Tabular Presentation of the *Protection Profile for General Purpose Operating Systems*

Version: 4.2

2018-05-22

National Information Assurance Partnership

Revision History

Version	Date	Comment

Introduction

This document presents the Security Functional Requirements and Security Assurance Requirements from the *Protection Profile for General Purpose Operating Systems*. This tabular representation is provided for those audiences whose interest primarily lies in those portions of that document. The Protection Profile itself remains the only complete and authoritative representation, and includes discussion of assumptions, threats, and objectives.

Security Functional Requirements

ID	Requirement	Assurance Activity
FCS_CKM.1.1	The **OS** shall generate **asymmetric** cryptographic keys in accordance with a specified cryptographic key generation algorithm [**selection**: • ***RSA schemes*** *using cryptographic key sizes of 2048-bit or greater that meet the following:* ***FIPS PUB 186-4, "Digital Signature Standard (DSS)", Appendix B.3***, • ***ECC schemes*** *using "NIST curves" P-256, P-384 and [**selection**: P-521, no other curves] that meet the following: FIPS PUB 186-4, "Digital Signature Standard (DSS)", Appendix B.4*, • ***FFC schemes*** *using cryptographic key sizes of 2048-bit or greater that meet the following: FIPS PUB 186-4, "Digital Signature Standard (DSS)", Appendix B.1*] ~~and specified cryptographic key sizes [assignment: cryptographic key sizes] that meet the following: [assignment: list of standards]~~ . **Application Note:** The ST author shall select all key generation schemes used for key establishment and entity authentication. When key generation is used for key establishment, the schemes in FCS_CKM.2 and selected cryptographic protocols must match the selection. When key generation is used for entity authentication, the public key is expected to be	The evaluator will ensure that the TSS identifies the key sizes supported by the OS. If the ST specifies more than one scheme, the evaluator will examine the TSS to verify that it identifies the usage for each scheme. The evaluator will verify that the AGD guidance instructs the administrator how to configure the OS to use the selected key generation scheme(s) and key size(s) for all uses defined in this PP. Evaluation Activity Note: The following tests may require the vendor to furnish a developer environment and developer tools that are typically not available to end-users of the OS. **Key Generation for FIPS PUB 186-4 RSA Schemes** The evaluator will verify the implementation of RSA Key Generation by the OS 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. Key Pair generation specifies 5 ways (or methods) to generate the primes p and q. These include: 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

	associated with an X.509v3 certificate. If the OS acts only as a receiver in the RSA key establishment scheme, the OS does not need to implement RSA key generation.	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. This includes the random seed(s), the public exponent of the RSA key, and the desired key length. For each key length supported, the evaluator shall have the TSF generate 25 key pairs. The evaluator will verify the correctness of the TSF's implementation by comparing values generated by the TSF with those generated from a known good implementation. If possible, the Random Probable primes method should also be verified against a known good implementation as described above. Otherwise, the evaluator will have the TSF generate 10 keys pairs for each supported key length nlen and verify: • $n = p \cdot q$, • p and q are probably prime according to Miller-Rabin tests, • GCD(p-1,e) = 1, • GCD(q-1,e) = 1, • $2^{16} \leq e \leq 2^{256}$ and e is an odd integer, • $\|p-q\| > 2^{nlen/2 - 100}$, • $p \geq 2^{nlen/2 -1/2}$, • $q \geq 2^{nlen/2 -1/2}$, • $2^{(nlen/2)} < d < LCM(p-1,q-1)$, • $e \cdot d = 1 \bmod LCM(p-1,q-1)$. **Key Generation for Elliptic Curve Cryptography (ECC)** FIPS 186-4 ECC Key Generation Test For each supported NIST curve, i.e., P-256, P-384 and P-521, the evaluator will require the implementation under test (IUT) to generate 10 private/public key pairs. The private key shall be generated using an approved random bit generator (RBG). To determine correctness, the evaluator will submit the generated key pairs to the public key verification (PKV) function of a known good implementation. FIPS 186-4 Public Key Verification (PKV) Test For each supported NIST curve, i.e., P-256, P-384 and P-521, the evaluator will generate 10 private/public key pairs using the key generation function of a known good implementation and modify five of the public key values so that they are incorrect, leaving five values unchanged (i.e., correct). The evaluator will obtain in response a set of 10 PASS/FAIL values. **Key Generation for Finite-Field Cryptography (FFC)** The evaluator will 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: • Cryptographic and Field Primes: • 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: • Cryptographic Group Generator: • 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: • Private Key: • 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 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 and/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 will have the TSF generate 25 parameter sets and key pairs. The evaluator will 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.
FCS_CKM.2.1	The OS shall implement functionality to perform cryptographic key establishment in accordance with a specified cryptographic key establishment method: **RSA-based key establishment schemes** that meets the following: NIST Special Publication 800-56B, "Recommendation for Pair-Wise Key Establishment Schemes Using Integer Factorization Cryptography" and [**selection**:	The evaluator will ensure that the supported key establishment schemes correspond to the key generation schemes identified in FCS_CKM.1.1. If the ST specifies more than one scheme, the evaluator will examine the TSS to verify that it identifies the usage for each scheme. The evaluator will verify that the AGD guidance instructs the administrator how to configure the OS to use the selected key establishment scheme(s). Evaluation Activity Note: The following tests require the developer to

- ***Elliptic curve-based key establishment schemes*** *that meets the following: NIST Special Publication 800-56A, "Recommendation for Pair-Wise Key Establishment Schemes Using Discrete Logarithm Cryptography",*
- ***Finite field-based key establishment schemes*** *that meets the following: NIST Special Publication 800-56A, "Recommendation for Pair-Wise Key Establishment Schemes Using Discrete Logarithm Cryptography",*
- *No other schemes*

] ~~that meets the following: [assignment: list of standards]~~ .

Application Note: The ST author shall select all key establishment schemes used for the selected cryptographic protocols. FCS_TLSC_EXT.1 requires cipher suites that use RSA-based key establishment schemes.

The RSA-based key establishment schemes are described in Section 9 of NIST SP 800-56B; however, Section 9 relies on implementation of other sections in SP 800-56B. If the OS acts as a receiver in the RSA key establishment scheme, the OS does not need to implement RSA key generation.

The elliptic curves used for the key establishment scheme shall correlate with the curves specified in FCS_CKM.1.1. The domain parameters used for the finite field-based key establishment scheme are specified by the key generation according to FCS_CKM.1.1.

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

Key Establishment Schemes

The evaluator will verify the implementation of the key establishment schemes supported by the OS using the applicable tests below.

SP800-56A Key Establishment Schemes

The evaluator will verify the OS'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 the OS has implemented the components of the key agreement scheme according to the specifications in the Recommendation. These components include the calculation of the discrete logarithm cryptography (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 will 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 MAC data and the calculation of MAC tag.

> **Function Test**
>
> The Function test verifies the ability of the OS to implement the key agreement schemes correctly. To conduct this test the evaluator will generate or obtain test vectors from a known good implementation of the OS's 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 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 will obtain the DKM, the corresponding OS's public keys (static and/or ephemeral), the MAC tag(s), and any inputs used in the KDF, such as the Other Information field OI and OS id fields.
>
> If the OS does not use a KDF defined in SP 800-56A, the evaluator will obtain only the public keys and the hashed value of the shared secret.
>
> The evaluator will 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 OS shall perform the above for each implemented approved MAC algorithm.
>
> **Validity Test**
>
> The Validity test verifies the ability of the OS to recognize another party's valid and invalid key agreement results with or without key confirmation. To conduct this test, the evaluator will obtain a list of the supporting cryptographic functions included in the SP800-56A key agreement implementation to determine which errors the OS should be able to recognize. The evaluator generates a set of 30 test vectors consisting of data sets including domain parameter values or NIST approved curves, the evaluator's public keys, the OS's public/private key pairs, MAC tag, and any inputs used in the KDF, such as the other info and OS id fields.
>
> The evaluator will inject an error in some of the test vectors to test that the OS 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 MAC'd, or the generated MAC tag. If the OS contains the full or partial (only ECC) public key validation, the evaluator will also individually inject errors in both parties' static public keys, both parties' ephemeral public keys and the OS's static private key to assure the OS detects errors in the public key validation function and/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 OS shall use these modified test vectors to emulate the key agreement scheme using the corresponding parameters. The evaluator will compare the OS's results with the results using a known good implementation verifying that the OS detects these errors.

SP800-56B Key Establishment Schemes

The evaluator will verify that the TSS describes whether the OS acts as a sender, a recipient, or both for RSA-based key establishment schemes.

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

> To conduct this test the evaluator will generate or obtain test vectors from a known good implementation of the OS's 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 MAC key and MAC tag if key confirmation is

		incorporated, and the outputted ciphertext. For each test vector, the evaluator shall perform a key establishment encryption operation on the OS with the same inputs (in cases where key confirmation is incorporated, the test shall use the MAC key from the test vector instead of the randomly generated MAC key used in normal operation) and ensure that the outputted ciphertext is equivalent to the ciphertext in the test vector. If the OS acts as a receiver, the following evaluation activities shall be performed to ensure the proper operation of every OS supported combination of RSA-based key establishment scheme: To conduct this test the evaluator will generate or obtain test vectors from a known good implementation of the OS's 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 KTS-OAEP is supported), the tester shall generate 10 sets of test vectors. Each test vector shall include the RSA private key, the plaintext keying material, any additional input parameters if applicable, the MAC tag in cases where key confirmation is incorporated, and the outputted ciphertext. For each test vector, the evaluator will perform the key establishment decryption operation on the OS and ensure that the outputted plaintext keying material is equivalent to the plaintext keying material in the test vector. In cases where key confirmation is incorporated, the evaluator will perform the key confirmation steps and ensure that the outputted MAC tag is equivalent to the MAC tag in the test vector. The evaluator will ensure that the TSS describes how the OS handles decryption errors. In accordance with NIST Special Publication 800-56B, the OS 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 will 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. If KTS-KEM-KWS is supported, the evaluator will create separate contrived ciphertext values that trigger each of the three decryption error checks described in NIST Special Publication 800-56B section 7.2.3.3, ensure that each decryption attempt results in an error, and ensure that any outputted or logged error message is identical for each.
FCS_CKM_EXT.4.1	The OS shall destroy cryptographic keys and key material in accordance with a specified cryptographic key destruction method [**selection**: • *For volatile memory, the destruction shall be executed by a [**selection**:* ◦ *single overwrite consisting of [**selection**: a pseudo-random pattern using the TSF's RBG, zeroes, ones, a new value of a key, [**assignment**: any value that does not contain any CSP]],* ◦ *removal of power to the memory,* ◦ *destruction of reference to the key directly followed by a request for garbage collection* *],* • *For non-volatile memory that consists of the invocation of an interface provided by the underlying platform that [**selection**:* ◦ *logically addresses the storage location of the key and performs a [**selection**: single, [**assignment**: ST author defined multi-pass]] overwrite consisting of [**selection**: zeroes, ones, pseudo-random pattern, a new value of a key of the same size, [**assignment**: any value that does not contain any CSP]],* ◦ *instructs the underlying platform to destroy the abstraction that represents the key* *]*] . **Application Note:** The interface referenced in the requirement could take different forms, the most likely of which is an application programming interface to an OS kernel. There may be various levels of abstraction visible. For instance, in a given implementation, selection a, the application may have access to the file system details and may be able to logically address specific memory locations. In another implementation, selection b, the application may simply have a handle to a resource and can only ask the platform to delete the resource, as may be the case with a platforms secure key store. Selection b should only be used for the most restricted access. The level of detail to which the TOE has access will be reflected in the TSS section of the ST. Several selections allow assignment of a 'value that does not contain any CSP'. This means that the TOE uses some other specified data not drawn from a source that may contain key material or reveal information about key material, and not being any of the particular values listed as other selection options. The point of the phrase 'does not contain any CSP' is to ensure that the	

	overwritten data is carefully selected, and not taken from a general 'pool' that might contain current or residual data that itself requires confidentiality protection.	
FCS_CKM_EXT.4.2	The OS shall destroy all keys and key material when no longer needed. **Application Note:** For the purposes of this requirement, key material refers to authentication data, passwords, secret/private symmetric keys, private asymmetric keys, data used to derive keys, values derived from passwords, etc. Key destruction procedures are performed in accordance with FCS_CKM_EXT.4.1.	The evaluator examines the TSS to ensure it describes how the keys are managed in volatile memory. This description includes details of how each identified key is introduced into volatile memory (e.g. by derivation from user input, or by unwrapping a wrapped key stored in non-volatile memory) and how they are overwritten. The evaluator will check to ensure the TSS lists each type of key that is stored in in non-volatile memory, and identifies how the TOE interacts with the underlying platform to manage keys (e.g., store, retrieve, destroy). The description includes details on the method of how the TOE interacts with the platform, including an identification and description of the interfaces it uses to manage keys (e.g., file system APIs, platform key store APIs). If the ST makes use of the open assignment and fills in the type of pattern that is used, the evaluator examines the TSS to ensure it describes how that pattern is obtained and used. The evaluator will verify that the pattern does not contain any CSPs. The evaluator will check that the TSS identifies any configurations or circumstances that may not strictly conform to the key destruction requirement. **Operational Guidance** There are a variety of concerns that may prevent or delay key destruction in some cases. The evaluator will 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 any other relevant Required Supplementary Information. The evaluator will check that the guidance documentation provides guidance on situations where key destruction may be delayed at the physical layer and how such situations can be avoided or mitigated if possible. Some examples of what is expected to be in the documentation are provided here. When the TOE does not have full access to the physical memory, it is possible that the storage may be implementing wear-leveling and garbage collection. This may create additional copies of the key that are logically inaccessible but persist physically. In this case, to mitigate this the drive should support the TRIM command and implements garbage collection to destroy these persistent copies when not actively engaged in other tasks. Drive vendors implement garbage collection in a variety of different ways, as such there is a variable amount of time until data is truly removed from these solutions. There is a risk that data may persist for a longer amount of time if it is contained in a block with other data not ready for erasure. To reduce this risk, the operating system and file system of the OE should support TRIM, instructing the non-volatile memory to erase copies via garbage collection upon their deletion. If a RAID array is being used, only set-ups that support TRIM are utilized. If the drive is connected via PCI-Express, the operating system supports TRIM over that channel. The drive should be healthy and contains minimal corrupted data and should be end-of-lifed before a significant amount of damage to drive health occurs, this minimizes the risk that small amounts of potentially recoverable data may remain in damaged areas of the drive. • **Test 1:** Applied to each key held as in volatile memory and subject to destruction by overwrite by the TOE (whether or not the value is subsequently encrypted for storage in volatile or non-volatile memory). In the case where the only selection made for the destruction method key was removal of power, then this test is unnecessary. The evaluator will: 1. Record the value of the key in the TOE subject to clearing. 2. Cause the TOE to perform a normal cryptographic processing with the key from Step #1. 3. Cause the TOE to clear the key. 4. Cause the TOE to stop the execution but not exit. 5. Cause the TOE to dump the entire memory of the TOE into a binary file. 6. Search the content of the binary file created in Step #5 for instances of the known key value from Step #1. Steps 1-6 ensure that the complete key does not exist anywhere in volatile memory. If a copy is found, then the test fails. • **Test 2:** Applied to each key help in non-volatile memory and subject to destruction by the TOE. The evaluator will use special tools (as needed), provided by the TOE developer if necessary, to ensure the tests function as intended. 1. Identify the purpose of the key and what access should fail when it is deleted. (e.g. the data encryption key being deleted would cause data decryption to fail.) 2. Cause the TOE to clear the key. 3. Have the TOE attempt the functionality that the cleared key would be necessary for. The test succeeds if step 3 fails. • **Test 3:** The following tests apply only to selection a), since the TOE in this instance has more visibility into what is happening within the underlying platform (e.g., a logical view of the media). In selection b), the TOE has no visibility into the inner workings and completely relies on the underlying platform, so there is no reason to test the

		TOE beyond test 2. For selection a), the following tests are used to determine the TOE is able to request the platform to overwrite the key with a TOE supplied pattern. Applied to each key held in non-volatile memory and subject to destruction by overwrite by the TOE. The evaluator will use a tool that provides a logical view of the media (e.g., MBR file system): 1. Record the value of the key in the TOE subject to clearing. 2. Cause the TOE to perform a normal cryptographic processing with the key from Step #1. 3. Cause the TOE to clear the key. 4. Search the logical view that the key was stored in for instances of the known key value from Step #1. If a copy is found, then the test fails. - **Test 4:** Applied to each key held as non-volatile memory and subject to destruction by overwrite by the TOE. The evaluator will use a tool that provides a logical view of the media: 1. Record the logical storage location of the key in the TOE subject to clearing. 2. Cause the TOE to perform a normal cryptographic processing with the key from Step #1. 3. Cause the TOE to clear the key. 4. Read the logical storage location in Step #1 of non-volatile memory to ensure the appropriate pattern is utilized. The test succeeds if correct pattern is used to overwrite the key in the memory location. If the pattern is not found the test fails.
FCS_COP.1.1(1)	The **OS** shall perform encryption/decryption services for data in accordance with a specified cryptographic algorithm [**selection**: - *AES-XTS (as defined in NIST SP 800-38E)*, - *AES-CBC (as defined in NIST SP 800-38A)*] and [**selection**: - *AES-CCMP (as defined in FIPS PUB 197, NIST SP 800-38C and IEEE 802.11-2012)*, - *AES Key Wrap (KW) (as defined in NIST SP 800-38F)*, - *AES Key Wrap with Padding (KWP) (as defined in NIST SP 800-38F)*, - *AES-GCM (as defined in NIST SP 800-38D)*, - *AES-CCM (as defined in NIST SP 800-38C)*, - *AES-CCMP-256 (as defined in NIST SP800-38C and IEEE 802.11ac-2013)*, - *AES-GCMP-256 (as defined in NIST SP800-38D and IEEE 802.11ac-2013)*, - *no other modes*] and cryptographic key sizes [**selection**: *128-bit*, *256-bit*] ~~that meet the following: [assignment: list of standards]~~ . **Application Note:** AES CCMP (which uses AES in CCM as specified in SP 800-38C) becomes mandatory and must be selected if the ST includes the WLAN Client EP. For the second selection, the ST author should choose the mode or modes in which AES operates. For the third selection, the ST author should choose the key sizes that are supported by this functionality. 128-bit key size is required in order to comply with FCS_TLSC_EXT.1 and FCS_CKM.1, if those are selected.	The evaluator will verify that the AGD documents contains instructions required to configure the OS to use the required modes and key sizes. The evaluator will execute all instructions as specified to configure the OS to the appropriate state. The evaluator will perform all of the following tests for each algorithm implemented by the OS and used to satisfy the requirements of this PP: **AES-CBC Known Answer Tests** There are four Known Answer Tests (KATs), described below. In all KATs, the plaintext, ciphertext, and IV values shall be 128-bit blocks. 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 will compare the resulting values to those obtained by submitting the same inputs to a known good implementation. - KAT-1. To test the encrypt functionality of AES-CBC, the evaluator will supply a set of 10 plaintext values 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. Five plaintext values shall be encrypted with a 128-bit all-zeros key, and the other five shall be encrypted with a 256-bit all- zeros key. To test the decrypt functionality of AES-CBC, the evaluator will perform the same test as for encrypt, using 10 ciphertext values as input and AES-CBC decryption. - KAT-2. To test the encrypt functionality of AES-CBC, the evaluator will supply a set of 10 key values 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. Five of the keys shall be 128-bit keys, and the other five shall be 256-bit keys. To test the decrypt functionality of AES-CBC, the evaluator will perform the same test as for encrypt, using an all-zero ciphertext value as input and AES-CBC decryption. - KAT-3. To test the encrypt functionality of AES-CBC, the evaluator will supply the two sets of key values described below and obtain the ciphertext value that results from AES encryption of an all-zeros plaintext using the given key value and an IV of all zeros. The first set of keys shall have 128 128-bit keys, and the second set shall have 256 256-bit keys. Key i in each set shall have the leftmost i bits be ones and the rightmost N-i bits be zeros, for i in [1,N]. To test the decrypt functionality of AES-CBC, the evaluator will supply the two sets of key and ciphertext value pairs described below and obtain the plaintext value that results from AES-CBC decryption of the given ciphertext using the given key and an IV of all zeros. The first set of key/ciphertext pairs shall have 128 128-bit key/ciphertext pairs, and the second set of key/ciphertext pairs shall have 256 256-bit key/ciphertext pairs. Key i in each set shall have the leftmost i bits be ones and the rightmost N-i bits be zeros, for i in [1,N]. The ciphertext value in each pair shall be the value that results in an all-zeros plaintext when decrypted with its corresponding key. - KAT-4. To test the encrypt functionality of AES-CBC, the evaluator will supply the set of 128 plaintext values described below and obtain the two ciphertext values that result from AES-CBC encryption of the given plaintext using a 128-bit key value of all zeros with an IV of all zeros and using a 256-bit key value of all zeros with an IV of all zeros, respectively. Plaintext value i in each set shall have the leftmost i bits be ones and the rightmost 128-i bits be zeros, for i in [1,128]. To test the decrypt functionality of AES-CBC, the evaluator will perform the same test as for encrypt, using ciphertext values of the same form as the plaintext in the encrypt test as input and AES-CBC decryption. **AES-CBC Multi-Block Message Test** The evaluator will test the encrypt functionality by encrypting an i-block message where $1 < i \leq 10$. The evaluator will choose a key, an IV and plaintext message of length i blocks and encrypt the message, using the

mode to be tested, with the chosen key and IV. The ciphertext shall be compared to the result of encrypting the same plaintext message with the same key and IV using a known good implementation. The evaluator will also test the decrypt functionality for each mode by decrypting an i-block message where 1 < i ≤10. The evaluator will choose a key, an IV and a ciphertext message of length i blocks and decrypt the message, using the mode to be tested, with the chosen key and IV. The plaintext shall be compared to the result of decrypting the same ciphertext message with the same key and IV using a known good implementation.

AES-CBC Monte Carlo Tests

The evaluator will test the encrypt functionality using a set of 200 plaintext, IV, and key 3- tuples. 100 of these shall use 128 bit keys, and 100 shall use 256 bit keys. The plaintext and IV values shall be 128-bit blocks. For each 3-tuple, 1000 iterations shall be run as follows:

```
# Input: PT, IV, Key
for i = 1 to 1000:
  if i == 1:
        CT[1] = AES-CBC-Encrypt(Key, IV, PT)
        PT = IV
  else:
    CT[i] = AES-CBC-Encrypt(Key, PT)
    PT = CT[i-1]
```

The ciphertext computed in the 1000th iteration (i.e., CT[1000]) is the result for that trial. This result shall be compared to the result of running 1000 iterations with the same values using a known good implementation.

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

AES-GCM Monte Carlo Tests

The evaluator will test the authenticated encrypt functionality of AES-GCM for each combination of the following input parameter lengths:

- 128 bit and 256 bit keys
- Two plaintext lengths. One of the plaintext lengths shall be a non-zero integer multiple of 128 bits, if supported. The other plaintext length shall not be an integer multiple of 128 bits, if supported.
- Three AAD lengths. One AAD length shall be 0, if supported. One AAD length shall be a non-zero integer multiple of 128 bits, if supported. One AAD length shall not be an integer multiple of 128 bits, if supported.
- Two IV lengths. If 96 bit IV is supported, 96 bits shall be one of the two IV lengths tested.

The evaluator will test the encrypt functionality using a set of 10 key, plaintext, AAD, and IV tuples for each combination of parameter lengths above and obtain the ciphertext value and tag that results from AES-GCM authenticated encrypt. Each supported tag length shall be tested at least once per set of 10. The IV value may be supplied by the evaluator or the implementation being tested, as long as it is known.

The evaluator will test the decrypt functionality using a set of 10 key, ciphertext, tag, AAD, and IV 5-tuples for each combination of parameter lengths above and obtain a Pass/Fail result on authentication and the decrypted plaintext if Pass. The set shall include five tuples that Pass and five that Fail.

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 will compare the resulting values to those obtained by submitting the same inputs to a known good implementation.

AES-CCM Tests

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

- 128 bit and 256 bit keys
- Two payload lengths. One payload length shall be the shortest supported payload length, greater than or equal to zero bytes. The other payload length shall be the longest supported payload length, less than or equal to 32 bytes (256 bits).
- Two or three associated data lengths. One associated data length shall be 0, if supported. One associated data length shall be the shortest supported payload length, greater than or equal to zero bytes. One associated data length shall be the longest supported payload length, less than or equal to 32 bytes (256 bits). If the implementation supports an associated data length of 2 16 bytes, an associated data length of 216 bytes shall be tested.
- Nonce lengths. All supported nonce lengths between 7 and 13 bytes, inclusive, shall be tested.
- Tag lengths. All supported tag lengths of 4, 6, 8, 10, 12, 14 and 16 bytes shall be tested.

To test the generation-encryption functionality of AES-CCM, the evaluator will perform the following four tests:

- **Test 1:** For EACH supported key and associated data length and ANY supported payload, nonce and tag length, the evaluator will supply one key value, one nonce value and 10 pairs of associated data and payload values and obtain the resulting ciphertext.
- **Test 2:** For EACH supported key and payload length and ANY supported associated data, nonce and tag length, the evaluator will supply one key value, one nonce value and 10 pairs of associated data and payload values and obtain the resulting ciphertext.
- **Test 3:** For EACH supported key and nonce length and ANY

supported associated data, payload and tag length, the evaluator will supply one key value and 10 associated data, payload and nonce value 3-tuples and obtain the resulting ciphertext.
- **Test 4:** For EACH supported key and tag length and ANY supported associated data, payload and nonce length, the evaluator will supply one key value, one nonce value and 10 pairs of associated data and payload values and obtain the resulting ciphertext.

To determine correctness in each of the above tests, the evaluator will compare the ciphertext with the result of generation-encryption of the same inputs with a known good implementation.

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 nonce, associated data and ciphertext 3-tuples and obtain either a FAIL result or a PASS result with the decrypted payload. The evaluator will supply 10 tuples that should FAIL and 5 that should PASS per set of 15.

Additionally, the evaluator will use tests from the IEEE 802.11-02/362r6 document "Proposed Test vectors for IEEE 802.11 TGi", dated September 10, 2002, Section 2.1 AESCCMP Encapsulation Example and Section 2.2 Additional AES CCMP Test Vectors to further verify the IEEE 802.11-2007 implementation of AES-CCMP.

AES-GCM Test

The evaluator will test the authenticated encrypt functionality of AES-GCM for each combination of the following input parameter lengths:

- 128 bit and 256 bit keys
- Two plaintext lengths. One of the plaintext lengths shall be a non-zero integer multiple of 128 bits, if supported. The other plaintext length shall not be an integer multiple of 128 bits, if supported.
- Three AAD lengths. One AAD length shall be 0, if supported. One AAD length shall be a non-zero integer multiple of 128 bits, if supported. One AAD length shall not be an integer multiple of 128 bits, if supported.
- Two IV lengths. If 96 bit IV is supported, 96 bits shall be one of the two IV lengths tested.

The evaluator will test the encrypt functionality using a set of 10 key, plaintext, AAD, and IV tuples for each combination of parameter lengths above and obtain the ciphertext value and tag that results from AES-GCM authenticated encrypt. Each supported tag length shall be tested at least once per set of 10. The IV value may be supplied by the evaluator or the implementation being tested, as long as it is known.

The evaluator will test the decrypt functionality using a set of 10 key, ciphertext, tag, AAD, and IV 5-tuples for each combination of parameter lengths above and obtain a Pass/Fail result on authentication and the decrypted plaintext if Pass. The set shall include five tuples that Pass and five that Fail.

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 will compare the resulting values to those obtained by submitting the same inputs to a known good implementation.

XTS-AES Test

The evaluator will test the encrypt functionality of XTS-AES for each combination of the following input parameter lengths:

- 256 bit (for AES-128) and 512 bit (for AES-256) keys
- Three data unit (i.e., plaintext) lengths. One of the data unit lengths shall be a nonzero integer multiple of 128 bits, if supported. One of the data unit lengths shall be an integer multiple of 128 bits, if supported. The third data unit length shall be either the longest supported data unit length or 216 bits, whichever is smaller.

using a set of 100 (key, plaintext and 128-bit random tweak value) 3-tuples and obtain the ciphertext that results from XTS-AES encrypt.

The evaluator may supply a data unit sequence number instead of the tweak value if the implementation supports it. The data unit sequence number is a base-10 number ranging between 0 and 255 that implementations convert to a tweak value internally.

The evaluator will 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 XTSAES decrypt.

AES Key Wrap (AES-KW) and Key Wrap with Padding (AES-KWP) Test

The evaluator will test the authenticated encryption functionality of AES-KW for EACH combination of the following input parameter lengths:

- 128 and 256 bit key encryption keys (KEKs)
- Three plaintext lengths. One of the plaintext lengths shall be two semi-blocks (128 bits). One of the plaintext lengths shall be three semi-blocks (192 bits). The third data unit length shall be the longest supported plaintext length less than or equal to 64 semi-blocks (4096 bits).

using a set of 100 key and plaintext pairs and obtain the ciphertext that results from AES-KW authenticated encryption. To determine correctness, the evaluator will use the AES-KW authenticated-encryption function of a known good implementation.

The evaluator will 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 with AES-KW authenticated-decryption.

		The evaluator will test the authenticated-encryption functionality of AES-KWP using the same test as for AES-KW authenticated-encryption with the following change in the three plaintext lengths: • One plaintext length shall be one octet. One plaintext length shall be 20 octets (160 bits). • One plaintext length shall be the longest supported plaintext length less than or equal to 512 octets (4096 bits). The evaluator will test the authenticated-decryption functionality of AES-KWP using the same test as for AES-KWP authenticated-encryption, replacing plaintext values with ciphertext values and AES-KWP authenticated-encryption with AES-KWP authenticated-decryption.
FCS_COP.1.1(2)	The **OS** shall perform cryptographic hashing services in accordance with a specified cryptographic algorithm SHA-1 and [**selection**: • *SHA-256*, • *SHA-384*, • *SHA-512*, • *no other algorithms*] and message digest sizes 160 bits and [**selection**: • *256 bits*, • *384 bits*, • *512 bits*, • *no other sizes*] that meet the following: FIPS Pub 180-4. **Application Note:** Per NIST SP 800-131A, SHA-1 for generating digital signatures is no longer allowed, and SHA-1 for verification of digital signatures is strongly discouraged as there may be risk in accepting these signatures. SHA-1 is currently required in order to comply with FCS_TLSC_EXT.1 and, depending on selections, FCS_DTLS_EXT.1. Vendors are strongly encouraged to implement updated protocols that support the SHA-2 family; until updated protocols are supported, this PP allows support for SHA-1 implementations in compliance with SP 800-131A. The intent of this requirement is to specify the hashing function. The hash selection must support the message digest size selection. The hash selection should be consistent with the overall strength of the algorithm used.	The evaluator will check that the association of the hash function with other application cryptographic functions (for example, the digital signature verification function) is documented in the TSS. The TSF hashing functions can be implemented in one of two modes. The first mode is the byte-oriented mode. In this mode the TSF only 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 mode is the bit-oriented mode. In this mode the TSF hashes messages of arbitrary length. As there are different tests for each mode, an indication is given in the following sections for the bit-oriented vs. the byte-oriented testmacs. The evaluator will perform all of the following tests for each hash algorithm implemented by the TSF and used to satisfy the requirements of this PP. The following tests require the developer to provide access to a test application that provides the evaluator with tools that are typically not found in the production application. • **Test 1:** Short Messages Test (Bit oriented Mode) - The evaluator will generate an input set consisting of m+1 messages, where m is the block length of the hash algorithm. The length of the messages range sequentially from 0 to m bits. The message text shall be pseudorandomly generated. The evaluator will 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. • **Test 2:** Short Messages Test (Byte oriented Mode) - The evaluator will generate an input set consisting of m/8+1 messages, where m is the block length of the hash algorithm. The length of the messages range sequentially from 0 to m/8 bytes, with each message being an integral number of bytes. The message text shall be pseudorandomly generated. The evaluator will 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. • **Test 3:** Selected Long Messages Test (Bit oriented Mode) - The evaluator will generate an input set consisting of m messages, where m is the block length of the hash algorithm. The length of the *i*th message is 512 + 99·i, where 1 ≤ i ≤ m. The message text shall be pseudorandomly generated. The evaluator will 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. • **Test 4:** Selected Long Messages Test (Byte oriented Mode) - The evaluator will generate an input set consisting of m/8 messages, where m is the block length of the hash algorithm. The length of the *i*th message is 512 + 8·99·i, where 1 ≤ i ≤ m/8. The message text shall be pseudorandomly generated. The evaluator will 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. • **Test 5:** Pseudorandomly Generated Messages Test - This test is for byte-oriented implementations only. The evaluator will 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 evaluator will then formulate a set of 100 messages and associated digests by following the algorithm provided in Figure 1 of [SHAVS]. The evaluator will then ensure that the correct result is produced when the messages are provided to the TSF.
FCS_COP.1.1(3)	The **OS** shall perform cryptographic signature services (generation and verification) in accordance with a specified cryptographic algorithm [**selection**: • ***RSA schemes*** *using cryptographic key sizes of 2048-bit or greater that meet the following: FIPS PUB 186-4, "Digital Signature Standard (DSS)", Section 4*, • ***ECDSA schemes*** *using "NIST curves" P-256, P-384 and [**selection**: P-521, no other curves] that meet the following: FIPS PUB 186-4, "Digital Signature Standard (DSS)", Section 5*] ~~and cryptographic key sizes [assignment: cryptographic algorithm] that meet the following: [assignment: list of standards]~~. **Application Note:** The ST Author should choose the algorithm implemented to perform digital signatures; if more than one algorithm is available, this requirement should be iterated to specify the functionality. For the algorithm chosen, the ST author should make the appropriate assignments/selections to specify the parameters that are implemented for that algorithm. RSA signature generation and verification is currently required in order to comply with FCS_TLSC_EXT.1.	The evaluator will perform the following activities based on the selections in the ST. The following tests require the developer to provide access to a test application that provides the evaluator with tools that are typically not found in the production application. **ECDSA Algorithm Tests** • **Test 1:** ECDSA FIPS 186-4 Signature Generation Test. For each supported NIST curve (i.e., P-256, P-384 and P-521) and SHA function pair, the evaluator will generate 10 1024-bit long messages and obtain for each message a public key and the resulting signature values R and S. To determine correctness, the evaluator will use the signature verification function of a known good implementation. • **Test 2:** ECDSA FIPS 186-4 Signature Verification Test. For each supported NIST curve (i.e., P-256, P-384 and P-521) and SHA function pair, the evaluator will generate a set of 10 1024-bit message, public key and signature tuples and modify one of the values (message, public key or signature) in five of the 10 tuples. The evaluator will verify that 5 responses indicate success and 5 responses indicate failure. **RSA Signature Algorithm Tests** • **Test 1:** Signature Generation Test. The evaluator will verify the

		implementation of RSA Signature Generation by the OS using the Signature Generation Test. To conduct this test the evaluator must generate or obtain 10 messages from a trusted reference implementation for each modulus size/SHA combination supported by the TSF. The evaluator will have the OS use its private key and modulus value to sign these messages. The evaluator will verify the correctness of the TSF's signature using a known good implementation and the associated public keys to verify the signatures. • **Test 2:** Signature Verification Test. The evaluator will perform the Signature Verification test to verify the ability of the OS to recognize another party's valid and invalid signatures. The evaluator will inject errors into the test vectors produced during the Signature Verification Test by introducing errors in some of the public keys, e, messages, IR format, and/or signatures. The evaluator will verify that the OS returns failure when validating each signature.
FCS_COP.1.1(4)	The **OS** shall perform keyed-hash message authentication services in accordance with a specified cryptographic algorithm [**selection**: *SHA-1*, *SHA-256*, *SHA-384*, *SHA-512*] with key sizes [**assignment**: *key size (in bits) used in HMAC*] **and message digest sizes [selection: *160 bits*, *256 bits*, *384 bits*, *512 bits*]** that meet the following: FIPS Pub 198-1 *The Keyed-Hash Message Authentication Code* and FIPS Pub 180-4 *Secure Hash Standard*. **Application Note:** The intent of this requirement is to specify the keyed-hash message authentication function used for key establishment purposes for the various cryptographic protocols used by the OS (e.g., trusted channel). The hash selection must support the message digest size selection. The hash selection should be consistent with the overall strength of the algorithm used for FCS_COP.1(1). Though HMAC with SHA-256 (HMAC-SHA-256) is listed as a selectable cipher suite in this requirement, FCS_TLSC_EXT.1 effectively makes its implementation mandatory for compliant OSes. SHA-1 is currently required in order to comply with FCS_TLSC_EXT.1 and, depending on selections, FCS_DTLS_EXT.1. SHA-1 is currently required in order to comply with FCS_TLSC_EXT.1, but has been deprecated and should not be used for any other purpose beyond TLS and DTLS	The evaluator will perform the following activities based on the selections in the ST. For each of the supported parameter sets, the evaluator will compose 15 sets of test data. Each set shall consist of a key and message data. The evaluator will have the OS generate HMAC tags for these sets of test data. The resulting MAC tags shall be compared against the result of generating HMAC tags with the same key and IV using a known-good implementation.
FCS_DTLS_EXT.1.1	The OS shall implement the DTLS protocol in accordance with [**selection**: *DTLS 1.0 (RFC 4347)*, *DTLS 1.2 (RFC 6347)*]. ***This is a selection-based requirement. Its inclusion depends upon selection in .***	• **Test 1:** The evaluator will attempt to establish a connection with a DTLS server, observe the traffic with a packet analyzer, and verify that the connection succeeds and that the traffic is identified as DTLS. Other tests are performed in conjunction with the evaluation activity listed for FCS_TLSC_EXT.1.
FCS_DTLS_EXT.1.2	The OS shall implement the requirements in TLS (FCS_TLSC_EXT.1) for the DTLS implementation, except where variations are allowed according to DTLS 1.2 (RFC 6347). ***This is a selection-based requirement. Its inclusion depends upon selection in .*** **Application Note:** Differences between DTLS 1.2 and TLS 1.2 are outlined in RFC 6347; otherwise the protocols are the same. In particular, for the applicable security characteristics defined for the TSF, the two protocols do not differ. Therefore, all application notes and evaluation activities that are listed for TLS apply to the DTLS implementation.	The evaluator will perform the evaluation activities listed for FCS_TLSC_EXT.1.
FCS_RBG_EXT.1.1	The OS shall perform all deterministic random bit generation (DRBG) services in accordance with NIST Special Publication 800-90A using [**selection**: • *Hash_DRBG (any)*, • *HMAC_DRBG (any)*, • *CTR_DRBG (AES)*] . **Application Note:** NIST SP 800-90A contains three different methods of generating random numbers; each of these, in turn, depends on underlying cryptographic primitives (hash functions/ciphers). The ST author will select the function used and include the specific underlying cryptographic primitives used in the requirement or in the TSS. While any of the identified hash functions (SHA-1, SHA-224, SHA-256, SHA-384, SHA-512) are allowed for Hash_DRBG or HMAC_DRBG, only AES-based implementations for CTR_DRBG are allowed.	The evaluator will perform the following tests: The evaluator will perform 15 trials for the RNG implementation. If the RNG is configurable, the evaluator will perform 15 trials for each configuration. The evaluator will also confirm that the operational guidance contains appropriate instructions for configuring the RNG functionality. 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 will 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 SP 800-90A). 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 will 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 list contains 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 less than or equal to 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 will 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_RBG_EXT.1.2	The deterministic RBG used by the OS shall be seeded by an entropy source that accumulates entropy from a [**selection**: • *software-based noise source*, • *platform-based noise source*] with a minimum of [**selection**: • *128 bits*, • *256 bits*] of entropy at least equal to the greatest security strength (according to NIST SP 800-57) of the keys and hashes that it will generate. **Application Note:** For the first selection in this requirement, the ST author selects 'software-based noise source' if any additional noise sources are used as input to the DRBG. In the second selection in this requirement, the ST author selects the appropriate number of bits of entropy that corresponds to the greatest security strength of the algorithms included in the ST. Security strength is defined in Tables 2 and 3 of NIST SP 800-57A. For example, if the implementation includes 2048-bit RSA (security strength of 112 bits), AES 128 (security strength 128 bits), and HMAC-SHA-256 (security strength 256 bits), then the ST author would select 256 bits.	Documentation shall be produced - and the evaluator will perform the activities - in accordance with and the Clarification to the Entropy Documentation and Assessment Annex. In the future, specific statistical testing (in line with NIST SP 800-90B) will be required to verify the entropy estimates.
FCS_STO_EXT.1.1	The OS shall implement functionality to encrypt sensitive data stored in non-volatile storage and provide interfaces to applications to invoke this functionality. **Application Note:** Sensitive data shall be identified in the TSS by the ST author, and minimally includes credentials and keys. The interface for invoking the functionality could take a variety of forms: it could consist of an API, or simply well-documented conventions for accessing credentials stored as files.	The evaluator will check the TSS to ensure that it lists all persistent sensitive data for which the OS provides a storage capability. For each of these items, the evaluator will confirm that the TSS lists for what purpose it can be used, and how it is stored. The evaluator will confirm that cryptographic operations used to protect the data occur as specified in FCS_COP.1(1). The evaluator will also consult the developer documentation to verify that an interface exists for applications to securely store credentials.
FCS_TLSC_EXT.1.1	The OS shall implement TLS 1.2 (RFC 5246) supporting the following cipher suites: [**selection**: • *TLS_RSA_WITH_AES_128_CBC_SHA as defined in RFC 5246*, • *TLS_RSA_WITH_AES_128_CBC_SHA256 as defined in RFC 5246*, • *TLS_RSA_WITH_AES_256_CBC_SHA256 as defined in RFC 5246*, • *TLS_RSA_WITH_AES_256_CBC_SHA384 as defined in RFC 5288*, • *TLS_DHE_RSA_WITH_AES_128_CBC_SHA256 as defined in RFC 5246*, • *TLS_DHE_RSA_WITH_AES_256_CBC_SHA256 as defined in RFC 5246*, • *TLS_DHE_RSA_WITH_AES_256_GCM_SHA384 as defined in RFC 5288*, • *TLS_ECDHE_ECDSA_WITH_AES_128_CBC_SHA256 as defined in RFC 5289*, • *TLS_ECDHE_ECDSA_WITH_AES_128_GCM_SHA256 as defined in RFC 5289*, • *TLS_ECDHE_ECDSA_WITH_AES_256_CBC_SHA384 as defined in RFC 5289*, • *TLS_ECDHE_ECDSA_WITH_AES_256_GCM_SHA384 as defined in RFC 5289*, • *TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA256 as defined in RFC 5289*, • *TLS_ECDHE_RSA_WITH_AES_128_GCM_SHA256 as defined in RFC 5289*, • *TLS_ECDHE_RSA_WITH_AES_256_CBC_SHA384 as defined in RFC 5289*, • *TLS_ECDHE_RSA_WITH_AES_256_GCM_SHA384 as defined in RFC 5289*]. **Application Note:** The cipher suites to be tested in the evaluated configuration are limited by this requirement. The ST author should select the optional cipher suites that are supported; if there are no cipher suites supported other than the mandatory suites, then "No	The evaluator will check the description of the implementation of this protocol in the TSS to ensure that the cipher suites supported are specified. The evaluator will check the TSS to ensure that the cipher suites specified include those listed for this component. The evaluator will also check the operational guidance to ensure that it contains instructions on configuring the OS so that TLS conforms to the description in the TSS. The evaluator will also perform the following tests: • **Test 1:** The evaluator will establish a TLS connection using each of the cipher suites specified by the requirement. This connection may be established as part of the establishment of a higher-level protocol, e.g., as part of an EAP session. It is sufficient to observe the successful negotiation of a cipher suite to satisfy the intent of the test; it is not necessary to examine the characteristics of the encrypted traffic in an attempt to discern the cipher suite being used (for example, that the cryptographic algorithm is 128-bit AES and not 256-bit AES). • **Test 2:** The evaluator will attempt to establish the connection using a server with a server certificate that contains the Server Authentication purpose in the *extendedKeyUsage* field and verify that a connection is established. The evaluator will then verify that the client rejects an otherwise valid server certificate that lacks the Server Authentication purpose in the *extendedKeyUsage* field and a connection is not established. Ideally, the two certificates should be identical except for the *extendedKeyUsage* field. • **Test 3:** The evaluator will send a server certificate in the TLS connection that does not match the server-selected cipher suite (for example, send a ECDSA certificate while using the TLS_RSA_WITH_AES_128_CBC_SHA cipher suite or send a RSA certificate while using one of the ECDSA cipher suites.) The evaluator will verify that the OS disconnects after receiving the server's Certificate handshake message. • **Test 4:** The evaluator will configure the server to select the TLS_NULL_WITH_NULL_NULL cipher suite and verify that the client denies the connection. • **Test 5:** The evaluator will perform the following modifications to the traffic: ◦ **Test 5.1:** Change the TLS version selected by the server in the Server Hello to a non-supported TLS version (for example 1.3 represented by the two bytes 03 04) and verify

	other cipher suite" should be selected. It is necessary to limit the cipher suites that can be used in an evaluated configuration administratively on the server in the test environment. The Suite B algorithms listed above (RFC 6460) are the preferred algorithms for implementation. TLS_RSA_WITH_AES_128_CBC_SHA is required in order to ensure compliance with RFC 5246. These requirements will be revisited as new TLS versions are standardized by the IETF. If any cipher suites are selected using ECDHE, then FCS_TLSC_EXT.2 is required.	that the client rejects the connection. • **Test 5.2:** Modify at least one byte in the server's nonce in the Server Hello handshake message, and verify that the client rejects the Server Key Exchange handshake message (if using a DHE or ECDHE cipher suite) or that the server denies the client's Finished handshake message. • **Test 5.3:** Modify the server's selected cipher suite in the Server Hello handshake message to be a cipher suite not presented in the Client Hello handshake message. The evaluator will verify that the client rejects the connection after receiving the Server Hello. • **Test 5.4:** If an ECDHE or DHE ciphersuite is selected, modify the signature block in the Server's Key Exchange handshake message, and verify that the client rejects the connection after receiving the Server Key Exchange message. • **Test 5.5:** Modify a byte in the Server Finished handshake message, and verify that the client sends a fatal alert upon receipt and does not send any application data. • **Test 5.6:** Send a garbled message from the Server after the Server has issued the Change Cipher Spec message and verify that the client denies the connection.
FCS_TLSC_EXT.1.2	The OS shall verify that the presented identifier matches the reference identifier according to RFC 6125. **Application Note:** The rules for verification of identity are described in Section 6 of RFC 6125. The reference identifier is established by the user (e.g. entering a URL into a web browser or clicking a link), by configuration (e.g. configuring the name of a mail server or authentication server), or by an application (e.g. a parameter of an API) depending on the OS service. Based on a singular reference identifier's source domain and application service type (e.g. HTTP, SIP, LDAP), the client establishes all reference identifiers which are acceptable, such as a Common Name for the Subject Name field of the certificate and a (case-insensitive) DNS name, URI name, and Service Name for the Subject Alternative Name field. The client then compares this list of all acceptable reference identifiers to the presented identifiers in the TLS server's certificate. The preferred method for verification is the Subject Alternative Name using DNS names, URI names, or Service Names. Verification using the Common Name is required for the purposes of backwards compatibility. Additionally, support for use of IP addresses in the Subject Name or Subject Alternative name is discouraged, as against best practices, but may be implemented. Finally, the client should avoid constructing reference identifiers using wildcards. However, if the presented identifiers include wildcards, the client must follow the best practices regarding matching; these best practices are captured in the evaluation activity.	The evaluator will ensure that the TSS describes the client's method of establishing all reference identifiers from the application-configured reference identifier, including which types of reference identifiers are supported (e.g. Common Name, DNS Name, URI Name, Service Name, or other application-specific Subject Alternative Names) and whether IP addresses and wildcards are supported. The evaluator will ensure that this description identifies whether and the manner in which certificate pinning is supported or used by the OS. The evaluator will verify that the AGD guidance includes instructions for setting the reference identifier to be used for the purposes of certificate validation in TLS. The evaluator will configure the reference identifier according to the AGD guidance and perform the following tests during a TLS connection: • **Test 1:** The evaluator will present a server certificate that does not contain an identifier in either the Subject Alternative Name (SAN) or Common Name (CN) that matches the reference identifier. The evaluator will verify that the connection fails. • **Test 2:** The evaluator will present a server certificate that contains a CN that matches the reference identifier, contains the SAN extension, but does not contain an identifier in the SAN that matches the reference identifier. The evaluator shall verify that the connection fails. The evaluator will repeat this test for each supported SAN type. • **Test 3:** [conditional] If the TOE does not mandate the presence of the SAN extension, the evaluator will present a server certificate that contains a CN that matches the reference identifier and does not contain the SAN extension. The evaluator will verify that the connection succeeds. If the TOE mandates the presence of the SAN extension, this test shall be omitted. • **Test 4:** The evaluator will present a server certificate that contains a CN that does not match the reference identifier but does contain an identifier in the SAN that matches. The evaluator will verify that the connection succeeds. • **Test 5:** The evaluator will perform the following wildcard tests with each supported type of reference identifier: ◦ **Test 5.1:** The evaluator will present a server certificate containing a wildcard that is not in the left-most label of the presented identifier (e.g. foo.*.example.com) and verify that the connection fails. ◦ **Test 5.2:** The evaluator will present a server certificate containing a wildcard in the left-most label but not preceding the public suffix (e.g. *.example.com). The evaluator will configure the reference identifier with a single left-most label (e.g. foo.example.com) and verify that the connection succeeds. The evaluator will configure the reference identifier without a left-most label as in the certificate (e.g. example.com) and verify that the connection fails. The evaluator will configure the reference identifier with two left-most labels (e.g. bar.foo.example.com) and verify that the connection fails. ◦ **Test 5.3:** The evaluator will present a server certificate containing a wildcard in the left-most label immediately preceding the public suffix (e.g. *.com). The evaluator will configure the reference identifier with a single left-most label (e.g. foo.com) and verify that the connection fails. The evaluator will configure the reference identifier with two left-most labels (e.g. bar.foo.com) and verify that the connection fails. • **Test 6:** [conditional] If URI or Service name reference identifiers are supported, the evaluator will configure the DNS name and the service identifier. The evaluator will present a server certificate containing the correct DNS name and service identifier in the URIName or SRVName fields of the SAN and verify that the connection succeeds. The evaluator will repeat this test with the wrong service identifier (but correct DNS name) and verify that the connection fails. • **Test 7:** [conditional] If pinned certificates are supported the evaluator will present a certificate that does not match the pinned certificate and verify that the connection fails.
FCS_TLSC_EXT.1.3	The OS shall only establish a trusted channel if the peer certificate is valid. **Application Note:** Validity is determined by the identifier verification, certificate path, the expiration	The evaluator will use TLS as a function to verify that the validation rules in FIA_X509_EXT.1.1 are adhered to and shall perform the following additional test: • **Test 1:** The evaluator will demonstrate that a peer using a

	date, and the revocation status in accordance with RFC 5280. Certificate validity shall be tested in accordance with testing performed for FIA_X509_EXT.1. For TLS connections, this channel shall not be established if the peer certificate is invalid.	certificate without a valid certification path results in an authenticate failure. Using the administrative guidance, the evaluator will then load the trusted CA certificate(s) needed to validate the peer's certificate, and demonstrate that the connection succeeds. The evaluator then shall delete one of the CA certificates, and show that the connection fails. • **Test 2:** The evaluator will demonstrate that a peer using a certificate which has been revoked results in an authentication failure. • **Test 3:** The evaluator will demonstrate that a peer using a certificate which has passed its expiration date results in an authentication failure. • **Test 4:** the evaluator will demonstrate that a peer using a certificate which does not have a valid identifier shall result in an authentication failure.
FCS_TLSC_EXT.2.1	The OS shall present the Supported Elliptic Curves Extension in the Client Hello with the following NIST curves: [**selection**: *secp256r1*, *secp384r1*, *secp521r1*]. ***This is a selection-based requirement. Its inclusion depends upon selection in .*** **Application Note:** This requirement does not limit the elliptic curves the client may propose for authentication and key agreement. Rather, it asks the ST author to define which of the NIST curves from FCS_COP.1(3) and FCS_CKM.1 and FCS_CKM.2 the TOE supports. This extension is required for clients supporting Elliptic Curve cipher suites.	The evaluator will verify that TSS describes the supported Elliptic Curves Extension and whether the required behavior is performed by default or may be configured. If the TSS indicates that the supported Elliptic Curves Extension must be configured to meet the requirement, the evaluator will verify that AGD guidance includes configuration of the supported Elliptic Curves Extension. The evaluator will also perform the following test: The evaluator will configure a server to perform ECDHE key exchange using each of the TOE's supported curves and shall verify that the TOE successfully connects to the server.
FCS_TLSC_EXT.3.1	The OS shall present the signature_algorithms extension in the Client Hello with the supported_signature_algorithms value containing the following hash algorithms: [**selection**: *SHA256*, *SHA384*, *SHA512*] and no other hash algorithms. ***This is currently an objective requirement.*** **Application Note:** This requirement limits the hashing algorithms supported for the purpose of digital signature verification by the client and limits the server to the supported hashes for the purpose of digital signature generation by the server. The signature_algorithm extension is only supported by TLS 1.2.	The evaluator will verify that TSS describes the signature_algorithm extension and whether the required behavior is performed by default or may be configured. If the TSS indicates that the signature_algorithm extension must be configured to meet the requirement, the evaluator will verify that AGD guidance includes configuration of the signature_algorithm extension. The evaluator will also perform the following test: The evaluator will configure the server to send a certificate in the TLS connection that is not supported according to the Client's HashAlgorithm enumeration within the signature_algorithms extension (for example, send a certificate with a SHA-1 signature). The evaluator will verify that the OS disconnects after receiving the server's Certificate handshake message.
FCS_TLSC_EXT.4.1	The OS shall support mutual authentication using X.509v3 certificates. ***This is an optional requirement. It may be required by Extended Packages of this Protection Profile.*** **Application Note:** The use of X.509v3 certificates for TLS is addressed in FIA_X509_EXT.2.1. This requirement adds that a client must be capable of presenting a certificate to a TLS server for TLS mutual authentication.	The evaluator will ensure that the TSS description required per FIA_X509_EXT.2.1 includes the use of client-side certificates for TLS mutual authentication. The evaluator will verify that the AGD guidance required per FIA_X509_EXT.2.1 includes instructions for configuring the client-side certificates for TLS mutual authentication. The evaluator will also perform the following test: • **Test 1:** The evaluator will establish a connection to a peer server that is not configured for mutual authentication (i.e. does not send Server's Certificate Request (type 13) message). The evaluator observes negotiation of a TLS channel and confirms that the TOE did not send Client's Certificate message (type 11) during handshake. • **Test 2:** The evaluator will establish a connection to a peer server with a shared trusted root that is configured for mutual authentication (i.e. it sends Server's Certificate Request (type 13) message). The evaluator observes negotiation of a TLS channel and confirms that the TOE responds with a non-empty Client's Certificate message (type 11) and Certificate Verify (type 15) messages.
FDP_ACF_EXT.1.1	The OS shall implement access controls which can prohibit unprivileged users from accessing files and directories owned by other users. **Application Note:** Effective protection by access controls may also depend upon system configuration. This requirement is designed to ensure that, for example, files and directories owned by one user in a multi user system can be protected from access by another user in that system.	The evaluator will confirm that the TSS comprehensively describes the access control policy enforced by the OS. The description must include the rules by which accesses to particular files and directories are determined for particular users. The evaluator will inspect the TSS to ensure that it describes the access control rules in such detail that given any possible scenario between a user and a file governed by the OS the access control decision is unambiguous. The evaluator will create two new standard user accounts on the system and conduct the following tests: • **Test 1:** The evaluator will authenticate to the system as the first user and create a file within that user's home directory. The evaluator will then log off the system and log in as the second user. The evaluator will then attempt to read the file created in the first user's home directory. The evaluator will ensure that the read attempt is denied. • **Test 2:** The evaluator will authenticate to the system as the first user and create a file within that user's home directory. The evaluator will then log off the system and log in as the second user. The evaluator will then attempt to modify the file created in the first user's home directory. The evaluator will ensure that the modification is denied. • **Test 3:** The evaluator will authenticate to the system as the first user and create a file within that user's user directory. The evaluator will then log off the system and log in as the second user. The evaluator will then attempt to delete the file created in the first user's home directory. The evaluator will ensure that the deletion is denied. • **Test 4:** The evaluator will authenticate to the system as the first user. The evaluator will attempt to create a file in the second user's

		home directory. The evaluator will ensure that the creation of the file is denied. • **Test 5:** The evaluator will authenticate to the system as the first user and attempt to modify the file created in the first user's home directory. The evaluator will ensure that the modification of the file is accepted. • **Test 6:** The evaluator will authenticate to the system as the first user and attempt to delete the file created in the first user's directory. The evaluator will ensure that the deletion of the file is accepted.
FDP_IFC_EXT.1.1	The OS shall [**selection**: • *provide an interface which allows a VPN client to protect all IP traffic using IPsec*, • *provide a VPN client which can protects all IP traffic using IPsec*] with the exception of IP traffic required to establish the VPN connection and [**selection**: *signed updates directly from the OS vendor, no other traffic*] . ***This is an optional requirement. It may be required by Extended Packages of this Protection Profile.*** **Application Note:** Typically, the traffic required to establish the VPN connection is referred to as "Control Plane" traffic, whereas the IP traffic protected by the IPsec VPN is referred to as "Data Plane" traffic. All Data Plane traffic must flow through the VPN connection and the VPN must not split-tunnel. If no native IPsec client is validated or third-party VPN clients may also implement the required Information Flow Control, the first option shall be selected. In these cases, the TOE provides an API to third-party VPN clients that allows them to configure the TOE's network stack to perform the required Information Flow Control. The ST author shall select the second option if the TSF implements a native VPN client (IPsec is selected in FTP_ITC_EXT.1). If the native VPN client is to be validated (IPsec is selected in FTP_ITC_EXT.1 and the TSF is validated against the *EP for IPsec Virtual Private Network (VPN) Clients*), the ST author shall also include FDP_IFC_EXT.1 from this package. In the future, this requirement may also make a distinction between the current requirement (which requires that when the IPsec trusted channel is enabled, all traffic from the TSF is routed through that channel) and having an option to force the establishment of an IPsec trusted channel to allow any communication by the TSF.	The evaluator will verify that the TSS section of the ST describes the routing of IP traffic when a VPN client is enabled. The evaluator will ensure that the description indicates which traffic does not go through the VPN and which traffic does, and that a configuration exists for each in which only the traffic identified by the ST author as necessary for establishing the VPN connection (IKE traffic and perhaps HTTPS or DNS traffic) is not encapsulated by the VPN protocol (IPsec). The evaluator will perform the following test: • **Test 1:** ◦ **Step 1:** The evaluator will enable a network connection. The evaluator will sniff packets while performing running applications that use the network such as web browsers and email clients. The evaluator will verify that the sniffer captures the traffic generated by these actions, turn off the sniffing tool, and save the session data. ◦ **Step 2:** The evaluator will configure an IPsec VPN client that supports the routing specified in this requirement. The evaluator will turn on the sniffing tool, establish the VPN connection, and perform the same actions with the device as performed in the first step. The evaluator will verify that the sniffing tool captures traffic generated by these actions, turn off the sniffing tool, and save the session data. ◦ **Step 3:** The evaluator will examine the traffic from both step one and step two to verify that all non-excepted Data Plane traffic in Step 2 is encapsulated by IPsec. The evaluator will examine the Security Parameter Index (SPI) value present in the encapsulated packets captured in Step 2 from the TOE to the Gateway and shall verify this value is the same for all actions used to generate traffic through the VPN. Note that it is expected that the SPI value for packets from the Gateway to the TOE is different than the SPI value for packets from the TOE to the Gateway. ◦ **Step 4:** The evaluator will perform a ping on the TOE host on the local network and verify that no packets sent are captured with the sniffer. The evaluator will attempt to send packets to the TOE outside the VPN tunnel (i.e. not through the VPN gateway), including from the local network, and verify that the TOE discards them.
FMT_MOF_EXT.1.1	The OS shall restrict the ability to perform the function indicated in the "Administrator" column in FMT_SMF_EXT.1.1 to the administrator. **Application Note:** The functions with an "X" in the "Administrator" column must be restricted to (or overridden by) the administrator in the TOE. The functions with an "O" in the "Administrator" column may be restricted to (or overridden by) the administrator when implemented in the TOE at the discretion of the ST author. For such functions, the ST author indicates this by replacing an "O" with an "X" in the ST.	The evaluator will verify that the TSS describes those management functions that are restricted to Administrators, including how the user is prevented from performing those functions, or not able to use any interfaces that allow access to that function. The evaluator will also perform the following test. • **Test 1:** For each function that is indicated as restricted to the administrator, the evaluation shall perform the function as an administrator, as specified in the Operational Guidance, and determine that it has the expected effect as outlined by the Operational Guidance and the SFR. The evaluator will then perform the function (or otherwise attempt to access the function) as a non-administrator and observe that they are unable to invoke that functionality.
FMT_SMF_EXT.1.1	The OS shall be capable of performing the following management functions: **Management Function** / **Administrator** / **User** Enable/disable [**selection**: *screen lock*, *session timeout*] — X — O Configure [**selection**: *screen lock*, *session*] inactivity timeout — X — O Configure local audit storage capacity — O — O Configure minimum password length — O — O Configure minimum number of special characters in password — O — O Configure minimum number of numeric characters in password — O — O Configure minimum number of uppercase characters in password — O — O Configure minimum number of lowercase characters in password — O — O Configure lockout policy — O — O	The evaluator will verify that every management function captured in the ST is described in the operational guidance and that the description contains the information required to perform the management duties associated with the management function. The evaluator will test the OS's ability to provide the management functions by configuring the operating system and testing each option selected from above. The evaluator is expected to test these functions in all the ways in which the ST and guidance documentation state the configuration can be managed.

	for unsuccessful authentication attempts through [**selection**: *timeouts between attempts*, *limiting number of attempts during a time period*]		
	Configure host-based firewall	O	O
	Configure name/address of directory server with which to bind	O	O
	Configure name/address of remote management server from which to receive management settings	O	O
	Configure name/address of audit/logging server to which to send audit/logging records	O	O
	Configure audit rules	O	O
	Configure name/address of network time server	O	O
	Enable/disable automatic software update	O	O
	Configure WiFi interface	O	O
	Enable/disable Bluetooth interface	O	O
	Enable/disable [**assignment**: *list of other external interfaces*]	O	O
	[**assignment**: *list of other management functions to be provided by the TSF*]	O	O

.

Application Note: The ST should indicate which of the optional management functions are implemented in the TOE. This can be done by copying the above table into the ST and adjusting the "Administrator" and "User" columns to "X" according to which capabilities are present or not present, and for which privilege level. The Application Note for FMT_MOF_EXT.1 explains how to indicate Administrator or User capability.

The terms "Administrator" and "User" are defined in . The intent of this requirement is to ensure that the ST is populated with the relevant management functions that are provided by the OS.

Sophisticated account management policies, such as intricate password complexity requirements and handling of temporary accounts, are a function of directory servers. The OS can enroll in such account management and enable the overall information system to achieve such policies by binding to a directory server.

FPT_ACF_EXT.1.1	The OS shall implement access controls which prohibit unprivileged users from modifying: • Kernel and its drivers/modules • Security audit logs • Shared libraries • System executables • System configuration files • [**assignment**: *other objects*] .	The evaluator will confirm that the TSS specifies the locations of kernel drivers/modules, security audit logs, shared libraries, system executables, and system configuration files. Every file does not need to be individually identified, but the system's conventions for storing and protecting such files must be specified. The evaluator will create an unprivileged user account. Using this account, the evaluator will ensure that the following tests result in a negative outcome (i.e., the action results in the OS denying the evaluator permission to complete the action): • **Test 1:** The evaluator will attempt to modify all kernel drivers and modules. • **Test 2:** The evaluator will attempt to modify all security audit logs generated by the logging subsystem. • **Test 3:** The evaluator will attempt to modify all shared libraries that are used throughout the system. • **Test 4:** The evaluator will attempt to modify all system executables. • **Test 5:** The evaluator will attempt to modify all system configuration files. • **Test 6:** The evaluator will attempt to modify any additional components selected.
FPT_ACF_EXT.1.2	The OS shall implement access controls which prohibit unprivileged users from reading: • Security audit logs • System-wide credential repositories • [**assignment**: *list of other objects*] . **Application Note:** "Credential repositories" refer, in	The evaluator will create an unprivileged user account. Using this account, the evaluator will ensure that the following tests result in a negative outcome (i.e., the action results in the OS denying the evaluator permission to complete the action): • **Test 1:** The evaluator will attempt to read security audit logs generated by the auditing subsystem • **Test 2:** The evaluator will attempt to read system-wide credential repositories • **Test 3:** The evaluator will attempt to read any other object

	this case, to structures containing cryptographic keys or passwords.	specified in the assignment
FPT_ASLR_EXT.1.1	The OS shall always randomize process address space memory locations with [**selection**: *8*, ***[assignment**: number greater than 8]*] bits of entropy except for [**assignment**: *list of explicit exceptions*].	The evaluator will select 3 executables included with the TSF. If the TSF includes a web browser it must be selected. If the TSF includes a mail client it must be selected. For each of these apps, the evaluator will launch the same executables on two separate instances of the OS on identical hardware and compare all memory mapping locations. The evaluator will ensure that no memory mappings are placed in the same location. If the rare chance occurs that two mappings are the same for a single executable and not the same for the other two, the evaluator will repeat the test with that executable to verify that in the second test the mappings are different. This test can also be completed on the same hardware and rebooting between application launches.
FPT_SBOP_EXT.1.1	The OS shall [**selection**: *employ stack-based buffer overflow protections*, *not store parameters/variables in the same data structures as control flow values*]. **Application Note:** Many OSes store control flow values (i.e. return addresses) in stack data structures that also contain parameters and variables. For these OSes, it is expected that most of the OS, to include the kernel, libraries, and application software from the OS vendor be compiled with stack-based buffer overflow protection enabled. OSes that store parameters and variables separately from control flow values do not need additional stack protections.	For stack-based OSes, the evaluator will determine that the TSS contains a description of stack-based buffer overflow protections used by the OS. These are referred to by a variety of terms, such as stack cookie, stack guard, and stack canaries. The TSS must include a rationale for any binaries that are not protected in this manner. The evaluator will also preform the following test: • **Test 1:** The evaluator will inventory the kernel, libraries, and application binaries to determine those that do not implement stack-based buffer overflow protections. This list should match up with the list provided in the TSS. For OSes that store parameters/variables separately from control flow values, the evaluator will verify that the TSS describes what data structures control values, parameters, and variables are stored. The evaluator will also ensure that the TSS includes a description of the safeguards that ensure parameters and variables do not intermix with control flow values.
FPT_SRP_EXT.1.1	The OS shall restrict execution to only programs which match an administrator-specified [**selection**: • *file path*, • *file digital signature*, • *version*, • *hash*, • *[**assignment**: other characteristics]*] . ***This is currently an objective requirement.*** **Application Note:** The assignment permits implementations which provide a low level of granularity such as a volume. The restriction is only against direct execution of executable programs. It does not forbid interpreters which may take data as an input, even if this data can subsequently result in arbitrary computation.	For each selection specified in the ST, the evaluator will ensure that the corresponding tests result in a negative outcome (i.e., the action results in the OS denying the evaluator permission to complete the action): • **Test 1:** The evaluator will configure the OS to only allow code execution from the core OS directories. The evaluator will then attempt to execute code from a directory that is in the allowed list. The evaluator will ensure that the code they attempted to execute has been executed. • **Test 2:** The evaluator will configure the OS to only allow code execution from the core OS directories. The evaluator will then attempt to execute code from a directory that is not in the allowed list. The evaluator will ensure that the code they attempted to execute has not been executed. • **Test 3:** The evaluator will configure the OS to only allow code that has been signed by the OS vendor to execute. The evaluator will then attempt to execute code signed by the OS vendor. The evaluator will ensure that the code they attempted to execute has been executed. • **Test 4:** The evaluator will configure the OS to only allow code that has been signed by the OS vendor to execute. The evaluator will then attempt to execute code signed by another digital authority. The evaluator will ensure that the code they attempted to execute has not been executed. • **Test 5:** The evaluator will configure the OS to allow execution of a specific application based on version. The evaluator will then attempt to execute the same version of the application. The evaluator will ensure that the code they attempted to execute has been executed. • **Test 6:** The evaluator will configure the OS to allow execution of a specific application based on version. The evaluator will then attempt to execute an older version of the application. The evaluator will ensure that the code they attempted to execute has not been executed. • **Test 7:** The evaluator will configure the OS to allow execution based on the hash of the application executable. The evaluator will then attempt to execute the application with the matching hash. The evaluator will ensure that the code they attempted to execute has been executed. • **Test 8:** The evaluator will configure the OS to allow execution based on the hash of the application executable. The evaluator will modify the application in such a way that the application hash is changed. The evaluator will then attempt to execute the application with the matching hash. The evaluator will ensure that the code they attempted to execute has not been executed.
FPT_TST_EXT.1.1	The OS shall verify the integrity of the bootchain up through the OS kernel and [**selection**: • *all executable code stored in mutable media*, • *[**assignment**: list of other executable code]*, • *no other executable code*] prior to its execution through the use of [**selection**: • *a digital signature using a hardware-protected asymmetric key and X.509 certificates*, • *a hardware-protected hash*] . **Application Note:** The bootchain of the OS is the sequence of software, to include the OS loader, the kernel, system drivers or modules, and system files, which ultimately result in loading the OS. The first part of the OS, usually referred to as the first-stage bootloader, must be loaded by the platform. Assessing its integrity, while critical, is the platform's responsibility; and therefore outside the scope of this PP.	The evaluator will verify that the TSS section of the ST includes a comprehensive description of the boot procedures, including a description of the entire bootchain, for the TSF. The evaluator will ensure that the OS cryptographically verifies each piece of software it loads in the bootchain to include bootloaders and the kernel. Software loaded for execution directly by the platform (e.g. first-stage bootloaders) is out of scope. For each additional category of executable code verified before execution, the evaluator will verify that the description in the TSS describes how that software is cryptographically verified. The evaluator will verify that the TSS contains a description of the protection afforded to the mechanism performing the cryptographic verification. The evaluator will also perform the following tests: • **Test 1:** The evaluator will perform actions to cause TSF software to load and observe that the integrity mechanism does not flag any executables as containing integrity errors and that the OS properly boots. • **Test 2:** The evaluator will modify a TSF executable that is part of

	All software loaded after this stage is potentially within the control of the OS and is in scope. The verification may be transitive in nature: a hardware-protected public key or hash may be used to verify the mutable bootloader code which contains a key or hash used by the bootloader to verify the mutable OS kernel code, which contains a key or hash to verify the next layer of executable code, and so on. However, the way in which the hardware stores and protects these keys is out of scope. If all executable code (including bootloader(s), kernel, device drivers, pre-loaded applications, user-loaded applications, and libraries) is verified, "all executable code stored in mutable media" should be selected.	the bootchain verified by the TSF (i.e. Not the first-stage bootloader) and attempt to boot. The evaluator will ensure that an integrity violation is triggered and the OS does not boot (Care must be taken so that the integrity violation is determined to be the cause of the failure to load the module, and not the fact that in such a way to invalidate the structure of the module.). • **Test 3:** If the ST author indicates that the integrity verification is performed using a public key, the evaluator will verify that the update mechanism includes a certificate validation according to FIA_X509_EXT.1.
FPT_TUD_EXT.1.1	The OS shall provide the ability to check for updates to the OS software itself. **Application Note:** This requirement is about the ability to check for the availability of authentic updates, while the installation of authentic updates is covered by FPT_TUD_EXT.1.2.	The evaluator will check for an update using procedures described in the documentation and verify that the OS provides a list of available updates. Testing this capability may require installing and temporarily placing the system into a configuration in conflict with secure configuration guidance which specifies automatic update. (The evaluator is also to ensure that this query occurs over a trusted channel as described in FTP_ITC_EXT.1.)
FPT_TUD_EXT.1.2	The OS shall cryptographically verify updates to itself using a digital signature prior to installation using schemes specified in FCS_COP.1(3).	For the following tests, the evaluator will initiate the download of an update and capture the update prior to installation. The download could originate from the vendor's website, an enterprise-hosted update repository, or another system (e.g. network peer). All supported origins for the update must be indicated in the TSS and evaluated. • **Test 1:** The evaluator will ensure that the update has a digital signature belonging to the vendor prior to its installation. The evaluator will modify the downloaded update in such a way that the digital signature is no longer valid. The evaluator will then attempt to install the modified update. The evaluator will ensure that the OS does not install the modified update. • **Test 2:** The evaluator will ensure that the update has a digital signature belonging to the vendor. The evaluator will then attempt to install the update (or permit installation to continue). The evaluator will ensure that the OS successfully installs the update.
FPT_TUD_EXT.2.1	The OS shall provide the ability to check for updates to application software. **Application Note:** This requirement is about the ability to check for authentic updates, while the actual installation of such updates is covered by FPT_TUD_EXT.2.2.	The evaluator will check for updates to application software using procedures described in the documentation and verify that the OS provides a list of available updates. Testing this capability may require temporarily placing the system into a configuration in conflict with secure configuration guidance which specifies automatic update. (The evaluator is also to ensure that this query occurs over a trusted channel as described in FTP_ITC_EXT.1.)
FPT_TUD_EXT.2.2	The OS shall cryptographically verify the integrity of updates to applications using a digital signature specified by FCS_COP.1(3) prior to installation.	The evaluator will initiate an update to an application. This may vary depending on the application, but it could be through the application vendor's website, a commercial app store, or another system. All origins supported by the OS must be indicated in the TSS and evaluated. However, this only includes those mechanisms for which the OS is providing a trusted installation and update functionality. It does not include user or administrator-driven download and installation of arbitrary files. • **Test 1:** The evaluator will ensure that the update has a digital signature which chains to the OS vendor or another trusted root managed through the OS. The evaluator will modify the downloaded update in such a way that the digital signature is no longer valid. The evaluator will then attempt to install the modified update. The evaluator will ensure that the OS does not install the modified update. • **Test 2:** The evaluator will ensure that the update has a digital signature belonging to the OS vendor or another trusted root managed through the OS. The evaluator will then attempt to install the update. The evaluator will ensure that the OS successfully installs the update.
FPT_W^X_EXT.1.1	The OS shall prevent allocation of any memory region with both write and execute permissions except for [**assignment**: *list of exceptions*]. ***This is currently an objective requirement.*** **Application Note:** Requesting a memory mapping with both write and execute permissions subverts the platform protection provided by DEP. If the OS provides no exceptions (such as for just-in-time compilation), then "no exceptions" should be indicated in the assignment. Full realization of this requirement requires hardware support, but this is commonly available.	The evaluator will inspect the vendor-provided developer documentation and verify that no memory-mapping can be made with write and execute permissions except for the cases listed in the assignment. The evaluator will also perform the following tests. • **Test 1:** The evaluator will acquire or construct a test program which attempts to allocate memory that is both writable and executable. The evaluator will run the program and confirm that it fails to allocate memory that is both writable and executable. • **Test 2:** The evaluator will acquire or construct a test program which allocates memory that is executable and then subsequently requests additional write/modify permissions on that memory. The evaluator will run the program and confirm that at no time during the lifetime of the process is the memory both writable and executable. • **Test 3:** The evaluator will acquire or construct a test program which allocates memory that is writable and then subsequently requests additional execute permissions on that memory. The evaluator will run the program and confirm that at no time during the lifetime of the process is the memory both writable and executable.
FAU_GEN.1.1	The **OS** shall be able to generate an audit record of the following auditable events: a. Start-up and shut-down of the audit functions; b. All auditable events for the **not specified** level of audit; and c. ◦ **Authentication events (Success/Failure);** ◦ **Use of privileged/special rights**	The evaluator will check the administrative guide and ensure that it lists all of the auditable events. The evaluator will check to make sure that every audit event type selected in the ST is included. The evaluator will test the OS's ability to correctly generate audit records by having the TOE generate audit records for the events listed in the ST. This should include all instance types of an event specified. When verifying the test results, the evaluator will ensure the audit records generated during testing match the format specified in the administrative guide, and

	events (Successful and unsuccessful security, audit, and configuration changes); ◦ **Privilege or role escalation events (Success/Failure);** ◦ **[selection:** ▪ ***File and object events (Successful and unsuccessful attempts to create, access, delete, modify, modify permissions)*****,** ▪ ***User and Group management events (Successful and unsuccessful add, delete, modify, disable, enable, and credential change)*****,** ▪ ***Audit and log data access events (Success/Failure)*****,** ▪ ***Cryptographic verification of software (Success/Failure)*****,** ▪ ***Attempted application invocation with arguments (Success/Failure e.g. due to software restriction policy)*****,** ▪ ***System reboot, restart, and shutdown events (Success/Failure)*****,** ▪ ***Kernel module loading and unloading events (Success/Failure)*****,** ▪ ***Administrator or root-level access events (Success/Failure)*****,** ▪ ***[assignment: other specifically defined auditable events].*** **]** .	that the fields in each audit record have the proper entries.
FAU_GEN.1.2	The **OS** shall record within each audit record at least the following information: a. Date and time of the event, type of event, subject identity (if applicable), and outcome (success or failure) of the event; and b. For each audit event type, based on the auditable event definitions of the functional components included in the PP/ST, [**assignment**: *other audit relevant information*] . **Application Note:** The term *subject* here is understood to be the user that the process is acting on behalf of. If no auditable event definitions of functional components are provided, then no additional audit-relevant information is required.	The evaluator will check the administrative guide and ensure that it provides a format for audit records. Each audit record format type must be covered, along with a brief description of each field. The evaluator will ensure that the fields contains the information required. The evaluator shall test the OS's ability to correctly generate audit records by having the TOE generate audit records for the events listed in the ST. The evaluator will ensure the audit records generated during testing match the format specified in the administrative guide, and that the fields in each audit record provide the required information.
FIA_AFL.1.1	The **OS** shall detect when [**selection**: • *[**assignment**: positive integer number],* • *an administrator configurable positive integer within [**assignment**: range of acceptable values]*] unsuccessful authentication attempts occur related to **events with [selection:** • ***authentication based on user name and password*****,** • ***authentication based on user name and a PIN that releases an asymmetric key stored in OE-protected storage*****,** • ***authentication based on X.509 certificates*** **]** .	The evaluator will set an administrator-configurable threshold for failed attempts, or note the ST-specified assignment. The evaluator will then (per selection) repeatedly attempt to authenticate with an incorrect password, PIN, or certificate until the number of attempts reaches the threshold. Note that the authentication attempts and lockouts must also be logged as specified in FAU_GEN.1.
FIA_AFL.1.2	When the defined number of unsuccessful authentication attempts for an account has been **met**, the **OS** shall: **[selection: *Account Lockout*, *Account Disablement*, *Mandatory Credential Reset*, *[assignment: list of actions]*]** . **Application Note:** The action to be taken shall be populated in the assignment of the ST and defined in the administrator guidance.	• **Test 1:** The evaluator will attempt to authenticate repeatedly to the system with a known bad password. Once the defined number of failed authentication attempts has been reached the evaluator will ensure that the account that was being used for testing has had the actions detailed in the assignment list above applied to it. The evaluator will ensure that an event has been logged to the security event log detailing that the account has had these actions applied. • **Test 2:** The evaluator will attempt to authenticate repeatedly to the system with a known bad certificate. Once the defined number of failed authentication attempts has been reached the evaluator will ensure that the account that was being used for testing has had the actions detailed in the assignment list above applied to it. The evaluator will ensure that an event has been logged to the security event log detailing that the account has had these actions applied. • **Test 3:** The evaluator will attempt to authenticate repeatedly to the system using both a bad password and a bad certificate. Once the defined number of failed authentication attempts has been reached the evaluator will ensure that the account that was being used for testing has had the actions detailed in the assignment list above applied to it. The evaluator will ensure that an event has been logged to the security event log detailing that the account has had these actions applied.
FIA_UAU.5.1	The **OS** shall provide the following authentication mechanisms **[selection:** • ***authentication based on user name and password*****,**	If user name and password authentication is selected, the evaluator will configure the OS with a known user name and password and conduct the following tests: • **Test 1:** The evaluator will attempt to authenticate to the OS using

	• ***authentication based on user name and a PIN that releases an asymmetric key stored in OE-protected storage,*** • ***authentication based on X.509 certificates,*** • ***for use in SSH only, SSH public key-based authentication as specified by the EP for Secure Shell*** **]** to support user authentication. **Application Note:** The "for use in SSH only, SSH public key-based authentication as specified by the EP for Secure Shell" selection can only be included, and must be included, if FTP_ITC_EXT.1.1 selects "SSH as conforming to the EP for Secure Shell".	the known user name and password. The evaluator will ensure that the authentication attempt is successful. • **Test 2:** The evaluator will attempt to authenticate to the OS using the known user name but an incorrect password. The evaluator will ensure that the authentication attempt is unsuccessful. If user name and PIN that releases an asymmetric key is selected, the evaluator will examine the TSS for guidance on supported protected storage and will then configure the TOE or OE to establish a PIN which enables release of the asymmetric key from the protected storage (such as a TPM, a hardware token, or isolated execution environment) with which the OS can interface. The evaluator will then conduct the following tests: • **Test 1:** The evaluator will attempt to authenticate to the OS using the known user name and PIN. The evaluator will ensure that the authentication attempt is successful. • **Test 2:** The evaluator will attempt to authenticate to the OS using the known user name but an incorrect PIN. The evaluator will ensure that the authentication attempt is unsuccessful. If X.509 certificate authentication is selected, the evaluator will generate an X.509v3 certificate for a user with the Client Authentication Enhanced Key Usage field set. The evaluator will provision the OS for authentication with the X.509v3 certificate. The evaluator will ensure that the certificates are validated by the OS as per FIA_X509_EXT.1.1 and then conduct the following tests: • **Test 1:** The evaluator will attempt to authenticate to the OS using the X.509v3 certificate. The evaluator will ensure that the authentication attempt is successful. • **Test 2:** The evaluator will generate a second certificate identical to the first except for the public key and any values derived from the public key. The evaluator will attempt to authenticate to the OS with this certificate. The evaluator will ensure that the authentication attempt is unsuccessful.
FIA_UAU.5.2	The **OS** shall authenticate any user's claimed identity according to the [**assignment**: *rules describing how the multiple authentication mechanisms provide authentication*]. **Application Note:** For all authentication mechanisms specified in FIA_UAU.5.1, the TSS shall describe the rules as to how each authentication mechanism is used. Example rules are how the authentication mechanism authenticates the user (i.e. how does the TSF verify that the correct password or authentication factor is used), the result of a successful authentication (i.e. is the user input used to derive or unlock a key) and which authentication mechanism can be used at which authentication factor interfaces (i.e. if there are times, for example, after a reboot, that only specific authentication mechanisms can be used). Rules regarding how the authentication factors interact in terms of unsuccessful authentication are covered in FIA_AFL.1.	The evaluator will ensure that the TSS describes each mechanism provided to support user authentication and the rules describing how the authentication mechanism(s) provide authentication. The evaluator will verify that configuration guidance for each authentication mechanism is addressed in the AGD guidance. • **Test 1:** For each authentication mechanism selected, the evaluator will enable that mechanism and verify that it can be used to authenticate the user at the specified authentication factor interfaces. • **Test 2:** For each authentication mechanism rule, the evaluator will ensure that the authentication mechanism(s) behave as documented in the TSS.
FIA_X509_EXT.1.1	The OS shall implement functionality to validate certificates in accordance with the following rules: • RFC 5280 certificate validation and certificate path validation. • The certificate path must terminate with a trusted CA certificate. • The OS shall validate a certificate path by ensuring the presence of the *basicConstraints* extension and that the *CA* flag is set to TRUE for all CA certificates. • The OS shall validate the revocation status of the certificate using [**selection**: *the Online Certificate Status Protocol (OCSP) as specified in RFC 2560, a Certificate Revocation List (CRL) as specified in RFC 5759, an OCSP TLS Status Request Extension (i.e., OCSP stapling) as specified in RFC 6066*]. • The OS shall validate the *extendedKeyUsage* field according to the following rules: ◦ Certificates used for trusted updates and executable code integrity verification shall have the Code Signing purpose (id-kp 3 with OID 1.3.6.1.5.5.7.3.3) in the *extendedKeyUsage* field. ◦ Server certificates presented for TLS shall have the Server Authentication purpose (id-kp 1 with OID 1.3.6.1.5.5.7.3.1) in the *extendedKeyUsage* field. ◦ Client certificates presented for TLS shall have the Client Authentication purpose (id-kp 2 with OID 1.3.6.1.5.5.7.3.2) in the *extendedKeyUsage* field. ◦ S/MIME certificates presented for email encryption and signature shall have the Email Protection purpose (id-kp 4 with OID 1.3.6.1.5.5.7.3.4) in the *extendedKeyUsage* field. ◦ OCSP certificates presented for OCSP responses shall have the OCSP Signing purpose (id-kp 9 with OID 1.3.6.1.5.5.7.3.9) in the *extendedKeyUsage* field. ◦ (Conditional) Server certificates presented for EST shall have the CMC Registration Authority (RA) purpose (id-kp-cmcRA with	The evaluator will ensure the TSS describes where the check of validity of the certificates takes place. The evaluator ensures the TSS also provides a description of the certificate path validation algorithm. The tests described must be performed in conjunction with the other certificate services evaluation activities, including the functions in FIA_X509_EXT.2.1. The tests for the *extendedKeyUsage* rules are performed in conjunction with the uses that require those rules. The evaluator will create a chain of at least four certificates: the node certificate to be tested, two Intermediate CAs, and the self-signed Root CA. • **Test 1:** The evaluator will demonstrate that validating a certificate without a valid certification path results in the function failing. The evaluator will then load a certificate or certificates as trusted CAs needed to validate the certificate to be used in the function, and demonstrate that the function succeeds. The evaluator shall then delete one of the certificates, and show that the function fails. • **Test 2:** The evaluator will demonstrate that validating an expired certificate results in the function failing. • **Test 3:** The evaluator will test that the OS can properly handle revoked certificates--conditional on whether CRL, OCSP, or OCSP stapling is selected; if multiple methods are selected, then a test shall be performed for each method. The evaluator will test revocation of the node certificate and revocation of the intermediate CA certificate (i.e. the intermediate CA certificate should be revoked by the root CA). The evaluator will ensure that a valid certificate is used, and that the validation function succeeds. The evaluator then attempts the test with a certificate that has been revoked (for each method chosen in the selection) to ensure when the certificate is no longer valid that the validation function fails. • **Test 4:** If either OCSP option is selected, the evaluator will configure the OCSP server or use a man-in-the-middle tool to present a certificate that does not have the OCSP signing purpose and verify that validation of the OCSP response fails. If CRL is selected, the evaluator will configure the CA to sign a CRL with a certificate that does not have the cRLsign key usage bit set, and verify that validation of the CRL fails. • **Test 5:** The evaluator will modify any byte in the first eight bytes of the certificate and demonstrate that the certificate fails to validate. (The certificate should fail to parse correctly.) • **Test 6:** The evaluator will modify any byte in the last byte of the certificate and demonstrate that the certificate fails to validate. (The signature on the certificate should not validate.) • **Test 7:** The evaluator will modify any byte in the public key of the

	OID 1.3.6.1.5.5.7.3.28) in the *extendedKeyUsage* field. . **Application Note:** FIA_X509_EXT.1.1 lists the rules for validating certificates. The ST author shall select whether revocation status is verified using OCSP or CRLs. FIA_X509_EXT.2 requires that certificates are used for HTTPS, TLS and DTLS; this use requires that the *extendedKeyUsage* rules are verified.	certificate and demonstrate that the certificate fails to validate. (The signature on the certificate should not validate.)
FIA_X509_EXT.1.2	The OS shall only treat a certificate as a CA certificate if the *basicConstraints* extension is present and the CA flag is set to TRUE. **Application Note:** This requirement applies to certificates that are used and processed by the TSF and restricts the certificates that may be added as trusted CA certificates.	The tests described must be performed in conjunction with the other certificate services evaluation activities, including the functions in FIA_X509_EXT.2.1. The evaluator will create a chain of at least four certificates: the node certificate to be tested, two Intermediate CAs, and the self-signed Root CA. • **Test 1:** The evaluator will construct a certificate path, such that the certificate of the CA issuing the OS's certificate does not contain the *basicConstraints* extension. The validation of the certificate path fails. • **Test 2:** The evaluator will construct a certificate path, such that the certificate of the CA issuing the OS's certificate has the CA flag in the *basicConstraints* extension not set. The validation of the certificate path fails. • **Test 3:** The evaluator will construct a certificate path, such that the certificate of the CA issuing the OS's certificate has the CA flag in the *basicConstraints* extension set to TRUE. The validation of the certificate path succeeds.
FIA_X509_EXT.2.1	The OS shall use X.509v3 certificates as defined by RFC 5280 to support authentication for TLS and [**selection**: *DTLS*, *HTTPS*, *[**assignment**: other protocols]*, *no other protocols*] connections.	The evaluator will acquire or develop an application that uses the OS TLS mechanism with an X.509v3 certificate. The evaluator will then run the application and ensure that the provided certificate is used to authenticate the connection. The evaluator will repeat the activity for any other selections listed.
FTA_TAB.1.1	Before establishing a user session, the OS shall display an advisory warning message regarding unauthorized use of the OS. ***This is an optional requirement. It may be required by Extended Packages of this Protection Profile.***	The evaluator will configure the OS, per instructions in the OS manual, to display the advisory warning message "TEST TEST Warning Message TEST TEST". The evaluator will then log out and confirm that the advisory message is displayed before logging in can occur.
FTP_ITC_EXT.1.1	The OS shall use [**selection**: • *TLS as conforming to the Package for Transport Layer Security*, • *DTLS as conforming to the Package for Transport Layer Security*, • *IPsec as conforming to the EP for IPsec VPN Clients*, • *SSH as conforming to the EP for Secure Shell*] to provide a trusted communication channel between itself and authorized IT entities supporting the following capabilities: [**selection**: *audit server*, *authentication server*, *management server*, *[**assignment**: other capabilities]*] that is logically distinct from other communication channels and provides assured identification of its end points and protection of the channel data from disclosure and detection of modification of the channel data. **Application Note:** The ST author must include the security functional requirements for the trusted channel protocol selected in FTP_ITC_EXT.1 in the main body of the ST. If the ST author selects TLS or DTLS, the TSF shall be validated against requirements from the *Package for Transport Layer Security*. The TSF can act as TLS client or server, or both. If the ST author selects IPsec, the TSF shall be validated against the *EP for IPsec Virtual Private Network (VPN) Clients*. If the ST author selects SSH, the TSF shall be validated against the *EP for Secure Shell*.	The evaluator will configure the OS to communicate with another trusted IT product as identified in the second selection. The evaluator will monitor network traffic while the OS performs communication with each of the servers identified in the second selection. The evaluator will ensure that for each session a trusted channel was established in conformance with the protocols identified in the first selection.
FTP_TRP.1.1	The OS shall provide a communication path between itself and [**selection**: *remote*, *local*] users that is logically distinct from other communication paths and provides assured identification of its endpoints and protection of the communicated data from modification and disclosure. **Application Note:** This requirement ensures that all remote administrative actions are protected. Authorized remote administrators must initiate all communication with the OS via a trusted path and all communication with the OS by remote administrators must be performed over this path. The data passed in this trusted communication channel is encrypted as defined in FTP_ITC_EXT.1. If local users access is selected and no unprotected traffic is sent to remote users, then this requirement is met. If remote users access is selected, the ST author must include the security functional requirements for the trusted channel protocol selected in FTP_ITC_EXT.1 in the main body of the ST. This requirement is tested with the evaluation activities for FTP_TRP.1.3.	

FTP_TRP.1.2	The OS shall permit [**selection**: *the TSF*, *local users*, *remote users*] to initiate communication via the trusted path. **Application Note:** This requirement is tested with the evaluation activities for FTP_TRP.1.3.	
FTP_TRP.1.3	The OS shall require use of the trusted path for all remote administrative actions. **Application Note:** This requirement ensures that authorized remote administrators initiate all communication with the OS via a trusted path, and that all communication with the OS by remote administrators is performed over this path. The data passed in this trusted communication channel is encrypted as defined in FTP_ITC_EXT.1. The evaluation activities for this requirement also test requirements FTP_TRP.1.1 and FTP_TRP.1.2.	The evaluator will examine the TSS to determine that the methods of remote OS administration are indicated, along with how those communications are protected. The evaluator will also confirm that all protocols listed in the TSS in support of OS administration are consistent with those specified in the requirement, and are included in the requirements in the ST. The evaluator will confirm that the operational guidance contains instructions for establishing the remote administrative sessions for each supported method. The evaluator will also perform the following tests: • **Test 1:** The evaluator will ensure that communications using each remote administration method is tested during the course of the evaluation, setting up the connections as described in the operational guidance and ensuring that communication is successful. • **Test 2:** For each method of remote administration supported, the evaluator will follow the operational guidance to ensure that there is no available interface that can be used by a remote user to establish a remote administrative sessions without invoking the trusted path. • **Test 3:** The evaluator will ensure, for each method of remote administration, the channel data is not sent in plaintext. • **Test 4:** The evaluator will ensure, for each method of remote administration, modification of the channel data is detected by the OS.

Security Assurance Requirements

ID	Requirement	Assurance Activity
ADV_FSP.1.1D	The developer shall provide a functional specification.	
ADV_FSP.1.2D	The developer shall provide a tracing from the functional specification to the SFRs. **Application Note:** As indicated in the introduction to this section, the functional specification is comprised of the information contained in the AGD_OPE and AGD_PRE documentation. The developer may reference a website accessible to application developers and the evaluator. The evaluation activities in the functional requirements point to evidence that should exist in the documentation and TSS section; since these are directly associated with the SFRs, the tracing in element ADV_FSP.1.2D is implicitly already done and no additional documentation is necessary.	
ADV_FSP.1.1C	The functional specification shall describe the purpose and method of use for each SFR-enforcing and SFR-supporting TSFI.	
ADV_FSP.1.2C	The functional specification shall identify all parameters associated with each SFR-enforcing and SFR-supporting TSFI.	
ADV_FSP.1.3C	The functional specification shall provide rationale for the implicit categorization of interfaces as SFR-non-interfering.	
ADV_FSP.1.4C	The tracing shall demonstrate that the SFRs trace to TSFIs in the functional specification.	
ADV_FSP.1.1E	The evaluator will confirm that the information provided meets all requirements for content and presentation of evidence.	
ADV_FSP.1.2E	The evaluator will determine that the functional specification is an accurate and complete instantiation of the SFRs.	There are no specific evaluation activities associated with these SARs, except ensuring the information is provided. The functional specification documentation is provided to support the evaluation activities described in , 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 evaluation 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.
AGD_OPE.1.1D	The developer shall provide operational user guidance. **Application Note:** The operational user guidance does not have to be contained in a single document. Guidance to users, administrators	

	and application developers can be spread among documents or web pages. Rather than repeat information here, the developer should review the evaluation activities for this component to ascertain the specifics of the guidance that the evaluator will be checking for. This will provide the necessary information for the preparation of acceptable guidance.	
AGD_OPE.1.1C	The operational user guidance shall describe, for each user role, the user-accessible functions and privileges that should be controlled in a secure processing environment, including appropriate warnings. **Application Note:** User and administrator are to be considered in the definition of user role.	
AGD_OPE.1.2C	The operational user guidance shall describe, for each user role, how to use the available interfaces provided by the OS in a secure manner.	
AGD_OPE.1.3C	The operational user guidance shall describe, for each user role, the available functions and interfaces, in particular all security parameters under the control of the user, indicating secure values as appropriate. **Application Note:** This portion of the operational user guidance should be presented in the form of a checklist that can be quickly executed by IT personnel (or end-users, when necessary) and suitable for use in compliance activities. When possible, this guidance is to be expressed in the eXtensible Configuration Checklist Description Format (XCCDF) to support security automation. Minimally, it should be presented in a structured format which includes a title for each configuration item, instructions for achieving the secure configuration, and any relevant rationale.	
AGD_OPE.1.4C	The operational user guidance shall, for each user role, clearly present each type of security-relevant event relative to the user-accessible functions that need to be performed, including changing the security characteristics of entities under the control of the TSF.	
AGD_OPE.1.5C	The operational user guidance shall identify all possible modes of operation of the OS (including operation following failure or operational error), their consequences, and implications for maintaining secure operation.	
AGD_OPE.1.6C	The operational user guidance shall, for each user role, describe the security measures to be followed in order to fulfill the security objectives for the operational environment as described in the ST.	
AGD_OPE.1.7C	The operational user guidance shall be clear and reasonable.	
AGD_OPE.1.1E	The evaluator will confirm that the information provided meets all requirements for content and presentation of evidence.	Some of the contents of the operational guidance are verified by the evaluation activities in and evaluation of the OS according to the . 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.1.1D	The developer shall provide the OS, including its preparative procedures. **Application Note:** As with the operational guidance, the developer should look to the evaluation activities to determine the required content with respect to preparative procedures.	
AGD_PRE.1.1C	The preparative procedures shall	

AGD_PRE.1.1C	The preparative procedures shall describe all the steps necessary for secure acceptance of the delivered OS in accordance with the developer's delivery procedures.	
AGD_PRE.1.2C	The preparative procedures shall describe all the steps necessary for secure installation of the OS and for the secure preparation of the operational environment in accordance with the security objectives for the operational environment as described in the ST.	
AGD_PRE.1.1E	The evaluator will confirm that the information provided meets all requirements for content and presentation of evidence.	
AGD_PRE.1.2E	The evaluator will apply the preparative procedures to confirm that the OS can be prepared securely for operation.	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.
ALC_CMC.1.1D	The developer shall provide the OS and a reference for the OS.	
ALC_CMC.1.1C	The OS shall be labeled with a unique reference. **Application Note:** Unique reference information includes: • OS Name • OS Version • OS Description • Software Identification (SWID) tags, if available	
ALC_CMC.1.1E	The evaluator will confirm that the information provided meets all requirements for content and presentation of evidence.	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.1.1D	The developer shall provide a configuration list for the OS.	
ALC_CMS.1.1C	The configuration list shall include the following: the OS itself; and the evaluation evidence required by the SARs.	
ALC_CMS.1.2C	The configuration list shall uniquely identify the configuration items.	
ALC_CMS.1.1E	The evaluator will confirm that the information provided meets all requirements for content and presentation of evidence.	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 evaluation 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.1.1D	The developer shall provide a description in the TSS of how timely security updates are made to the OS.	
ALC_TSU_EXT.1.2D	The developer shall provide a description in the TSS of how users are notified when updates change security properties or the configuration of the product.	
ALC_TSU_EXT.1.1C	The description shall include the process for creating and deploying security updates for the OS software.	
ALC_TSU_EXT.1.2C	The description shall include the mechanisms publicly available for reporting security issues pertaining to the OS.	
ALC_TSU_EXT.1.1E	The evaluator will confirm that the information provided meets all requirements for content and presentation of evidence.	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.
ATE_IND.1.1D	The developer shall provide the OS for testing.	
ATE_IND.1.1C	The OS shall be suitable for testing.	
ATE_IND.1.1E	The evaluator *shall confirm* that the information provided meets all requirements for content and presentation of evidence.	
ATE_IND.1.2E	The evaluator will test a subset of the TSF to confirm that the TSF operates as specified. **Application Note:** The evaluator will test the OS on the most current fully patched version of the platform.	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 and the body of this PP's evaluation activities. While it is not necessary to have one test case per test listed in an evaluation 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.
AVA_VAN.1.1D	The developer shall provide the OS for testing.	
AVA_VAN.1.1C	The OS shall be suitable for testing.	
AVA_VAN.1.1E	The evaluator will confirm that the information provided meets all requirements for content and presentation of evidence.	
AVA_VAN.1.2E	The evaluator will perform a search of public domain sources to identify potential vulnerabilities in the OS. **Application Note:** Public domain sources include the Common Vulnerabilities and Exposures (CVE) dictionary for publicly-known vulnerabilities. Public domain sources also include sites which provide free checking of files for viruses.	
AVA_VAN.1.3E	The evaluator will conduct penetration testing, based on the identified potential vulnerabilities, to determine that the OS is resistant to attacks performed by an attacker possessing Basic attack potential.	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.

Glossary

Common Criteria (CC)	Common Criteria for Information Technology Security Evaluation.
Common Evaluation Methodology (CEM)	Common Evaluation Methodology for Information Technology Security Evaluation.

Extended Package (EP)	An implementation-independent set of security requirements for a specific subset of products described.
Protection Profile (PP)	An implementation-independent set of security requirements for a category of products.
Security Target (ST)	A set of implementation-dependent security requirements for a specific product.
Target of Evaluation (TOE)	The product under evaluation. In this case, the Operating System as described in section and its supporting documentation.
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 a ST.
Security Functional Requirement (SFR)	A requirement for security enforcement by the TOE.
Security Assurance Requirement (SAR)	A requirement to assure the security of the TOE.

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.
Data At Rest (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.
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.