs.

Guidance
There are no AGD evaluation activities for this component.

KMD
There are no KMD evaluation activities for this component.

Tests
For each supported import method selected in FDP_ITC_EXT.2.1 and for each supported integrity verification method selected in FDP_ITC_EXT.2.2 used by the selected import method, provide one SDO with valid integrity credentials, one with invalid integrity credentials (e.g. hash). The operations with invalid integrity credentials must result in error. The operations with valid integrity credentials must return an SDO with valid security attributes in accordance with FDP_ITC_EXT.2.3, FDP_ITC_EXT.2.4, and FDP_ITC_EXT.2.5.

FDP_MFW_EXT.1 Mutable/Immutable Firmware

TSS
The evaluator shall examine the TSS and ensure that details of which firmware components are considered mutable and which firmware components are considered immutable, as well as how these firmware components can/cannot be modified or altered, are described. For example, DSC firmware components that are stored in ROM would be considered immutable.

Guidance
If the TOE has mutable firmware, the evaluator shall examine the operational guidance to ensure that it describes how to modify the firmware.

KMD
There are no KMD evaluation activities for this component.

Tests
If the TOE has mutable firmware, the evaluator shall perform the activities described in the operational guidance to modify the firmware.

FDP_RIP.1 Subset Residual Information Protection

TSS
The evaluator shall check to ensure that the TSS describes resource deallocation to the extent that they can determine that no data will be reused when reallocating resources following the destruction of an SDE or SDO. The evaluator shall ensure that this description at a minimum describes how the previous data is destroyed. The evaluator shall also ensure that this destruction method is consistent with FCS_CKM.4.

Guidance
There are no AGD evaluation activities for this component.

KMD
There are no KMD evaluation activities for this component.

Tests
Testing for FCS_CKM.4 is sufficient to address this component.

FDP_SDC_EXT.1 Confidentiality of SDEs

TSS
The evaluator shall examine the TSS to determine that it describes the protection for SDEs and authorization data and the methods of protection (e.g. protected storage, symmetric encryption, key wrapping, etc.).

The evaluator shall also examine the TSS to determine whether the TSF stores this data inside the TOE boundary or in its operational environment. If the TSF stores this data inside the TOE boundary, the evaluator shall ensure that TSF uses one of the listed methods to provide confidentiality. If the data is stored in the TOE's operational environment, the evaluator shall ensure that the TSF uses key wrapping to provide confidentiality.

The evaluator shall examine the TSS to confirm is sufficiently describes each method used to provide confidentiality for SDEs. The evaluator shall also confirm that the TOE supports all encryption methods selected.

Guidance
There are no AGD evaluation activities for this component.

KMD
There are no KMD evaluation activities for this component.

Tests
If the TOE stores SDEs and authorization data inside the TSF, the evaluator shall ensure that external interfaces cannot extract this data in plaintext.

In this case, use the evaluation activities of the FPT_PHP.3 if protected storage is selected, FCS_COP.1/SK if symmetric encryption using… is selected, and FCS_COP.1/KAT if key wrapping using… is selected.

If the TOE stores authentication data inside the operational environment, the evaluator shall ensure that plaintext data is not visible on the interface between the TOE and the operational environment.

FDP_SDI.2 Stored Data Integrity Monitoring and Action

TSS
The evaluator shall confirm that the ST author describes the methods for protecting the integrity of SDOs stored with the TOE, and shall identify the iteration of FCS_COP.1/Hash or FCS_COP.1/HMAC that covers any cryptographic algorithm used. The evaluator shall also confirm that the TSS describes the response upon the detection of an integrity error.

The evaluator shall confirm that the TSS describes the actions the TSF takes when the integrity verification fails for an SDO, including the circumstances that cause a notification to be sent when this occurs.

The evaluator shall confirm that TSS describes how integrity of SDOs is protected in FMT_MSA.3 during initialization, and how the integrity of SDOs are verified during parsing (import) in FDP_ITC_EXT.

Guidance
The evaluator shall examine the operational guidance to verify that it describes the conditions that cause a notification to be sent when an integrity error is detected, and what the contents of the notification are.

KMD
There are no KMD evaluation activities for this component.

Tests
The tests for FDP_ITC_EXT and FMT_MSA.3 shall suffice for this component.

2.2.3 Identification and Authentication

FIA_AFL_EXT.1 Authorization Failure Handling

TSS
The evaluator shall examine the TSS to determine that it contains a description for how successive unsuccessful authorization attempts are detected and tracked. The evaluator shall examine the TSS to determine that is contains a description of the actions in the event that the authorization attempt threshold is met or exceeded.

The evaluator shall also examine the TSS to determine that it describes how the failed authorization attempt counter is incremented before the authorization is verified.

The evaluator shall also examine the TSS to determine the behaviour that will occur if there are excessive failed authorization attempts, specifically whether future attempts are prevented for a static or configurable amount of time, future attempts are prevented indefinitely, or a factory reset is triggered.

Guidance

The evaluator shall examine the guidance documentation to ensure that instructions for configuring the number of successive unsuccessful authentication attempts and time period (if implemented) are provided, and that the process of unlocking the SDOs is described for each "action" specified (if that option is chosen).

The evaluator shall examine the guidance documentation to confirm that it describes, and identifies the importance of, any actions that are required in order to ensure that access to SDOs can be maintained, unless it is made permanently unavailable due to a factory reset.

KMD

There are no KMD evaluation activities for this component.

Tests

The evaluator shall perform the following tests for each method by which the TSF authorizes access the SDOs (e.g. any passwords entered as part of establishing authorization):

- **Test 1:** The evaluator shall use the operational guidance to configure the number of successive unsuccessful authorization attempts allowed by the TOE (and, if the time period selection in FIA_AFL_EXT.1.3 is included in the ST, then the evaluator shall also use the operational guidance to configure the time period after which access is re-enabled). The evaluator shall test that once the authorization attempts limit is reached, authorization attempts with valid credentials are no longer successful.
- **Test 2:** After reaching the limit for unsuccessful authorization attempts as in Test 1 above, the evaluator shall proceed as follows. If the action selected in FIA_AFL_EXT.1.3 is included in the ST then the evaluator shall confirm by testing that following the operational guidance and performing each action specified in the ST to re-enable access results in successful access. If the time period selection in FIA_AFL_EXT.1.3 is included in the ST, then the evaluator shall wait for just less than the time period configured in Test 1 and show that an authorization attempt using valid credentials does not result in successful access. The evaluator shall then wait until just after the time period configured in Test 1 and show that an authorization attempt using valid credentials results in successful access.
- **Test 3:** [conditional]: If <u>factory reset the TOE wiping out all non-persistent SDOs, as described by FDP_FRS_EXT.2</u> is selected in FIA_AFL_EXT.1.3, the evaluator shall perform the test required by FDP_FRS_EXT.2 with step 5 replaced with "The evaluator shall initiate a factory reset by deliberately meeting or surpassing the threshold for unsuccessful authorization attempts, depending on whether meets or surpasses is selected in FIA_AFL_EXT.1.3."

FIA_SOS.2 TSF Generation of Secrets

TSS

The evaluator shall ensure that the TSS describes for each of the TSF functions listed in FIA_SOS.2.2, if the available key space is configurable, and the size (or range) of the key space employed to generate authorization values.

The evaluator shall ensure that the TSS states that the quality metrics provided is based on the assumption of sufficient entropy being provided in accordance with the information given in [DSCcPP] Annex D.

The evaluator shall ensure that the TSS describes the restrictions implemented in order to restrict consecutive authentication attempts. (Authentication throttling)

The evaluator shall ensure that the TSS describes the mechanism used to generate authorization values and documents the quality metric that the mechanism provides. The information provided in the TSS shall demonstrate that:

a. The probability that a random single authentication attempt will be successful is less than one in 1,000,000; and
b. The probability that random multiple authentication attempts during a one (1) minute period will be successful is less than one in 100,000.

Guidance

The evaluator shall examine the guidance documentation to determine that it describes any configuration necessary to enforce the use of TSF generated authorization values listed in FIA_SOS.2.2.

The evaluator shall ensure that the guidance documentation provides any instructions needed to set parameters affecting the available key spaces.

KMD

There are no KMD evaluation activities for this component.

Tests

The evaluator shall perform the following tests.

- **Test 1:** The evaluator shall compose a set of 50 authorization values that meet the requirements, and 50 authorization values that fail to meet the requirements.
    a. For each authentication value that meets the requirements, the evaluator shall verify that the TOE supports the authentication value.
    b. For each authentication value that does not meet the requirements, the evaluator shall verify that the TOE does not support the authentication value.
  While the evaluator is not required (nor is it feasible) to test all possible compositions of authentication values, the evaluator shall ensure that the key space identified in the TSS is valid.
- **Test 2:** For each TSF function listed in FIA_SOS.2.2 the TOE shall be configured to generate the authentication values; the evaluator shall check that the TOE produces the authentication values.

FIA_UAU.2 User Authentication before Any Action

TSS

The evaluator shall examine the TSS to determine that it describes the identification and authentication process for each supported method (PIN/try-PIN, salted hash, etc.), the circumstances in which each supported method is used, and constitutes "successful authentication."

The evaluator shall examine the TSS to determine that it describes which actions are allowed before user identification and authentication. The evaluator shall also determine that the TSS describes, for each action that does require identification and authentication, the method and circumstances by which the authentication is performed (e.g., as per the application note, the TSF may authenticate a user once rather than each time access to an SDO is attempted; the TSS shall describe when authentication is or is not required in order to perform a TSF-mediated action).

Guidance
The evaluator shall examine the guidance documentation to determine that any necessary preparatory steps (e.g., establishing valid credential material such as PIN) to logging in are described. For each supported the login method, the evaluator shall ensure the guidance documentation provides clear instructions for successfully logging on.

KMD
There are no KMD evaluation activities for this component.

Tests
The evaluator shall use the guidance documentation to configure the appropriate credentials supported for each authentication method. For that authentication method, the evaluator shall attempt to perform TSF-mediated actions that require successful use of that authentication method and subsequently show that providing correct I&A information results in the ability to perform the requested action, while providing incorrect information results in denial of access.

FIA_UAU.5 Multiple Authentication Mechanisms

TSS
The evaluator shall examine the TSS and ensure that it describes the authentication mechanisms used to support user authentication for the Prove service as well as how each authentication mechanism provides authentication for the Prove service.

Guidance
If the supported authentication mechanisms are configurable, the evaluator shall examine the operational guidance to verify that it describes how to configure the authentication mechanisms used to provide authentication for the Prove service.

KMD
There are no KMD evaluation activities for this component.

Tests
For each supported authentication mechanism, the evaluator shall verify that valid credentials result in successful authentication and invalid credentials result in a rejected authentication attempt. If the supported authentication mechanisms are configurable, the evaluator shall follow the operational guidance to enable/disable the various mechanisms and ensure that valid credentials do not result in successful authentication if that mechanism is disabled, or that there is no interface to provide authentication credentials over an external interface when that mechanism is disabled.

FIA-UAU.6 Re-Authenticating

TSS
The evaluator shall examine the TSS to determine that it describes each of the options for reauthorization.

Guidance
There are no AGD evaluation activities for this component.

KMD
There are no KMD evaluation activities for this component.

Tests
The evaluator shall use the configuration guidance to create an SDO with each of the options for reauthorization, then identify functions to exercise each of these options, then execute these options providing the correct authorization confirming that the operation succeeded with respect to the reauthorization option chosen. The evaluator shall then attempt to execute these functions while providing the incorrect authorization and confirming that the operation fails.

2.2.4 Security Management (FMT)

FMT_MOF_EXT.1 Management of Security Functions Behavior

FMT_MSA.1 Management of Security Attributes

FMT_MSA.3 Static Attribute Initialization

2.3 Evaluation Activities for Selection-Based SFRs

The PP-Module does not define any selection-based requirements.

2.4 Evaluation Activities for Objective SFRs

The PP-Module does not define any objective requirements.

3 Evaluation Activities for SARs

3.1 Class ADV: Development

ADV_FSP.1 Basic Functional Specification (ADV_FSP.1)

There are no specific assurance activities associated with these SARs, except ensuring the information is provided. The functional specification documentation is provided to support the evaluation activities described in Section 5.1 Security Functional Requirements, and other activities described for AGD, ATE, and AVA SARs. The requirements on the content of the functional specification information is implicitly assessed by virtue of the other assurance activities being performed; if the evaluator is unable to perform an activity because there is insufficient interface information, then an adequate functional specification has not been provided.

3.2 Class AGD: Guidance Documentation

AGD_OPE.1 Operational User Guidance (AGD_OPE.1)

Some of the contents of the operational guidance are verified by the assurance activities in Section 5.1 Security Functional Requirements and evaluation of the OS according to the [CEM]. The following additional information is also required. If cryptographic functions are provided by the OS, the operational guidance shall contain instructions for configuring the cryptographic engine associated with the evaluated configuration of the OS. It shall provide a warning to the administrator that use of other cryptographic engines was not evaluated nor tested during the CC evaluation of the OS. The documentation must describe the process for verifying updates to the OS by verifying a digital signature – this may be done by the OS or the underlying platform. The evaluator will verify that this process includes the following steps: Instructions for obtaining the update itself. This should include instructions for making the update accessible to the OS (e.g., placement in a specific directory). Instructions for initiating the update process, as well as discerning whether the process was successful or unsuccessful. This includes generation of the hash/digital signature. The OS will likely contain security functionality that does not fall in the scope of evaluation under this PP. The operational guidance shall make it clear to an administrator which security functionality is covered by the evaluation activities.

AGD_PRE.1 Preparative Procedures (AGD_PRE.1)

As indicated in the introduction above, there are significant expectations with respect to the documentation—especially when configuring the operational environment to support OS functional requirements. The evaluator shall check to ensure that the guidance provided for the OS adequately addresses all platforms claimed for the OS in the ST.

3.3 Class ALC: Life-cycle Support

ALC_CMC.1 Labeling of the TOE (ALC_CMC.1)

The evaluator will check the ST to ensure that it contains an identifier (such as a product name/version number) that specifically identifies the version that meets the requirements of the ST. Further, the evaluator will check the AGD guidance and OS samples received for testing to ensure that the version number is consistent with that in the ST. If the vendor maintains a web site advertising the OS, the evaluator will examine the information on the web site to ensure that the information in the ST is sufficient to distinguish the product.

ALC_CMS.1 TOE CM Coverage (ALC_CMS.1)

The "evaluation evidence required by the SARs" in this PP is limited to the information in the ST coupled with the guidance provided to administrators and users under the AGD requirements. By ensuring that the OS is specifically identified and that this identification is consistent in the ST and in the AGD guidance (as done in the assurance activity for ALC_CMC.1), the evaluator implicitly confirms the information required by this component. Life-cycle support is targeted aspects of the developer's life-cycle and instructions to providers of applications for the developer's devices, rather than an in-depth examination of the TSF manufacturer's development and configuration management process. This is not meant to diminish the critical role that a developer's practices play in contributing to the overall trustworthiness of a product; rather, it's a reflection on the information to be made available for evaluation.
The evaluator will ensure that the developer has identified (in guidance documentation for application developers concerning the targeted platform) one or more development environments appropriate for use in developing applications for the developer's platform. For each of these development environments, the developer shall provide information on how to configure the environment to ensure that buffer overflow protection mechanisms in the environment(s) are invoked (e.g., compiler and linker flags). The evaluator will ensure that this documentation also includes an indication of whether such protections are on by default, or have to be specifically enabled. The evaluator will ensure that the TSF is uniquely identified (with respect to other products from the TSF vendor), and that documentation provided by the developer in association with the requirements in the ST is associated with the TSF using this unique identification.

ALC_TSU_EXT.1 Timely Security Updates

The evaluator will verify that the TSS contains a description of the timely security update process used by the developer to create and deploy security updates. The evaluator will verify that this description addresses the entire application. The evaluator will also verify that, in addition to the OS developer's process, any third-party processes are also addressed in the description. The evaluator will also verify that each mechanism for deployment of security updates is described.

The evaluator will verify that, for each deployment mechanism described for the update process, the TSS lists a time between public disclosure of a vulnerability and public availability of the security update to the OS patching this vulnerability, to include any third-party or carrier delays in deployment. The evaluator will verify that this time is expressed in a number or range of days.

The evaluator will verify that this description includes the publicly available mechanisms (including either an email address or website) for reporting security issues related to the OS. The evaluator shall verify that the description of this mechanism includes a method for protecting the report either using a public key for encrypting email or a trusted channel for a website.

3.4 Class ATE: Tests

ATE_IND.1 Independent Testing – Conformance (ATE_IND.1)

The evaluator will prepare a test plan and report documenting the testing aspects of the system, including any application crashes during testing. The evaluator shall determine the root cause of any application crashes and include that information in the report. The test plan covers all of the testing actions contained in the [CEM] and the body of this PP's Assurance Activities.

While it is not necessary to have one test case per test listed in an Assurance Activity, the evaluator must document in the test plan that each applicable testing requirement in the ST is covered. The test plan identifies the platforms to be tested, and for those platforms not included in the test plan but included in the ST, the test plan provides a justification for not testing the platforms. This justification must address the differences between the tested platforms and the untested platforms, and make an argument that the differences do not affect the testing to be performed. It is not sufficient to merely assert that the differences have no affect; rationale must be provided. If all platforms claimed in the ST are tested, then no rationale is necessary. The test plan describes the composition of each platform to be tested, and any setup that is necessary beyond what is contained in the AGD documentation. It should be noted that the evaluator is expected to follow the AGD documentation for installation and setup of each platform either as part of a test or as a standard pre-test condition. This may include special test drivers or tools. For each driver or tool, an argument (not just an assertion) should be provided that the driver or tool will not adversely affect the performance of the functionality by the OS and its platform.

This also includes the configuration of the cryptographic engine to be used. The cryptographic algorithms implemented by this engine are those specified by this PP and used by the cryptographic protocols being evaluated (IPsec, TLS). The test plan identifies high-level test objectives as well as the test procedures to be followed to achieve those objectives. These procedures include expected results.

The test report (which could just be an annotated version of the test plan) details the activities that took place when the test procedures were executed, and includes the actual results of the tests. This shall be a cumulative account, so if there was a test run that resulted in a failure; a fix installed; and then a successful re-run of the test, the report would show a "fail" and "pass" result (and the supporting details), and not just the "pass" result.

3.5 Class AVA: Vulnerability Assessment

AVA_VAN.1 Vulnerability Survey (AVA_VAN.1)

The evaluator will generate a report to document their findings with respect to this requirement. This report could physically be part of the overall test report mentioned in ATE_IND, or a separate document. The evaluator performs a search of public information to find vulnerabilities that have been found in similar applications with a particular focus on network protocols the application uses and document formats it parses. The evaluator documents the sources consulted and the vulnerabilities found in the report.

For each vulnerability found, the evaluator either provides a rationale with respect to its non-applicability, or the evaluator formulates a test (using the guidelines provided in ATE_IND) to confirm the vulnerability, if suitable. Suitability is determined by assessing the attack vector needed to take advantage of the vulnerability. If exploiting the vulnerability requires expert skills and an electron microscope, for instance, then a test would not be suitable and an appropriate justification would be formulated.

4 Required Supplementary Information

This Supporting Document has no required supplementary information beyond the ST, operational guidance, and testing.

Appendix A - References

Identifier	Title
[CEM]	Common Evaluation Methodology for Information Technology Security - Evaluation Methodology, CCMB-2012-09-004, Version 3.1, Revision 4, September 2012.
[CESG]	CESG - End User Devices Security and Configuration Guidance
[CSA]	Computer Security Act of 1987, H.R. 145, June 11, 1987.
[OMB]	Reporting Incidents Involving Personally Identifiable Information and Incorporating the Cost for Security in Agency Information Technology Investments, OMB M-06-19, July 12, 2006.