Tabular Presentation of the

Protection Profile for Application Software

Version: 1.3

2019-03-01

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 Application Software*. 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
	The application shall [**selection**: • *use no DRBG functionality*, • *invoke platform-provided DRBG functionality*, • *implement DRBG functionality*] for its cryptographic operations. **Application Note:** The selection *invoke platform-provided DRBG functionality* should only be chosen for direct invocations of the platform DRBG, calls to platform protocols that may then call the platform's DRBG are not directly using DRBG functionality and should select *use no DRBG functionality*. If *implement DRBG functionality* is chosen, then additional FCS_RBG_EXT.2 elements shall be included in the ST. In this requirement, cryptographic operations include all cryptographic key generation/derivation/agreement, IVs (for certain modes), as well as protocol-specific random values. Cryptographic operations in this requirement refer to the other cryptographic requirements in this PP, not additional functionality that is not in scope.	If **use no DRBG functionality** is selected, the evaluator shall inspect the application and its developer documentation and verify that the application needs no random bit generation services. If **implement DRBG functionality** is selected, the evaluator shall ensure that additional FCS_RBG_EXT.2 elements are included in the ST. If **invoke platform-provided DRBG functionality** is selected, the evaluator performs the following activities. The evaluator shall examine the TSS to confirm that it identifies all functions (as described by the SFRs included in the ST) that obtain random numbers from the platform RBG. The evaluator shall determine that for each of these functions, the TSS states which platform interface (API) is used to obtain the random numbers. The evaluator shall confirm that each of these interfaces corresponds to the acceptable interfaces listed for each platform below. It should be noted that there is no expectation that the evaluators attempt to confirm that the APIs are being used correctly for the functions identified in the TSS; the activity is to list the used APIs and then do an existence check via decompilation. If **invoke platform-provided DRBG functionality** is selected, the following tests shall be performed: The evaluator shall decompile the application binary using a decompiler suitable for the application (TOE). The evaluator shall search the output of the decompiler

ID	Requirement	Assurance Activity
		to determine that, for each API listed in the TSS, that API appears in the output. If the representation of the API does not correspond directly to the strings in the following list, the evaluator shall provide a mapping from the decompiled text to its corresponding API, with a description of why the API text does not directly correspond to the decompiled text and justification that the decompiled text corresponds to the associated API. The following are the per-platform list of acceptable APIs: ***For Android:*** The evaluator shall verify that the application uses at least one of javax.crypto.KeyGenerator class or the java.security.SecureRandom class or /dev/random or /dev/urandom. ***For Windows:*** The evaluator shall verify that rand_s, RtlGenRandom, BCryptGenRandom, or CryptGenRandom API is used for classic desktop applications. The evaluator shall verify the application uses the RNGCryptoServiceProvider class or derives a class from System.Security.Cryptography.RandomNumberGenerator API for Windows Universal Applications. It is only required that the API is called/invoked, there is no requirement that the API be used directly. In future versions of this document, CryptGenRandom may be removed as an option as it is no longer the preferred API per vendor documentation. ***For iOS:*** The evaluator shall verify that the application invokes SecRandomCopyBytes or uses /dev/random directly to acquire random. ***For Linux:*** The evaluator shall verify that the application collects random from /dev/random or /dev/urandom. ***For Solaris:*** The evaluator shall verify that the application collects random from /dev/random. ***For macOS:*** The evaluator shall verify that the application collects random from /dev/random. If invocation of platform-provided functionality is achieved in another way, the evaluator shall ensure the TSS describes how this is carried out, and how it is equivalent to the methods listed here (e.g. higher-level API invokes identical low-level API).
	The application 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)*] ***This is a selection-based requirement. Its inclusion depends upon selection in .*** **Application Note:** This requirement shall be included in STs in which *implement DRBG functionality* is chosen in FCS_RBG_EXT.1.1. The ST author should select the standard to which the RBG services comply (either SP 800-90A or FIPS 140-2 Annex C). 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 (if SP 800-90A is selected), 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 shall perform the following tests, depending on the standard to which the RBG conforms. Implementations Conforming to FIPS 140-2 Annex C. The reference for the tests contained in this section is The Random Number Generator Validation System (RNGVS). The evaluators shall conduct the following two tests. Note that the "expected values" are produced by a reference implementation of the algorithm that is known to be correct. Proof of correctness is left to each Scheme. • **Test 1:** The evaluators shall perform a Variable Seed Test. The evaluators shall provide a set of 128 (Seed, DT) pairs to the TSF RBG function, each 128 bits. The evaluators shall also provide a key (of the length appropriate to the AES algorithm) that is constant for all 128 (Seed, DT) pairs. The DT value is incremented by 1 for each set. The seed values shall have no repeats within the set. The evaluators ensure that the values returned by the TSF match the expected values. • **Test 2:** The evaluators shall perform a Monte Carlo Test. For this test, they supply an initial Seed and DT value to the TSF RBG function; each of these is 128 bits. The evaluators shall also provide a key (of the length appropriate to the AES algorithm) that is constant throughout the test. The evaluators then invoke the TSF RBG 10,000 times, with the DT value being incremented by 1 on each iteration, and the new seed for the subsequent iteration produced as specified in NIST-Recommended Random Number Generator Based on ANSI X9.31 Appendix A.2.4 Using the 3-Key Triple DES and AES Algorithms, Section E.3. The evaluators ensure that the 10,000th value produced matches the expected value. Implementations Conforming to NIST Special Publication 800-90A • **Test 1:** The evaluator shall perform 15 trials for the RNG implementation. If the RNG is configurable, the evaluator shall perform 15 trials for each configuration. The evaluator shall 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 shall generate eight input values for each trial. The first is a count (0 – 14). The next three are entropy input, nonce, and personalization string for the instantiate operation. The next two are additional input and entropy input for the first call to generate. The final two are additional input and entropy input for the second call to generate. These values are randomly generated. “generate one block of random bits” means to generate random bits with number of returned bits equal to the Output Block Length (as defined in NIST 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 shall generate eight input values for each trial. The first is a count (0 – 14). The next three are entropy input, nonce, and personalization string for the instantiate operation. The fifth value is additional input to the first call to generate. The sixth and seventh are additional input and entropy input to the call to reseed. The final value is additional input to the second generate call. The following paragraphs contain more information on some of the input values to be generated/selected by the evaluator. **Entropy input:** the length of the entropy input value must equal the seed length. **Nonce:** If a nonce is supported (CTR_DRBG with no Derivation Function does not use a nonce), the nonce bit length is one-half the seed length. **Personalization string:** The length of the personalization string must be less then 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 shall use personalization strings of two different lengths. If the implementation does not use a personalization string, no value needs to be supplied.

ID	Requirement	Assurance Activity
		Additional input: the additional input bit lengths have the same defaults and restrictions as the personalization string lengths.
	The deterministic RBG shall be seeded by an entropy source that accumulates entropy from a platform-based DRBG and [**selection**: • *a software-based noise source*, • *a hardware-based noise source*, • *no other 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. ***This is a selection-based requirement. Its inclusion depends upon selection in .*** **Application Note:** This requirement shall be included in STs in which *implement DRBG functionality* is chosen in FCS_RBG_EXT.1.1. 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 application's DRBG. Note that the application must use the platform's DRBG to seed its 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) and AES 256 (security strength 256 bits), then the ST author would select 256 bits.	Documentation shall be produced - and the evaluator shall 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.
	The application shall [**selection**: • *generate no asymmetric cryptographic keys*, • *invoke platform-provided functionality for asymmetric key generation*, • *implement asymmetric key generation*]. **Application Note:** If *implement asymmetric key generation* or *invoke platform-provided functionality for asymmetric key generation* is chosen, then additional FCS_CKM.1/1 elements shall be included in the ST.	The evaluator shall inspect the application and its developer documentation to determine if the application needs asymmetric key generation services. If not, the evaluator shall verify the **generate no asymmetric cryptographic keys** selection is present in the ST. Otherwise, the evaluation activities shall be performed as stated in the selection-based requirements.

	The **application** shall [**selection**:
	• *invoke platform-provided functionality*,
	• *implement functionality*
	] **to 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]*** ,
	• *[FFC Schemes]* ***using Diffie-Hellman group 14*** *that meet the following:* ***RFC 3526, Section 3*** ,
	• *[FFC Schemes]* ***using "safe-prime" groups*** *that meet the following:* ***NIST Special Publication 800-56A Revision 3, "Recommendation for Pair-Wise Key Establishment Schemes Using Discrete Logarithm Cryptography"*** *and [**selection**: RFC 3526, RFC 7919]*
	].
	
	This is a selection-based requirement. Its inclusion depends upon selection in .
	
	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.2.1 and selected cryptographic protocols must match the selection. When key generation is used for entity authentication, the public key is expected to be associated with an X.509v3 certificate.
	If the TOE acts as a receiver in the RSA key establishment scheme, the TOE does not need to implement RSA key generation.
		The evaluator shall ensure that the TSS identifies the key sizes supported by the TOE. If the ST specifies more than one scheme, the evaluator shall examine the TSS to verify that it identifies the usage for each scheme.
		
		If the application **invokes platform-provided functionality for asymmetric key generation**, then the evaluator shall examine the TSS to verify that it describes how the key generation functionality is invoked.
		
		The evaluator shall verify that the AGD guidance instructs the administrator how to configure the TOE to use the selected key generation scheme(s) and key size(s) for all uses defined in this PP.
		
		If the application **implements asymmetric key generation**, then the following test activities shall be carried out.
		
		Evaluation Activity Note: The following tests may require the developer to provide access to a developer environment that provides the evaluator with tools that are typically available to end-users of the application.
		
		Key Generation for FIPS PUB 186-4 RSA Schemes
		
		The evaluator shall verify the implementation of RSA Key Generation by the TOE using the Key Generation test. This test verifies the ability of the TSF to correctly produce values for the key components including the public verification exponent e, the private prime factors p and q, the public modulus n and the calculation of the private signature exponent d. 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 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 shall 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 shall have the TSF generate 10 keys pairs for each supported key length nlen and verify:
		• n = p·q,
		• p and q are probably prime according to Miller-Rabin tests,
		• GCD(p-1,e) = 1,
		• GCD(q-1,e) = 1,
		• $2^{16} \le e \le 2^{256}$ and e is an odd integer,
		• $\|p-q\| > 2^{nlen/2 - 100}$,
		• $p \ge 2^{nlen/2 - 1/2}$,

ID	Requirement	Assurance Activity
		• $q > 2^{nlen/2 -1/2}$, • $2^{(nlen/2)} < d < LCM(p-1,q-1)$, • $e \cdot d = 1 \bmod LCM(p-1,q-1)$. **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 shall 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 shall 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 shall 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 shall obtain in response a set of 10 PASS/FAIL values. **Key Generation for Finite-Field Cryptography (FFC)** The evaluator shall verify the implementation of the Parameters Generation and the Key Generation for FFC by the TOE using the Parameter Generation and Key Generation test. This test verifies the ability of the TSF to correctly produce values for the field prime p, the cryptographic prime q (dividing p-1), the cryptographic group generator g, and the calculation of the private key x and public key y. The Parameter generation specifies 2 ways (or methods) to generate the cryptographic prime q and the field prime p: 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 $1 \le x \le q-1$ • len(q) + 64 bit output of RBG, followed by a mod q-1 operation where $1 \le x \le q-1$. The security strength of the RBG must be at least that of the security offered by the FFC parameter set. To test the cryptographic and field prime generation method for the provable primes method 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 shall have the TSF generate 25 parameter sets and key pairs. The evaluator shall verify the correctness of the TSF's implementation by comparing values generated by the TSF with those generated from a known good implementation. Verification must also confirm • $g \neq 0,1$ • q divides p-1 • $g^q \bmod p = 1$ • $g^x \bmod p = y$ for each FFC parameter set and key pair. **Diffie-Hellman Group 14 and FFC Schemes using “safe-prime” groups** Testing for FFC Schemes using Diffie-Hellman group 14 and/or safe-prime groups is done as part of testing in CKM.2.1.
	The **application** shall generate **symmetric** cryptographic keys **using a Random Bit Generator as specified in FCS_RBG_EXT.1** and specified cryptographic key sizes [**selection**: • *128 bit*, • *256 bit*]. ***This is an optional requirement. It may be required by Extended Packages of this Protection Profile.*** **Application Note:** Symmetric keys may be used to generate keys along the key chain.	The evaluator shall review the TSS to determine that it describes how the functionality described by FCS_RBG_EXT.1 is invoked. If the application is relying on random bit generation from the host platform, the evaluator shall verify the TSS includes the name/manufacturer of the external RBG and describes the function call and parameters used when calling the external DRBG function. If different external RBGs are used for different platforms, the evaluator shall verify the TSS identifies each RBG for each platform. Also, the evaluator shall verify the TSS includes a short description of the vendor's assumption for the amount of entropy seeding the external DRBG. The evaluator uses the description of the RBG functionality in FCS_RBG_EXT or documentation available for the operational environment to determine that the key size being requested is identical to the key size and mode to be used for the encryption/decryption of the user data.
	Refinement: A password/passphrase shall perform [Password-based Key Derivation Functions] in accordance with a specified cryptographic algorithm as specified in FCS_COP.1/4, with [assignment: positive integer of 1,000 or more] iterations, and output cryptographic key sizes [**selection**: *128*, *256*] that meet the following **[NIST SP 800-132]**. ***This is a selection-based requirement. Its inclusion depends upon selection in .***	
	The TSF shall generate salts using a RBG that meets FCS_RGB_EXT.1 and with entropy corresponding to the security strength selected for PBKDF in FCS_CKM.1.1/3 ***This is a selection-based requirement. Its inclusion depends upon selection in .*** **Application Note:** This should be included if selected in FCS_STO_EXT.1 Conditioning can be performed using one of the identified hash functions or the process described in NIST SP 800-132; the method used is selected by the ST Author. SP 800-132 requires the use of a pseudo-random function (PRF) consisting of HMAC with an approved hash function. The ST author selects the hash function used, also includes the appropriate requirements for HMAC and the hash function. Appendix A of SP 800-132 recommends setting the iteration count in order to increase the computation needed to derive a key from a password and, therefore,	Support for PBKDF: The evaluator shall examine the password hierarchy TSS to ensure that the formation of all password based derived keys is described and that the key sizes match that described by the ST author. The evaluator shall check that the TSS describes the method by which the password/passphrase is first encoded and then fed to the SHA algorithm. The settings for the algorithm (padding, blocking, etc.) shall be described, and the evaluator shall verify that these are supported by the selections in this component as well as the selections concerning the hash function itself. The evaluator shall verify that the TSS contains a description of how the output of the hash function is used to form the submask that will be input into the function. For the NIST SP 800-132-based conditioning of the password/passphrase, the required evaluation activities will be performed when doing the evaluation activities for the appropriate requirements (FCS_COP.1.1/4). No explicit testing of the formation of the submask from the input password is required. FCS_CKM.1.1/3: The ST author shall provide a description in the TSS

ID	Requirement	Assurance Activity
	increase the workload of performing a password recovery attack. A significantly higher value is recommended to ensure optimal security. This value is expected to increase to a minimum of 10,000 in a future iteration based on SP800-63.	regarding the salt generation. The evaluator shall confirm that the salt is generated using an RBG described in FCS_RBG_EXT.1.
	The application shall [**selection**: *invoke platform-provided functionality*, *implement functionality*] to perform cryptographic key establishment in accordance with a specified cryptographic key establishment method: [**selection**: • ***[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”]*** , • ***[RSA-based key establishment schemes]*** *that meet the following:* ***RSAES-PKCS1-v1_5 as specified in Section 7.2 of RFC 8017, “Public-Key Cryptography Standards (PKCS) #1: RSA Cryptography Specifications Version 2.1”*** , • ***[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”]*** , • ***[Key establishment scheme using Diffie-Hellman group 14]*** *that meets the following:* ***RFC 3526, Section 3*** , • ***[FFC Schemes using “safe-prime” groups]*** *that meet the following:* ***‘NIST Special Publication 800-56A Revision 3, “Recommendation for Pair-Wise Key Establishment Schemes Using Discrete Logarithm Cryptography”*** *and [**selection**: RFC 3526, RFC 7919].*]. ***This is a selection-based requirement. Its inclusion depends upon selection in .*** **Application Note:** The ST author shall select all key establishment schemes used for the selected cryptographic protocols. TLS 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 TOE acts as a receiver in the RSA key establishment scheme, the TOE 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.1.1/1. The domain parameters used for the finite field-based key establishment scheme are specified by the key generation according to FCS_CKM.1.1/1.	The evaluator shall 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 shall examine the TSS to verify that it identifies the usage for each scheme. The evaluator shall verify that the AGD guidance instructs the administrator how to configure the TOE to use the selected key establishment scheme(s). Evaluation Activity Note: The following tests require the developer to provide access to a test platform that provides the evaluator with tools that are typically not found on factory products. **Key Establishment Schemes** The evaluator shall verify the implementation of the key establishment schemes supported by the TOE using the applicable tests below. **SP800-56A Key Establishment Schemes** The evaluator shall verify a TOE's implementation of SP800-56A key agreement schemes using the following Function and Validity tests. These validation tests for each key agreement scheme verify that a TOE has implemented the components of the key agreement scheme according to the specifications in the Recommendation. These components include the calculation of the DLC primitives (the shared secret value Z) and the calculation of the derived keying material (DKM) via the Key Derivation Function (KDF). If key confirmation is supported, the evaluator shall also verify that the components of key confirmation have been implemented correctly, using the test procedures described below. This includes the parsing of the DKM, the generation of MACdata and the calculation of MACtag. **Function Test** The Function test verifies the ability of the TOE to implement the key agreement schemes correctly. To conduct this test the evaluator shall generate or obtain test vectors from a known good implementation of the TOE supported schemes. For each supported key agreement scheme-key agreement role combination, KDF type, and, if supported, key confirmation role- key confirmation type combination, the tester shall generate 10 sets of test vectors. The data set consists of one set of domain parameter values (FFC) or the NIST approved curve (ECC) per 10 sets of public keys. These keys are static, ephemeral or both depending on the scheme being tested. The evaluator shall obtain the DKM, the corresponding TOE’s public keys (static and/or ephemeral), the MAC tag(s), and any inputs used in the KDF, such as the Other Information *(OtherInfo)* and TOE id fields. If the TOE does not use a KDF defined in SP 800-56A, the evaluator shall obtain only the public keys and the hashed value of the shared secret. The evaluator shall verify the correctness of the TSF’s implementation of a given scheme by using a known good implementation to calculate the shared secret value, derive the keying material DKM, and compare hashes or MAC tags generated from these values. If key confirmation is supported, the TSF shall perform the above for each implemented approved MAC algorithm. **Validity Test** The Validity test verifies the ability of the TOE to recognize another party’s valid and invalid key agreement results with or without key confirmation. To conduct this test, the evaluator shall obtain a list of the supporting cryptographic functions included in the SP800-56A key agreement implementation to determine which errors the TOE should be able to recognize. The evaluator generates a set of 24 (FFC) or 30 (ECC) test vectors consisting of data sets including domain parameter values or NIST approved curves, the evaluator’s public keys, the TOE’s public/private key pairs, MACTag, and any inputs used in the KDF, such as the OtherInfo and TOE id fields. The evaluator shall inject an error in some of the test vectors to test that the TOE recognizes invalid key agreement results caused by the following fields being incorrect: the shared secret value Z, the DKM, the OtherInfo field, the data to be MACed, or the generated MACTag. If the TOE 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 TOE’s static private key to assure the TOE 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 TOE shall use these modified test vectors to emulate the key agreement scheme using the corresponding parameters. The evaluator shall compare the TOE’s results with the results using a known good implementation verifying that the TOE detects these errors. **SP800-56B Key Establishment Schemes** The evaluator shall verify that the TSS describes whether the TOE acts as a sender, a recipient, or both for RSA-based key establishment schemes. If the TOE acts as a sender, the following evaluation activity shall be performed to ensure the proper operation of every TOE supported combination of RSA-based key establishment scheme: To conduct this test the evaluator shall generate or obtain test vectors from a known good implementation of the TOE supported schemes. For each combination of supported key establishment scheme and its options (with or without key confirmation if supported, for each supported key confirmation MAC function if key confirmation is supported, and for each supported mask generation function if KTS-OAEP is supported), the tester shall generate 10 sets of test vectors. Each test vector shall include the RSA public key, the plaintext keying material, any additional input parameters if applicable, the MacKey and MacTag if key confirmation is incorporated, and the outputted ciphertext. For each test vector, the evaluator shall perform a key establishment encryption operation on the TOE with the same inputs (in cases where key confirmation is incorporated, the test shall use the MacKey from the test vector instead of the randomly generated MacKey used in normal operation) and ensure that the outputted ciphertext is equivalent to the ciphertext in the test vector.

ID	Requirement	Assurance Activity
		If the TOE acts as a receiver, the following evaluation activities shall be performed to ensure the proper operation of every TOE supported combination of RSA-based key establishment scheme: To conduct this test the evaluator shall generate or obtain test vectors from a known good implementation of the TOE supported schemes. For each combination of supported key establishment scheme and its options (with our without key confirmation if supported, for each supported key confirmation MAC function if key confirmation is supported, and for each supported mask generation function if 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 (KeyData), any additional input parameters if applicable, the MacTag in cases where key confirmation is incorporated, and the outputted ciphertext. For each test vector, the evaluator shall perform the key establishment decryption operation on the TOE and ensure that the outputted plaintext keying material (KeyData) is equivalent to the plaintext keying material in the test vector. In cases where key confirmation is incorporated, the evaluator shall perform the key confirmation steps and ensure that the outputted MacTag is equivalent to the MacTag in the test vector. The evaluator shall ensure that the TSS describes how the TOE handles decryption errors. In accordance with NIST Special Publication 800-56B, the TOE must not reveal the particular error that occurred, either through the contents of any outputted or logged error message or through timing variations. If KTS-OAEP is supported, the evaluator shall create separate contrived ciphertext values that trigger each of the three decryption error checks described in NIST Special Publication 800-56B section 7.2.2.3, ensure that each decryption attempt results in an error, and ensure that any outputted or logged error message is identical for each. If KTS-KEM-KWS is supported, the evaluator shall create separate contrived ciphertext values that trigger each of the three decryption error checks described in NIST Special Publication 800-56B section 7.2.3.3, ensure that each decryption attempt results in an error, and ensure that any outputted or logged error message is identical for each. **RSA-based key establishment** The evaluator shall verify the correctness of the TSF's implementation of RSAES-PKCS1-v1_5 by using a known good implementation for each protocol selected in FTP_DIT_EXT.1 that uses RSAES-PKCS1-v1_5. **Diffie-Hellman Group 14** The evaluator shall verify the correctness of the TSF's implementation of Diffie-Hellman group 14 by using a known good implementation for each protocol selected in FTP_DIT_EXT.1 that uses Diffie-Hellman group 14. **FFC Schemes using "safe-prime" groups** The evaluator shall verify the correctness of the TSF's implementation of safe-prime groups by using a known good implementation for each protocol selected in FTP_DIT_EXT.1 that uses safe-prime groups. This test must be performed for each safe-prime group that each protocol uses.
	The **application** shall perform *encryption/decryption* in accordance with a specified cryptographic algorithm [**selection**: • *AES-CBC (as defined in NIST SP 800-38A) mode*, • *AES-GCM (as defined in NIST SP 800-38D) mode*, • *AES-XTS (as defined in NIST SP 800-38E) mode*] and cryptographic key sizes [**selection**: *128-bit*, *256-bit*] . ***This is a selection-based requirement. Its inclusion depends upon selection in .*** **Application Note:** This is dependent on implementing cryptographic functionality, as in FTP_DIT_EXT.1. For the first selection, the ST author should choose the mode or modes in which AES operates. For the second 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 certain TLS implementations.	The evaluator checks the AGD documents to determine that any configuration that is required to be done to configure the functionality for the required modes and key sizes is present. The evaluator shall perform all of the following tests for each algorithm implemented by the TSF 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 shall 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 shall 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 shall 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 shall 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 shall 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 shall 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 shall 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 shall 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].

<table>
<tr><th>ID</th><th>Requirement</th><th>Assurance Activity</th></tr>
<tr><td></td><td></td><td>
To test the decrypt functionality of AES-CBC, the evaluator shall 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 shall test the encrypt functionality by encrypting an i-block message where 1 < i <= 10. The evaluator shall 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 shall also test the decrypt functionality for each mode by decrypting an i-block message where 1 < i <=10. The evaluator shall 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 shall 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:
<pre>
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]
</pre>
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 shall 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 shall 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 shall 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 shall 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 shall compare the resulting values to those obtained by submitting the same inputs to a known good implementation.
AES-XTS Tests

The evaluator shall 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 non-zero 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 shall test the decrypt functionality of XTS-AES using the same test as for encrypt, replacing plaintext values with ciphertext values and XTS-AES encrypt with XTS-AES decrypt.
</td></tr>
<tr><td></td><td>
The application shall perform cryptographic hashing services in accordance with a specified cryptographic algorithm [selection:

SHA-1,
SHA-256,
SHA-384,
SHA-512,
no other

] and message digest sizes [selection:

160,
256,
384,

</td><td>
The evaluator shall 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 hashes only 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 shall 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.
</td></tr>
</table>

ID	Requirement	Assurance Activity
	• *512* • *no other*] bits that meet the following: FIPS Pub 180-4. ***This is a selection-based requirement. Its inclusion depends upon selection in .*** **Application Note:** This is dependent on implementing cryptographic functionality, as in FTP_DIT_EXT.1. 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 included in order to comply with the TLS. If the TLS package is included in the ST, the hashing algorithms selection for FCS_COP.1(2) must match the hashing algorithms used in the mandatory and selected cipher suites of the TLS package. 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 (for example, SHA 256 for 128-bit keys).	• **Test 1:** Short Messages Test - Bit oriented Mode The evaluators devise 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 evaluators compute the message digest for each of the messages and ensure that the correct result is produced when the messages are provided to the TSF. • **Test 2:** Short Messages Test - Byte oriented Mode The evaluators devise 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 evaluators compute the message digest for each of the messages and ensure that the correct result is produced when the messages are provided to the TSF. • **Test 3:** Selected Long Messages Test - Bit oriented Mode The evaluators devise an input set consisting of m messages, where m is the block length of the hash algorithm. The length of the ith message is 512 + 99*i, where 1 ≤ i ≤ m. The message text shall be pseudorandomly generated. The evaluators compute the message digest for each of the messages and ensure that the correct result is produced when the messages are provided to the TSF. • **Test 4:** Selected Long Messages Test - Byte oriented Mode The evaluators devise an input set consisting of m/8 messages, where m is the block length of the hash algorithm. The length of the ith message is 512 + 8*99*i, where 1 ≤ i ≤ m/8. The message text shall be pseudorandomly generated. The evaluators compute the message digest for each of the messages and ensure that the correct result is produced when the messages are provided to the TSF. • **Test 5:** Pseudorandomly Generated Messages Test This test is for byte-oriented implementations only. The evaluators randomly generate a seed that is n bits long, where n is the length of the message digest produced by the hash function to be tested. The evaluators then formulate a set of 100 messages and associated digests by following the algorithm provided in Figure 1 of [SHAVS]. The evaluators then ensure that the correct result is produced when the messages are provided to the TSF.
	The **application** 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*] . ***This is a selection-based requirement. Its inclusion depends upon selection in .*** **Application Note:** This is dependent on implementing cryptographic functionality, as in FTP_DIT_EXT.1. 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.	The evaluator shall 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 shall 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 shall 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 shall 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 shall obtain in response a set of 10 PASS/FAIL values. RSA Signature Algorithm Tests • **Test 1:** Signature Generation Test. The evaluator shall verify the implementation of RSA Signature Generation by the TOE 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 shall have the TOE use their private key and modulus value to sign these messages. The evaluator shall 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 shall perform the Signature Verification test to verify the ability of the TOE to recognize another party’s valid and invalid signatures. The evaluator shall 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 TOE attempts to verify the signatures and returns success or failure.
	The **application** shall perform *keyed-hash message authentication* in accordance with a specified cryptographic algorithm • HMAC-SHA-256 and [**selection**: • *SHA-1*, • *SHA-384*, • *SHA-512*, • *no other algorithms*] with key sizes [**assignment**: *key size (in bits) used in HMAC*] and message digest sizes 256 and [**selection**: *160*, *384*, *512*, *no other size*] bits that meet the following: FIPS Pub 198-1 *The Keyed-Hash Message Authentication Code* and FIPS Pub 180-4 *Secure Hash Standard*. ***This is a selection-based requirement. Its inclusion depends upon selection in .*** **Application Note:** This is dependent on implementing cryptographic functionality, as in FTP_DIT_EXT.1. 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 application (e.g., trusted channel). The hash selection must support the message digest size selection. The hash selection should be consistent with the	The evaluator shall perform the following activities based on the selections in the ST. For each of the supported parameter sets, the evaluator shall compose 15 sets of test data. Each set shall consist of a key and message data. The evaluator shall have the TSF generate HMAC tags for these sets of test data. The resulting MAC tags shall be compared to the result of generating HMAC tags with the same key and IV using a known-good implementation.

ID	Requirement	Assurance Activity
	overall strength of the algorithm used for FCS_COP.1/1.	
	The application shall [**selection**: • *not store any credentials*, • *invoke the functionality provided by the platform to securely store [**assignment**: list of credentials]* , • *implement functionality to securely store [**assignment**: list of credentials] according to [**selection**: FCS_COP.1(1), FCS_CKM.1(3)]*] to non-volatile memory. **Application Note:** This requirement ensures that persistent credentials (secret keys, PKI private keys, passwords, etc) are stored securely, and never persisted in cleartext form. Application developers are encouraged to use platform mechanisms for the secure storage of credentials. Depending on the platform that may include hardware-backed protection for credential storage. Application developers must choose a selection, or multiple selections, based on all credentials that the application stores. If *not store any credentials* is selected then the application must not store any credentials. If *invoke the functionality provided by the platform to securely store* is selected then the Application developer must closely review the EA for their platform and provide documentation indicating which platform mechanisms are used to store credentials. If *implement functionality to securely store credentials* is selected, then the following components must be included in the ST: FCS_COP.1/1 or FCS_CKM.1/3. If other cryptographic operations are used to implement the secure storage of credentials, the corresponding requirements must be included in the ST. If the OS is Linux and Java KeyStores are used to store credentials, *implement functionality to securely store credentials* must be selected.	The evaluator shall check the TSS to ensure that it lists all persistent credentials (secret keys, PKI private keys, or passwords) needed to meet the requirements in the ST. For each of these items, the evaluator shall confirm that the TSS lists for what purpose it is used, and how it is stored. For all credentials for which the application implements functionality, the evaluator shall verify credentials are encrypted according to FCS_COP.1/1 or conditioned according to FCS_CKM.1.1/1 and FCS_CKM.1/3. For all credentials for which the application invokes platform-provided functionality, the evaluator shall perform the following actions which vary per platform. ***For Android:*** The evaluator shall verify that the application uses the Android KeyStore or the Android KeyChain to store certificates. ***For Windows:*** The evaluator shall verify that all certificates are stored in the Windows Certificate Store. The evaluator shall verify that other credentials, like passwords, are stored in the Windows Credential Manager or stored using the Data Protection API (DPAPI). For Windows Universal Applications, the evaluator shall verify that the application is using the ProtectData class and storing credentials in IsolatedStorage. ***For iOS:*** The evaluator shall verify that all credentials are stored within a Keychain. ***For Linux:*** The evaluator shall verify that all keys are stored using Linux keyrings. ***For Solaris:*** The evaluator shall verify that all keys are stored using Solaris Key Management Framework (KMF). ***For macOS:*** The evaluator shall verify that all credentials are stored within Keychain.
	The application shall implement the HTTPS protocol that complies with RFC 2818. ***This is a selection-based requirement. Its inclusion depends upon selection in .***	The evaluator shall attempt to establish an HTTPS connection with a webserver, observe the traffic with a packet analyzer, and verify that the connection succeeds and that the traffic is identified as TLS or HTTPS.
	The application shall implement HTTPS using TLS as defined in the TLS package. ***This is a selection-based requirement. Its inclusion depends upon selection in .***	Other tests are performed in conjunction with the TLS package.
	The application shall [**selection**: *not establish the application-initiated connection* , *notify the user and not establish the user-initiated connection* , *notify the user and request authorization to establish the user-initiated connection*] if the peer certificate is deemed invalid. ***This is a selection-based requirement. Its inclusion depends upon selection in .*** **Application Note:** Validity is determined by the certificate path, the expiration 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, and the evaluator shall perform the following test: • **Test 1:** The evaluator shall demonstrate that using a certificate without a valid certification path results in the selected action in the SFR. If "notify the user" is selected in the SFR, then the evaluator shall also determine that the user is notified of the certificate validation failure. Using the administrative guidance, the evaluator shall then load a certificate or certificates to the Trust Anchor Database needed to validate the certificate to be used in the function, and demonstrate that the function succeeds. The evaluator then shall delete one of the certificates, and show that again, using a certificate without a valid certification path results in the selected action in the SFR, and if "notify the user" was selected in the SFR, the user is notified of the validation failure.
	The application shall restrict its access to [**selection**: • *no hardware resources*, • *network connectivity*, • *camera*, • *microphone*, • *location services*, • *NFC*, • *USB*, • *Bluetooth*, • *[**assignment**: list of additional hardware resources]*]. **Application Note:** The intent is for the evaluator to ensure that the selection captures all hardware resources which the application accesses, and that these are restricted to those which are justified. On some platforms, the application must explicitly solicit permission in order to access hardware resources. Seeking such permissions, even if the application does not later make use of the hardware resource, should still be considered access. Selections should be expressed in a manner consistent with how the application expresses its access needs to the underlying platform. For example, the platform may provide *location services* which implies the potential use of a variety of hardware resources (e.g. satellite receivers, WiFi, cellular radio) yet *location services* is the proper selection. This is because use of these resources can be inferred, but also because the actual usage may vary based on the particular platform. Resources that do not need to be explicitly identified are those which are ordinarily used by any application such as central processing units, main memory, displays, input devices (e.g. keyboards, mice), and persistent storage devices provided by the platform.	The evaluator shall perform the platform-specific actions below and inspect user documentation to determine the application's access to hardware resources. The evaluator shall ensure that this is consistent with the selections indicated. The evaluator shall review documentation provided by the application developer and for each resource which it accesses, identify the justification as to why access is required. ***For Android:*** The evaluator shall inspect permissions presented at installation time (Android 5.1 and below) or on-access (Android 6.0 and above) for each hardware resource an app intends to access. ***For Windows:*** For Windows Universal Applications the evaluator shall check the WMAppManifest.xml file for a list of required hardware capabilities. The evaluator shall verify that the user is made aware of the required hardware capabilities when the application is first installed. This includes permissions such as ID_CAP_ISV_CAMERA, ID_CAP_LOCATION, ID_CAP_NETWORKING, ID_CAP_MICROPHONE, ID_CAP_PROXIMITY and so on. A complete list of Windows App permissions can be found at: • http://msdn.microsoft.com/en-US/library/windows/apps/jj206936.aspx For Windows Desktop Applications the evaluator shall identify in either the application software or its documentation the list of the required hardware resources. ***For iOS:*** The evaluator shall verify that either the application or the documentation provides a list of the hardware resources it accesses. ***For Linux:*** The evaluator shall verify that either the application software or its documentation provides a list of the hardware resources it accesses. ***For Solaris:*** The evaluator shall verify that either the application software or its documentation provides a list of the hardware resources it accesses. ***For macOS:*** The evaluator shall verify that either the application software or its documentation provides a list of the hardware resources it accesses.
	The application shall restrict its access to [**selection**: • *no sensitive information repositories*, • *address book*, • *calendar*, • *call lists*, • *system logs*, • *[**assignment**: list of additional sensitive information repositories]*] . **Application Note:** *Sensitive information repositories* are defined as those collections of sensitive data that could be expected to be shared among some applications, users, or user roles, but to which not all of these would ordinarily require access.	The evaluator shall perform the platform-specific actions below and inspect user documentation to determine the application's access to sensitive information repositories. The evaluator shall ensure that this is consistent with the selections indicated. The evaluator shall review documentation provided by the application developer and for each sensitive information repository which it accesses, identify the justification as to why access is required. ***For Android:*** The evaluator shall inspect permissions presented at installation time (Android 5.1 and below) or on-access (Android 6.0 and above) for each sensitive information repository an app intends to access. ***For Windows:*** For Windows Universal Applications the evaluator shall check the WMAppManifest.xml file for a list of required capabilities. The evaluator shall identify the required information repositories when the application is first installed. This includes permissions such as ID_CAP_CONTACTS,ID_CAP_APPOINTMENTS,ID_CAP_MEDIALIB and so on. A complete list of Windows App permissions can be found at: • http://msdn.microsoft.com/en-US/library/windows/apps/jj206936.aspx For Windows Desktop Applications the evaluator shall identify in either the application software or its documentation the list of sensitive information repositories it accesses.

ID	Requirement	Assurance Activity
		For iOS: The evaluator shall verify that either the application software or its documentation provides a list of the sensitive information repositories it accesses. ***For Linux:*** The evaluator shall verify that either the application software or its documentation provides a list of sensitive information repositories it accesses. ***For Solaris:*** The evaluator shall verify that either the application software or its documentation provides a list of sensitive information repositories it accesses. ***For macOS:*** The evaluator shall verify that either the application software or its documentation provides a list of sensitive information repositories it accesses.
	The application shall restrict network communication to [**selection**: • *no network communication*, • *user-initiated communication for [**assignment**: list of functions for which the user can initiate network communication]*, • *respond to [**assignment**: list of remotely initiated communication]*, • *[**assignment**: list of application-initiated network communication]*] . **Application Note:** This requirement is intended to restrict both inbound and outbound network communications to only those required, or to network communications that are user initiated. It does not apply to network communications in which the application may generically access the filesystem which may result in the platform accessing remotely mounted drives/shares.	The evaluator shall perform the following tests: • **Test 1:** The evaluator shall run the application. While the application is running, the evaluator shall sniff network traffic ignoring all non-application associated traffic and verify that any network communications witnessed are documented in the TSS or are user-initiated. • **Test 2:** The evaluator shall run the application. After the application initializes, the evaluator shall run network port scans to verify that any ports opened by the application have been captured in the ST for the third selection and its assignment. This includes connection-based protocols (e.g. TCP, DCCP) as well as connectionless protocols (e.g. UDP). ***For Android:*** If "no network communication" is selected, the evaluator shall ensure that the application's AndroidManifest.xml file does not contain a uses-permission or uses-permission-sdk-23 tag containing android:name="android.permission.INTERNET". In this case, it is not necessary to perform the above Tests 1 and 2, as the platform will not allow the application to perform any network communication.
	The application shall [**selection**: • *leverage platform-provided functionality to encrypt sensitive data*, • *implement functionality to encrypt sensitive data as defined in the PP-Module for File Encryption*, • *protect sensitive data in accordance with FCS_STO_EXT.1*, • *not store any sensitive data*] in non-volatile memory. **Application Note:** If *implement functionality to encrypt sensitive data as defined in the PP-Module for File Encryption* is selected, the TSF must claim conformance to a PP-Configuration that includes the File Encryption PP-Module. Any file that may potentially contain sensitive data (to include temporary files) shall be protected. The only exception is if the user intentionally exports the sensitive data to non-protected files. ST authors should select *protect sensitive data in accordance with FCS_STO_EXT.1* for the sensitive data that is covered by the FCS_STO_EXT.1 SFR.	The evaluator shall examine the TSS to ensure that it describes the sensitive data processed by the application. The evaluator shall then ensure that the following activities cover all of the sensitive data identified in the TSS. If **not store any sensitive data** is selected, the evaluator shall inspect the TSS to ensure that it describes how sensitive data cannot be written to non-volatile memory. The evaluator shall also ensure that this is consistent with the filesystem test below. Evaluation activities (after the identification of the sensitive data) are to be performed on all sensitive data listed that are not covered by FCS_STO_EXT.1. The evaluator shall inventory the filesystem locations where the application may write data. The evaluator shall run the application and attempt to store sensitive data. The evaluator shall then inspect those areas of the filesystem to note where data was stored (if any), and determine whether it has been encrypted. If **leverage platform-provided functionality** is selected, the evaluation activities will be performed as stated in the following requirements, which vary on a per-platform basis. ***For Android:*** The evaluator shall inspect the TSS and verify that it describes how files containing sensitive data are stored with the MODE_PRIVATE flag set. ***For Windows:*** The Windows platform currently does not provide data-at-rest encryption services which depend upon invocation by application developers. The evaluator shall verify that the Operational User Guidance makes the need to activate platform encryption, such as BitLocker or Encrypting File System (EFS), clear to the end user. ***For iOS:*** The evaluator shall inspect the TSS and ensure that it describes how the application uses the Complete Protection, Protected Unless Open, or Protected Until First User Authentication Data Protection Class for each data file stored locally. ***For Linux:*** The Linux platform currently does not provide data-at-rest encryption services which depend upon invocation by application developers. The evaluator shall verify that the Operational User Guidance makes the need to activate platform encryption clear to the end user. ***For Solaris:*** The Solaris platform currently does not provide data-at-rest encryption services which depend upon invocation by application developers. The evaluator shall verify that the Operational User Guidance makes the need to activate platform encryption clear to the end user. ***For macOS:*** The macOS platform currently does not provide data-at-rest encryption services which depend upon invocation by application developers. The evaluator shall verify that the Operational User Guidance makes the need to activate platform encryption clear to the end user.

	The application shall [**selection**: *invoked platform-provided functionality* , *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 application 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 application 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 5280 Section 6.3* , *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 application 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.
	  ◦ Server certificates presented for EST shall have the CMC
		The evaluator shall 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. If the application supports chains of length four or greater, the evaluator shall create a chain of at least four certificates: the node certificate to be tested, two Intermediate CAs, and the self-signed Root CA. If the application supports a maximum trust depth of two, then a chain with no Intermediate CA should instead be created.
		
		• **Test 1:** The evaluator shall demonstrate that validating a certificate without a valid certification path results in the function failing. The evaluator shall 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 shall demonstrate that validating an expired certificate results in the function failing.
		• **Test 3:** The evaluator shall test that the TOE can properly handle revoked certificates-–conditional on whether CRL, OCSP, or OCSP Stapling is selected; if multiple methods are selected, then the following tests shall be performed for each method:
		  ◦ The evaluator shall test revocation of the node certificate.
		  ◦ The evaluator shall also test revocation of an intermediate CA certificate (i.e. the intermediate CA certificate should be revoked by the root CA), if intermediate CA certificates are supported.
		The evaluator shall 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.

ID	Requirement	Assurance Activity
	Registration Authority (RA) purpose (id-kp-cmcRA with OID 1.3.6.1.5.5.7.3.28) in the extendedKeyUsage field. ***This is a selection-based requirement. Its inclusion depends upon selection in .*** **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. Regardless of the selection of *implement functionality* or *invoke platform-provided functionality*, the validation is expected to end in a trusted root CA certificate in a root store managed by the platform.	• **Test 4:** If OCSP is selected, the evaluator shall 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 shall 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 shall modify any byte in the first eight bytes of the certificate and demonstrate that the certificate fails to validate. (The certificate will fail to parse correctly.) • **Test 6:** The evaluator shall modify any byte in the last byte of the certificate and demonstrate that the certificate fails to validate. (The signature on the certificate will not validate.) • **Test 7:** The evaluator shall modify any byte in the public key of the certificate and demonstrate that the certificate fails to validate. (The signature on the certificate will not validate.)
	The application shall treat a certificate as a CA certificate only if the basicConstraints extension is present and the CA flag is set to TRUE. ***This is a selection-based requirement. Its inclusion depends upon selection in .*** **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. If the application supports chains of length four or greater, the evaluator shall create a chain of at least four certificates: the node certificate to be tested, two Intermediate CAs, and the self-signed Root CA. If the application supports a maximum trust depth of two, then a chain with no Intermediate CA should instead be created. • **Test 1:** The evaluator shall ensure that the certificate of at least one of the CAs in the chain does not contain the basicConstraints extension. The evaluator shall confirm that validation of the certificate path fails (i) as part of the validation of the peer certificate belonging to this chain; and/or (ii) when attempting to add the CA certificate without the basicConstraints extension to the TOE's trust store. • **Test 2:** The evaluator shall ensure that the certificate of at least one of the CAs in the chain has the CA flag in the basicConstraints extension not set (or set to FALSE). The evaluator shall confirm that validation of the certificate path fails (i) as part of the validation of the peer certificate belonging to this chain; and/or (ii) when attempting to add the CA certificate with the CA flag not set (or set to FALSE) in the basicConstraints extension to the TOE's trust store.
	The application shall use X.509v3 certificates as defined by RFC 5280 to support authentication for [**selection**: *HTTPS*, *TLS*, *DTLS*, *SSH*, *IPsec*]. ***This is a selection-based requirement. Its inclusion depends upon selection in .*** **Application Note:** The ST author's selection shall match the selection in FTP_DIT_EXT.1.1.	
	When the application cannot establish a connection to determine the validity of a certificate, the application shall [**selection**: *allow the administrator to choose whether to accept the certificate in these cases* , *accept the certificate* , *not accept the certificate*]. ***This is a selection-based requirement. Its inclusion depends upon selection in .*** **Application Note:** Often a connection must be established to perform a verification of the revocation status of a certificate - either to download a CRL or to perform OCSP. The selection is used to describe the behavior in the event that such a connection cannot be established (for example, due to a network error). If the TOE has determined the certificate valid according to all other rules in FIA_X509_EXT.1, the behavior indicated in the selection shall determine the validity. The TOE must not accept the certificate if it fails any of the other validation rules in FIA_X509_EXT.1.	The evaluator shall check the TSS to ensure that it describes how the TOE chooses which certificates to use, and any necessary instructions in the administrative guidance for configuring the operating environment so that the TOE can use the certificates. The evaluator shall examine the TSS to confirm that it describes the behavior of the TOE when a connection cannot be established during the validity check of a certificate used in establishing a trusted channel. The evaluator shall verify that any distinctions between trusted channels are described. If the requirement that the administrator is able to specify the default action, then the evaluator shall ensure that the operational guidance contains instructions on how this configuration action is performed. The evaluator shall perform the following test for each trusted channel: • **Test 1:** The evaluator shall demonstrate that using a valid certificate that requires certificate validation checking to be performed in at least some part by communicating with a non-TOE IT entity. The evaluator shall then manipulate the environment so that the TOE is unable to verify the validity of the certificate, and observe that the action selected in FIA_X509_EXT.2.2 is performed. If the selected action is administrator-configurable, then the evaluator shall follow the operational guidance to determine that all supported administrator-configurable options behave in their documented manner. • **Test 2:** The evaluator shall demonstrate that an invalid certificate that requires certificate validation checking to be performed in at least some part by communicating with a non-TOE IT entity cannot be accepted.
	The application shall [**selection**: *invoke the mechanisms recommended by the platform vendor for storing and setting configuration options*, *implement functionality to encrypt and store configuration options as defined by FDP_PRT_EXT.1 in the PP-Module for File Encryption*] **Application Note:** Configuration options that are stored remotely are not subject to this requirement. is generally not considered part of configuration options and should be stored according to fdp_dar_ext.1 or fcs_sto_ext.1.	The evaluator shall review the TSS to identify the application's configuration options (e.g. settings) and determine whether these are stored and set using the mechanisms supported by the platform or implemented by the application in accordance with the PP-Module for File Encryption. At a minimum the TSS shall list settings related to any SFRs and any settings that are mandated in the operational guidance in response to an SFR. Conditional: If "implement functionality to encrypt and store configuration options as defined by FDP_PRT_EXT.1 in the PP-Module for File Encryption" is selected, the evaluator shall ensure that the TSS identifies those options, as well as indicates where the encrypted representation of these options is stored. If "invoke the mechanisms recommended by the platform vendor for storing and setting configuration options" is chosen, the method of testing varies per platform as follows: ***For Android:*** The evaluator shall run the application and make security-related changes to its configuration. The evaluator shall check that at least one XML file at location /data/data/package/shared_prefs/ reflects the changes made to the configuration to verify that the application used SharedPreferences and/or PreferenceActivity classes for storing configuration data, where *package* is the Java package of the application. ***For Windows:*** The evaluator shall determine and verify that Windows Universal Applications use either the Windows.UI.ApplicationSettings namespace or the IsolatedStorageSettings namespace for storing application specific settings. For .NET applications, the evaluator shall determine and verify that the application uses one of the locations listed in https://docs.microsoft.com/en-us/dotnet/framework/configure-apps/ for storing application specific settings. For Classic Desktop applications, the evaluator shall run the application while monitoring it with the SysInternals tool ProcMon and make changes to its configuration. The evaluator shall verify that ProcMon logs show corresponding changes to the the Windows Registry or C:\ProgramData\ directory. ***For iOS:*** The evaluator shall verify that the app uses the user defaults system or

ID	Requirement	Assurance Activity
		key-value store for storing all settings. ***For Linux:*** The evaluator shall run the application while monitoring it with the utility strace. The evaluator shall make security-related changes to its configuration. The evaluator shall verify that strace logs corresponding changes to configuration files that reside in /etc (for system-specific configuration), in the user's home directory (for user-specific configuration), or /var/lib/ (for configurations controlled by UI and not intended to be directly modified by an administrator). ***For Solaris:*** The evaluator shall run the application while monitoring it with the utility dtrace. The evaluator shall make security-related changes to its configuration. The evaluator shall verify that dtrace logs corresponding changes to configuration files that reside in /etc (for system-specific configuration) or in the user's home directory(for user-specific configuration). ***For macOS:*** The evaluator shall verify that the application stores and retrieves settings using the NSUserDefaults class. If "implement functionality to encrypt and store configuration options as defined by FDP_PRT_EXT.1 in the PP-Module for File Encryption" is selected, for all configuration options listed in the TSS as being stored and protected using encryption, the evaluator shall examine the contents of the configuration option storage (identified in the TSS) to determine that the options have been encrypted.
	The application shall provide only enough functionality to set new credentials when configured with default credentials or no credentials. **Application Note:** Default credentials are credentials (e.g., passwords, keys) that are automatically (without user interaction) loaded onto the platform during application installation. Credentials that are generated during installation using requirements laid out in FCS_RBG_EXT.1 are not by definition default credentials.	The evaluator shall check the TSS to determine if the application requires any type of credentials and if the application installs with default credentials. If the application uses any default credentials the evaluator shall run the following tests. • **Test 1:** The evaluator shall install and run the application without generating or loading new credentials and verify that only the minimal application functionality required to set new credentials is available. • **Test 2:** The evaluator shall attempt to clear all credentials and verify that only the minimal application functionality required to set new credentials is available. • **Test 3:** The evaluator shall run the application, establish new credentials and verify that the original default credentials no longer provide access to the application.
	The application shall be configured by default with file permissions which protect the application binaries and data files from modification by normal unprivileged users. **Application Note:** The precise expectations for file permissions vary per platform but the general intention is that a trust boundary protects the application and its data.	The evaluator shall install and run the application. The evaluator shall inspect the filesystem of the platform (to the extent possible) for any files created by the application and ensure that their permissions are adequate to protect them. The method of doing so varies per platform. ***For Android:*** The evaluator shall run ls -alR|grep -E '^........w.' inside the application's data directories to ensure that all files are not world-writable. The command should not print any files. The evaluator shall also verify that no sensitive data is written to external storage which could be read/modified by any application containing the READ_EXTERNAL_STORAGE and/or WRITE_EXTERNAL_STORAGE permissions. ***For Windows:*** The evaluator shall run the SysInternals tools, Process Monitor and Access Check (or tools of equivalent capability, like icacls.exe) for Classic Desktop applications to verify that files written to disk during an application's installation have the correct file permissions, such that a standard user cannot modify the application or its data files. For Windows Universal Applications the evaluator shall consider the requirement met because of the AppContainer sandbox. ***For iOS:*** The evaluator shall determine whether the application leverages the appropriate Data Protection Class for each data file stored locally. ***For Linux:*** The evaluator shall run the command find . -perm /002 inside the application's data directories to ensure that all files are not world-writable. The command should not print any files. ***For Solaris:*** The evaluator shall run the command find . \(-perm -002 \) inside the application's data directories to ensure that all files are not world-writable. The command should not print any files. ***For macOS:*** The evaluator shall run the command find . -perm +002 inside the application's data directories to ensure that all files are not world-writable. The command should not print any files.
	The TSF shall be capable of performing the following management functions [**selection**: • *no management functions*, • *enable/disable the transmission of any information describing the system's hardware, software, or configuration* , • *enable/disable the transmission of any PII* , • *enable/disable transmission of any application state (e.g. crashdump) information*, • *enable/disable network backup functionality to [**assignment**: list of enterprise or commercial cloud backup systems]* , • *[**assignment**: list of other management functions to be provided by the TSF]*] . **Application Note:** This requirement stipulates that an application needs to provide the ability to enable/disable only those functions that it actually implements. The application is not responsible for controlling the behavior of the platform or other applications.	The evaluator shall verify that every management function mandated by the PP 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 shall test the application's ability to provide the management functions by configuring the application 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.
	The application shall [**selection**: • *not transmit PII over a network* , • *require user approval before executing [**assignment**: list of functions that transmit PII over a network]*] . **Application Note:** This requirement applies only to PII that is specifically requested by the application; it does not apply if the user volunteers PII without prompting from the application into a general (or inappropriate) data field. A dialog box that declares intent to send PII presented to the user at the time the application is started is sufficient to meet this requirement.	The evaluator shall inspect the TSS documentation to identify functionality in the application where PII can be transmitted. If require user approval before executing is selected, the evaluator shall run the application and exercise the functionality responsibly for transmitting PII and verify that user approval is required before transmission of the PII.
	The application shall use only documented platform APIs. **Application Note:** The definition of *documented* may vary depending upon whether the application is provided by a third party (who relies upon documented platform APIs) or by a platform vendor who may be able to guarantee support for platform APIs.	The evaluator shall verify that the TSS lists the platform APIs used in the application. The evaluator shall then compare the list with the supported APIs (available through e.g. developer accounts, platform developer groups) and ensure that all APIs listed in the TSS are supported.

ID	Requirement	Assurance Activity
	The application [**selection**: *shall use platform-provided libraries, does not implement functionality*] for parsing [**assignment**: *list of formats parsed that are included in the IANA MIME media types*]. ***This is currently an objective requirement.*** **Application Note:** The IANA MIME types are listed at http://www.iana.org/assignments/media-types and include many image, audio, video, and content file formats. This requirement does not apply if providing parsing services is the purpose of the application.	The evaluator shall verify that the TSS lists the IANA MIME media types (as described by http://www.iana.org/assignments/media-types) for all formats the application processes and that it maps those formats to parsing services provided by the platform.
	The application shall not request to map memory at an explicit address except for [**assignment**: *list of explicit exceptions*]. **Application Note:** Requesting a memory mapping at an explicit address subverts address space layout randomization (ASLR).	The evaluator shall ensure that the TSS describes the compiler flags used to enable ASLR when the application is compiled. The evaluator shall perform either a static or dynamic analysis to determine that no memory mappings are placed at an explicit and consistent address. The method of doing so varies per platform. ***For Android:*** The evaluator shall run the same application on two different Android systems. Both devices do not need to be evaluated, as the second device is acting only as a tool. Connect via ADB and inspect /proc/PID/maps. Ensure the two different instances share no memory mappings made by the application at the same location. Alternatively, the evaluator may use the same device. After collecting the first instance of mappings, the evaluator must uninstall the application, reboot the device, and reinstall the application to collect the second instance of mappings. ***For Windows:*** The evaluator shall run the same application on two different Windows systems and run a tool that will list all memory mapped addresses for the application. The evaluator shall then verify the two different instances share no mapping locations. The Microsoft SysInternals tool, VMMap, could be used to view memory addresses of a running application. The evaluator shall use a tool such as Microsoft's BinScope Binary Analyzer to confirm that the application has ASLR enabled. ***For iOS:*** The evaluator shall perform a static analysis to search for any mmap calls (or API calls that call mmap), and ensure that no arguments are provided that request a mapping at a fixed address. ***For Linux:*** The evaluator shall run the same application on two different Linux systems. The evaluator shall then compare their memory maps using pmap -x *PID* to ensure the two different instances share no mapping locations. ***For Solaris:*** The evaluator shall run the same application on two different Solaris systems. The evaluator shall then compare their memory maps using pmap -x *PID* to ensure the two different instances share no mapping locations. ***For macOS:*** The evaluator shall run the same application on two different Mac systems. The evaluator shall then compare their memory maps using vmmap *PID* to ensure the two different instances share no mapping locations.
	The application shall [**selection**: • *not allocate any memory region with both write and execute permissions* , • *allocate memory regions with write and execute permissions for only [**assignment**: list of functions performing just-in-time compilation]*] . **Application Note:** Requesting a memory mapping with both write and execute permissions subverts the platform protection provided by DEP. If the application performs no just-in-time compiling, then the first selection must be chosen.	The evaluator shall verify that no memory mapping requests are made with write and execute permissions. The method of doing so varies per platform. ***For Android:*** The evaluator shall perform static analysis on the application to verify that • mmap is never invoked with both the PROT_WRITE and PROT_EXEC permissions, and • mprotect is never invoked. ***For Windows:*** The evaluator shall use a tool such as Microsoft's BinScope Binary Analyzer to confirm that the application passes the NXCheck. The evaluator may also ensure that the /NXCOMPAT flag was used during compilation to verify that DEP protections are enabled for the application. ***For iOS:*** The evaluator shall perform static analysis on the application to verify that mprotect is never invoked with the PROT_EXEC permission. ***For Linux:*** The evaluator shall perform static analysis on the application to verify that both • mmap is never be invoked with both the PROT_WRITE and PROT_EXEC permissions, and • mprotect is never invoked with the PROT_EXEC permission. ***For Solaris:*** The evaluator shall perform static analysis on the application to verify that both • mmap is never be invoked with both the PROT_WRITE and PROT_EXEC permissions, and • mprotect is never invoked with the PROT_EXEC permission. ***For macOS:*** The evaluator shall perform static analysis on the application to verify that mprotect is never invoked with the PROT_EXEC permission.
	The application shall be compatible with security features provided by the platform vendor. **Application Note:** This requirement is designed to ensure that platform security features do not need to be disabled in order for the application to run.	The evaluator shall configure the platform in the ascribed manner and carry out one of the prescribed tests: ***For Android:*** Applications running on Android cannot disable Android security features, therefore this requirement is met and no evaluation activity is required. ***For Windows:*** If the OS platform supports Windows Defender Exploit Guard (Windows 10 version 1709 or later), then the evaluator shall ensure that the application can run successfully with Windows Defender Exploit Guard Exploit Protection configured with the following minimum mitigations enabled; Control Flow Guard (CFG), Randomize memory allocations (Bottom-Up ASLR), Export address filtering (EAF), Import address filtering (IAF), and Data Execution Prevention (DEP). The following link describes how to enable Exploit Protection, https://docs.microsoft.com/en-us/windows/security/threat-protection/windows-defender-exploit-guard/customize-exploit-protection. If the OS platform supports the Enhanced Mitigation Experience Toolkit (EMET) which can be installed on Windows 10 version 1703 and earlier, then the evaluator shall ensure that the application can run successfully with EMET configured with the following minimum mitigations enabled; Memory Protection Check, Randomize memory allocations (Bottom-Up ASLR), Export address filtering (EAF), and Data Execution Prevention (DEP). ***For iOS:*** Applications running on iOS cannot disable security features, therefore this requirement is met and no evaluation activity is required. ***For Linux:*** The evaluator shall ensure that the application can successfully run on a system with either SELinux or AppArmor enabled and in enforce mode. ***For Solaris:*** The evaluator shall ensure that the application can run with Solaris Trusted Extensions enabled and enforcing. ***For macOS:*** The evaluator shall ensure that the application can successfully run on macOS without disabling any security features.
	The application shall not write user-modifiable files to directories that contain	The evaluator shall run the application and determine where it writes its files. For

ID	Requirement	Assurance Activity
	executable files unless explicitly directed by the user to do so. **Application Note:** The purpose of this requirement is to help ensure the integrity of application binaries by supporting file protection mechanisms such as directory-level file permissions and application whitelisting. A user-modifiable file for purposes of this requirement is a file that is writable by an unprivileged user of the application -- either directly through application execution or independently of the application. If the application runs in the context of the application user, then the application should not be able to write to the directory containing the application binaries -- regardless of whether the files are configuration data, audit data, or temporary files. Executables and user-modifiable files may not share the same parent directory, but may share directories above the parent.	files where the user does not choose the destination, the evaluator shall check whether the destination directory contains executable files. This varies per platform: ***For Android:*** The evaluator shall run the program, mimicking normal usage, and note where all user-modifiable files are written. The evaluator shall ensure that there are no executable files stored under /data/data/package/ where *package* is the Java package of the application. ***For Windows:*** For Windows Universal Applications the evaluator shall consider the requirement met because the platform forces applications to write all data within the application working directory (sandbox). For Windows Desktop Applications the evaluator shall run the program, mimicking normal usage, and note where all user-modifiable files are written. The evaluator shall ensure that there are no executable files stored in the same directories to which the application wrote user-modifiable files. ***For iOS:*** The evaluator shall consider the requirement met because the platform forces applications to write all data within the application working directory (sandbox). ***For Linux:*** The evaluator shall run the program, mimicking normal usage, and note where all user-modifiable files are written. The evaluator shall ensure that there are no executable files stored in the same directories to which the application wrote user-modifiable files. ***For Solaris:*** The evaluator shall run the program, mimicking normal usage, and note where all user-modifiable files are written. The evaluator shall ensure that there are no executable files stored in the same directories to which the application wrote user-modifiable files. ***For macOS:*** The evaluator shall run the program, mimicking normal usage, and note where all user-modifiable files are written. The evaluator shall ensure that there are no executable files stored in the same directories to which the application wrote user-modifiable files.
	The application shall be built with stack-based buffer overflow protection enabled.	The evaluator will inspect every native executable included in the TOE to ensure that stack-based buffer overflow protection is present. ***For Windows:*** Applications that run as Managed Code in the .NET Framework do not require these stack protections. Applications developed in Object Pascal using the Delphi IDE compiled with RangeChecking enabled comply with this element. For other code, the evaluator shall review the TSS and verify that the /GS flag was used during compilation. The evaluator shall run a tool like, BinScope, that can verify the correct usage of /GS. ***For PE*** , the evaluator will disassemble each and ensure the following sequence appears: mov rcx, QWORD PTR [rsp+*(...)*] xor rcx, *(...)* call *(...)* . ***For ELF executables***, the evaluator will ensure that each contains references to the symbol **__stack_chk_fail**. Tools such as Canary Detector may help automate these activities.
	The application shall [**selection**: *provide the ability*, *leverage the platform*] to check for updates and patches to the application software. **Application Note:** This requirement is about the ability to "check" for updates. The actual installation of any updates should be done by the platform. This requirement is intended to ensure that the application can check for updates provided by the vendor, as updates provided by another source may contain malicious code.	The evaluator shall check to ensure the guidance includes a description of how updates are performed. The evaluator shall check for an update using procedures described in either the application documentation or the platform documentation and verify that the application does not issue an error. If it is updated or if it reports that no update is available this requirement is considered to be met.
	The application shall [**selection**: *provide the ability*, *leverage the platform*] to query the current version of the application software.	The evaluator shall verify guidance includes a description of how to query the current version of the application. The evaluator shall query the application for the current version of the software according to the operational user guidance. The evaluator shall then verify that the current version matches that of the documented and installed version.
	The application shall not download, modify, replace or update its own binary code. **Application Note:** This requirement applies to the code of the application; it does not apply to mobile code technologies that are designed for download and execution by the application.	The evaluator shall verify that the application's executable files are not changed by the application. The evaluator shall complete the following test: • **Test 1:** The evaluator shall install the application and then locate all of its executable files. The evaluator shall then, for each file, save off either a hash of the file or a copy of the file itself. The evaluator shall then run the application and exercise all features of the application as described in the ST. The evaluator shall then compare each executable file with the either the saved hash or the saved copy of the files. The evaluator shall verify that these are identical.
	The application installation package and its updates shall be digitally signed such that its platform can cryptographically verify them prior to installation. **Application Note:** The specifics of the verification of installation packages and updates involves requirements on the platform (and not the application), so these are not fully specified here.	The evaluator shall verify that the TSS identifies how the application installation package and updates to it are signed by an authorized source. The definition of an authorized source must be contained in the TSS. The evaluator shall also ensure that the TSS (or the operational guidance) describes how candidate updates are obtained.
	The application is distributed [**selection**: *with the platform OS* , *as an additional software package to the platform OS*]. **Application Note:** Application software that is distributed as part of the platform operating system is not required to be package for installation or uninstallation. If "as an additional software package to the OS" is selected the requirements from FPT_TUD_EXT.2 must be included in the ST.	The evaluator shall verify that the TSS identifies how the application is distributed. If "with the platform" is selected the evaluated shall perform a clean installation or factory reset to confirm that TOE software is included as part of the platform OS. If "as an additional package" is selected the evaluator shall perform the tests in FPT_TUD_EXT.2.
	The application shall be distributed using the format of the platform-supported package manager. ***This is a selection-based requirement. Its inclusion depends upon selection in .***	The evaluator shall verify that application updates are distributed in the format supported by the platform. This varies per platform: ***For Android:*** The evaluator shall ensure that the application is packaged in the Android application package (APK) format. ***For Windows:*** The evaluator shall ensure that the application is packaged in the standard Windows Installer (.MSI) format, the Windows Application Software (.EXE) format signed using the Microsoft Authenticode process, or the Windows Universal Application package (.APPX) format. See https://msdn.microsoft.com/en-us/library/ms537364(v=vs.85).aspx for details regarding Authenticode signing. ***For iOS:*** The evaluator shall ensure that the application is packaged in the IPA format.

ID	Requirement	Assurance Activity
		For Linux: The evaluator shall ensure that the application is packaged in the format of the package management infrastructure of the chosen distribution. For example, applications running on Red Hat and Red Hat derivatives shall be packaged in RPM format. Applications running on Debian and Debian derivatives shall be packaged in DEB format. ***For Solaris:*** The evaluator shall ensure that the application is packaged in the PKG format. ***For macOS:*** The evaluator shall ensure that application is packaged in the DMG format, the PKG format, or the MPKG format.
	The application shall be packaged such that its removal results in the deletion of all traces of the application, with the exception of configuration settings, output files, and audit/log events. ***This is a selection-based requirement. Its inclusion depends upon selection in .*** **Application Note:** Applications software bundled with the system/firmware image are not subject to this requirement if the user is unable to remove the application through means provided by the OS.	***For Android:*** The evaluator shall consider the requirement met because the platform forces applications to write all data within the application working directory (sandbox). ***For iOS:*** The evaluator shall consider the requirement met because the platform forces applications to write all data within the application working directory (sandbox). ***For All Other Platforms:*** The evaluator shall record the path of every file on the entire filesystem prior to installation of the application, and then install and run the application. Afterwards, the evaluator shall then uninstall the application, and compare the resulting filesystem to the initial record to verify that no files, other than configuration, output, and audit/log files, have been added to the filesystem.
	The application shall be packaged with only [**assignment**: *list of third-party libraries*]. **Application Note:** The intention of this requirement is for the evaluator to discover and document whether the application is including unnecessary or unexpected third-party libraries. This includes adware libraries which could present a privacy threat, as well as ensuring documentation of such libraries in case vulnerabilities are later discovered.	The evaluator shall install the application and survey its installation directory for dynamic libraries. The evaluator shall verify that libraries found to be packaged with or employed by the application are limited to those in the assignment.
	The application shall be versioned with [**selection**: *SWID tags that comply with minimum requirements from ISO/IEC 19770-2:2015* , [***assignment***: *other version information*]] . **Application Note:** The use of SWID tag to identify application software is a requirement for DOD IT based on DoD Instruction 8500.01 which requires the use of SCAP which includes SWID tags per the NIST standard. The PP selection of "other version information" will be removed in the next major release of this protection profile. Vendors should begin to version software with valid SWID tags. Valid SWID tags must contain a SoftwareIdentity element and an Entity element as defined in the ISO/IEC 19770-2:2015 standard. SWID tags must be stored with a .swidtag file extensions as defined in the ISO/IEC 19770-2:2015.	If "other version information" is selected the evaluator shall verify that the TSS contains an explaination of the versioning methodology. The evaluator shall install the application, then check for the / existence of version information. If SWID tags is selected the evaluator shall check for a .swidtag file. The evaluator shall open the file and verify that is contains at least a SoftwareIdentity element and an Entity element.
	The application shall [**selection**: • *not transmit any [**selection**: data, sensitive data]* , • *encrypt all transmitted [**selection**: sensitive data, data] with [**selection**: HTTPS in accordance with FCS_HTTPS_EXT.1 , TLS as defined in the TLS Package, DTLS as defined in the TLS Package, SSH as conforming to the Extended Package for Secure Shell, IPsec as defined in the PP-Module for VPN Client]* , • *invoke platform-provided functionality to encrypt all transmitted sensitive data with [**selection**: HTTPS, TLS, DTLS, SSH]* , • *invoke platform-provided functionality to encrypt all transmitted data with [**selection**: HTTPS, TLS, DTLS, SSH]*] between itself and another trusted IT product. **Application Note:** Encryption is not required for applications transmitting data that is not sensitive. If *encrypt all transmitted* is selected and *TLS* is selected, then evaluation of elements from either FCS_TLSC_EXT.1 or FCS_TLSS_EXT.1 is required. If *encrypt all transmitted* is selected and *HTTPS* is selected, FCS_HTTPS_EXT.1 is required. If *encrypt all transmitted* is selected and *DTLS* is selected, FCS_DTLS_EXT.1 is required. If *encrypt all transmitted* is selected and *SSH* is selected, the TSF shall be validated against the *Extended Package for Secure Shell*. If *encrypt all trasnmitted* is selected and *IPsec* is selected, the TSF must claim conformance to a *PP-Configuration that includes the VPN Client PP-Module* If *encrypt all transmitted* is selected the corresponding FCS_COP.1 requirements will be included.	For platform-provided functionality, the evaluator shall verify the TSS contains the calls to the platform that TOE is leveraging to invoke the functionality. The evaluator shall perform the following tests. • **Test 1:** The evaluator shall exercise the application (attempting to transmit data; for example by connecting to remote systems or websites) while capturing packets from the application. The evaluator shall verify from the packet capture that the traffic is encrypted with HTTPS, TLS, DTLS, or SSH, or IPsec in accordance with the selection in the ST. • **Test 2:** The evaluator shall exercise the application (attempting to transmit data; for example by connecting to remote systems or websites) while capturing packets from the application. The evaluator shall review the packet capture and verify that no sensitive data is transmitted in the clear. • **Test 3:** The evaluator shall inspect the TSS to determine if user credentials are transmitted. If credentials are transmitted the evaluator shall set the credential to a known value. The evaluator shall capture packets from the application while causing credentials to be transmitted as described in the TSS. The evaluator shall perform a string search of the captured network packets and verify that the plaintext credential previously set by the evaluator is not found. ***For Android:*** If "not transmit any data" is selected, the evaluator shall ensure that the application's AndroidManifest.xml file does not contain a uses-permission or uses-permission-sdk-23 tag containing android:name="android.permission.INTERNET". In this case, it is not necessary to perform the above Tests 1, 2, or 3, as the platform will not allow the application to perform any network communication. ***For iOS:*** If "encrypt all transmitted data" is selected, the evaluator shall ensure that the application's Info.plist file does not contain the NSAllowsArbitraryLoads or NSExceptionAllowsInsecureHTTPLoads keys, as these keys disable iOS's Application Transport Security feature.

Security Assurance Requirements

ID	Requirement	Assurance Activity
	The developer shall provide a functional specification.	
	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	

ID	Requirement	Assurance Activity
	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.	
	The functional specification shall describe the purpose and method of use for each SFR-enforcing and SFR-supporting TSFI.	
	The functional specification shall identify all parameters associated with each SFR-enforcing and SFR-supporting TSFI.	
	The functional specification shall provide rationale for the implicit categorization of interfaces as SFR-non-interfering.	
	The tracing shall demonstrate that the SFRs trace to TSFIs in the functional specification.	
	The evaluator shall confirm that the information provided meets all requirements for content and presentation of evidence.	
	The evaluator shall 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.
	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. Where appropriate, the guidance documentation is expressed in the eXtensible Configuration Checklist Description Format (XCCDF) to support security automation. 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.	
	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.	
	The operational user guidance shall describe, for each user role, how to use the available interfaces provided by the TOE in a secure manner.	
	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.	
	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.	
	The operational user guidance shall identify all possible modes of operation of the TOE (including operation following failure or operational error), their consequences, and implications for maintaining secure operation.	
	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.	
	The operational user guidance shall be clear and reasonable.	
	The evaluator shall confirm that the information provided meets all requirements for content and presentation of evidence.	Some of the contents of the operational guidance will be verified by the evaluation activities in and evaluation of the TOE according to the [CEM]. The following additional information is also required. If cryptographic functions are provided by the TOE, the operational guidance shall contain instructions for configuring the cryptographic engine associated with the evaluated configuration of the TOE. 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 TOE. The documentation must describe the process for verifying updates to the TOE by verifying a digital signature – this may be

ID	Requirement	Assurance Activity
		done by the TOE or the underlying platform. The evaluator shall 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 TOE (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 digital signature. The TOE 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.
	The developer shall provide the TOE, 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.	
	The preparative procedures shall describe all the steps necessary for secure acceptance of the delivered TOE in accordance with the developer's delivery procedures.	
	The preparative procedures shall describe all the steps necessary for secure installation of the TOE and for the secure preparation of the operational environment in accordance with the security objectives for the operational environment as described in the ST.	
	The evaluator shall confirm that the information provided meets all requirements for content and presentation of evidence.	
	The evaluator shall apply the preparative procedures to confirm that the TOE 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 TOE functional requirements. The evaluator shall check to ensure that the guidance provided for the TOE adequately addresses all platforms claimed for the TOE in the ST.
	The developer shall provide the TOE and a reference for the TOE.	
	The application shall be labeled with a unique reference. **Application Note:** Unique reference information includes: • Application Name • Application Version • Application Description • Platform on which Application Runs • Software Identification (SWID) tags, if available	
	The evaluator shall confirm that the information provided meets all requirements for content and presentation of evidence.	The evaluator shall 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 shall check the AGD guidance and TOE 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 TOE, the evaluator shall examine the information on the web site to ensure that the information in the ST is sufficient to distinguish the product.
	The developer shall provide a configuration list for the TOE.	
	The configuration list shall include the following: the TOE itself; and the evaluation evidence required by the SARs.	
	The configuration list shall uniquely identify the configuration items.	
	The evaluator shall 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 TOE 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 shall 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 flags). The evaluator shall ensure that this documentation also includes an indication of whether such protections are on by default, or have to be specifically enabled. The evaluator shall 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.
	The developer shall provide a description in the TSS of how timely security updates are made to the TOE.	
	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.	
	The description shall include the process for creating and deploying security updates for the TOE software.	
	The description shall express the time window as the length of time, in days,	

ID	Requirement	Assurance Activity
	between public disclosure of a vulnerability and the public availability of security updates to the TOE.	
	The description shall include the mechanisms publicly available for reporting security issues pertaining to the TOE.	
	The evaluator *shall confirm* that the information provided meets all requirements for content and presentation of evidence.	The evaluator shall 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 shall verify that this description addresses the entire application. The evaluator shall also verify that, in addition to the TOE developer's process, any third-party processes are also addressed in the description. The evaluator shall also verify that each mechanism for deployment of security updates is described. The evaluator shall 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 TOE patching this vulnerability, to include any third-party or carrier delays in deployment. The evaluator shall verify that this time is expressed in a number or range of days. The evaluator shall verify that this description includes the publicly available mechanisms (including either an email address or website) for reporting security issues related to the TOE. 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.
	The developer shall provide the TOE for testing. **Application Note:** The developer must provide at least one product instance of the TOE for complete testing on at least platform regardless of equivalency. See the Equivalency Appendix for more details.	
	The TOE shall be suitable for testing.	
	The evaluator *shall confirm* that the information provided meets all requirements for content and presentation of evidence.	
	The evaluator shall test a subset of the TSF to confirm that the TSF operates as specified. **Application Note:** The evaluator shall test the application on the most current fully patched version of the platform.	The evaluator shall prepare a test plan and report documenting the testing aspects of the system, including any application crashes during testing. The evaluator shall determine the root cause of any application crashes and include that information in the report. The test plan covers all of the testing actions contained in the [CEM] and the body of this PP's 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 effect; 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 TOE 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 (e.g SSH). 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.
	The developer shall provide the TOE for testing.	
	The application shall be suitable for testing. **Application Note:** Suitability for testing means not being obfuscated or packaged in such a way as to disrupt either static or dynamic analysis by the evaluator.	
	The evaluator shall confirm that the information provided meets all requirements for content and presentation of evidence.	
	The evaluator shall perform a search of public domain sources to identify potential vulnerabilities in the TOE. **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.	
	The evaluator shall conduct penetration testing, based on the identified potential vulnerabilities, to determine that the TOE is resistant to attacks performed by an attacker possessing Basic attack potential.	The evaluator shall 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 shall also run a virus scanner with the most current virus definitions against the application files and verify that no files are flagged as malicious. 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