www.GossamerSec.com

Assurance Activity Report for Infinera Corporation Transcend Management System Client 18.10.3

Version 0.3
12/06/22

Prepared by:
Gossamer Security Solutions
Accredited Security Testing Laboratory – Common Criteria Testing
Columbia, MD 21045

Prepared for:
National Information Assurance Partnership
Common Criteria Evaluation and Validation Scheme

Document: AAR-VID11318

Revision History

Revision	Date	Authors	Summary
Version 0.1	11/11/22	Smoley	Initial draft
Version 0.2	12/02/22	Smoley	Addressed First Round ECR comments
Version 0.3	12/06/22	Smoley	Updated for Second Round ECR comments

The TOE Evaluation was Sponsored by:

Infinera Corporation
9005 Junction Drive, Suite C
Annapolis Junction, MD 20701

Evaluation Personnel:

- Raymond Smoley

Common Criteria Versions:

- Common Criteria for Information Technology Security Evaluation Part 1: Introduction, Version 3.1, Revision 5, April 2017
- Common Criteria for Information Technology Security Evaluation Part 2: Security functional components, Version 3.1, Revision 5, April 2017
- Common Criteria for Information Technology Security Evaluation Part 3: Security assurance components, Version 3.1, Revision 5, April 2017

Common Evaluation Methodology Versions:

- Common Methodology for Information Technology Security Evaluation, Evaluation Methodology, Version 3.1, Revision 5, April 2017

Table of Contents

1. Introduction

This document presents evaluations results of the Infinera Corporation Transcend Network Management System (TNMS) Client 18.10.3 ASPP14/ PKGTLS11 evaluation. This document contains a description of the assurance activities and associated results as performed by the evaluators.

1.1 Test Equivalence

The evaluator fully tested on the TNMS Client on a Windows 10 Home platform. The TOE also claims compatibility for any Windows 10 deployment and multiple processors. Since the TOE is a Java application which abstracts the function calls made by the TOE and the same TOE image is provided across different deployments, the evaluation team concludes that the evidence provided should be sufficient to match all of the deployments claimed in the ST.

Additionally, the evaluator made use of an external TNMS Server to produce test evidence. The TNMS Server is not a part of this evaluation as it is separately evaluated, and it is only used to demonstrate the correct operations of the TOE. All security requirements are met by the TOE completely. Functional behavior from the TNMS server (including server credentials and network element management) are not claimed under this evaluation.

1.2 CAVP Certificates

The TOE performs cryptographic algorithms in accordance with the following NIST standards and has received the following CAVP algorithm certificates.

SFR	Algorithm	NIST Standard	Cert#
FCS_CKM.1/AK (Key Gen)	ECDH Key Generation P-256, 384, 521	FIPS 186-4, ECDSA	A2313
	RSA IFC key generation 2048 bits	FIPS 186-4, RSA	A2313
FCS_CKM.2 (Key Establishment)	ECDH Key Exchange	SP 800-56A, KAS ECC	A2313
	RSA Key Exchange	N/A	
FCS_COP.1/SKC	AES 128/256 CBC, GCM	FIPS 197, SP 800-38A/D	A2313
FCS_COP.1/Hash	SHA Hashing SHA-1, 256, 384, 512	FIPS 180-4	A2313
FCS_COP.1/Sig	RSA Sign/Verify 2048 bits	FIPS 186-4, RSA	A2313
	ECDSA Sign/Verify P-256, 384, 521	FIPS 186-4, ECDSA	A2313
FCS_COP.1/KeyedHash	HMAC-SHA HMAC-SHA 1, 256, 384, 512	FIPS 198-1 & 180-4	A2313
FCS_RBG_EXT.2 (Random)	DRBG Bit Generation Hash DRBG 256 bits	SP 800-90A	A2313

Table 1-1 Bouncy Castle CAVP Certificates

2. Protection Profile SFR Assurance Activities

This section of the AAR identifies each of the assurance activities included in the claimed Protection Profile and describes the findings in each case.

The following evidence was used to complete the Assurance Activities:

AAR v0.2

- Infinera Corporation Transcend Network Management System Client 18.10.3 Security Target, Version 1.5 12/06/2022 **[ST]**
- Infinera Transcend Network Management System Client 18.10.3 Administrative Guidance for Common Criteria, Version 1.2, December 6st 2022 **[Admin Guide]**

2.1 Cryptographic support (FCS)

2.1.1 Cryptographic Key Generation Services (ASPP14:FCS_CKM.1)

2.1.1.1 ASPP14:FCS_CKM.1.1

TSS Assurance Activities: None Defined

Guidance Assurance Activities: None Defined

Testing Assurance Activities: None Defined

Component TSS Assurance Activities: 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.

Section 6.1 of the ST states that the TOE generates asymmetric RSA and ECDH keys during TLS connections to the TNMS server. The evaluator examined the AGD document and found references to algorithms used by TLS and SSH, however no other references to key generation outside of those claimed in the ST.

Component Guidance Assurance Activities: None Defined

Component Testing Assurance Activities: None Defined

2.1.2 Cryptographic Asymmetric Key Generation - per TD0659 (ASPP14:FCS_CKM.1/AK)

2.1.2.1 ASPP14:FCS_CKM.1/AK.1

TSS Assurance Activities: None Defined

Guidance Assurance Activities: None Defined

Testing Assurance Activities: None Defined

Component TSS Assurance Activities: 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.

Section 6.1 of the ST states that the TOE generates 2048-bit RSA keys P-256 and P-384 EDCHE keys as part of TLS secured connections to the TNMS server. The TOE does not claim to invoke any platform-provided functionality for asymmetric key generation.

Component Guidance Assurance Activities: 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.

The TOE generates asymmetric keys as part of TLS secured connections to the TNMS server. Section *3.5.4 TLS Configuration* notes that "Other than setting the trusted root certificates, TLS settings in TNMS Client – such as supported cipher suites and key sizes - are fixed and not configurable by the user."

Additionally, *Section 3.4 Enabling FIPS mode* indicates that the TOE installation requires setting up the cryptographic library into FIPS mode in order to be in compliance with the Common Criteria evaluated configuration. Additionally, this section also states that the use of any other cryptographic engines, or configurations other than what is described in this section was not evaluated nor tested during the Common Criteria Evaluation of TNMS Client.

Component Testing Assurance Activities: 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 <= e <= 2^256 and e is an odd integer,

|p-q| > 2^(nlen/2 - 100),

p >= 2^(nlen/2 -1/2),

q >= 2^(nlen/2 -1/2),

2^(nlen/2) < d < LCM(p-1,q-1),

e*d = 1 mod 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, P384 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., P256, 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 =x = q-1

len(q) + 64 bit output of RBG, followed by a mod q-1 operation where 1= x=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 not= 0,1

q divides p-1

g^q mod p = 1

g^x mod 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 TOE TNMS Client includes the CAVP-certified BC-FJA (Bouncy Castle FIPS Java API). See Section 1.2 for a listing of CAVP certificates.

2.1.3 Cryptographic Symmetric Key Generation (ASPP14:FCS_CKM.1/SK)

2.1.3.1 ASPP14:FCS_CKM.1/SK.1

TSS Assurance Activities: None Defined

Guidance Assurance Activities: None Defined

Testing Assurance Activities: None Defined

Component TSS Assurance Activities: 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.

Section 6.1 of the ST states that the TOE generates 128 and 256-bit AES keys during the TLS handshake and uses its Bouncy Castle cryptographic library to generate the random values used during the handshake.

In regards to how the functionality for random bit generation is invoked, the same section states the TOE's Bouncy Castle library calls the java.security.SecureRandom class [specifically calling SecureRandom.generateSeed()] to obtain a 256-bit seed, which is assumed to contain 256-bits of entropy.

Component Guidance Assurance Activities: None Defined

Component Testing Assurance Activities: None Defined

2.1.4 Cryptographic Key Establishment (ASPP14:FCS_CKM.2)

2.1.4.1 ASPP14:FCS_CKM.2.1

TSS Assurance Activities: None Defined

Guidance Assurance Activities: None Defined

Testing Assurance Activities: None Defined

Component TSS Assurance Activities: 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.

Section 6.1 of the ST states the TOE uses RSA-based key establishment and ECDHE as part of its TLS connection to the TNMS server. Those claims match the key generation schemes identified under FCS_CKM.1.1.

Component Guidance Assurance Activities: The evaluator shall verify that the AGD guidance instructs the administrator how to configure the TOE to use the selected key establishment scheme(s).

The TOE uses RSA-based key establishment and ECDHE as part of its TLS connection to the TNMS server. Section *3.5.4 TLS Configuration* notes that "Other than setting the trusted root certificates, TLS settings in TNMS Client – such as supported cipher suites and key sizes - are fixed and not configurable by the user."

Additionally, *Section 3.4 Enabling FIPS mode* indicates that the TOE installation requires setting up the cryptographic library into FIPS mode in order to be in compliance with the Common Criteria evaluated configuration. Additionally, this section also states that the use of any other cryptographic engines, or configurations other than what is described in this section was not evaluated nor tested during the Common Criteria Evaluation of TNMS Client.

Component Testing Assurance Activities: 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.

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 RSAESPKCS1-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 TOE TNMS Client includes the CAVP-certified BC-FJA (Bouncy Castle FIPS Java API). See Section 1.2 for a listing of CAVP certificates.

2.1.5 Cryptographic Operation - Hashing (ASPP14:FCS_COP.1/Hash)

2.1.5.1 ASPP14:FCS_COP.1/Hash.1

TSS Assurance Activities: None Defined

Guidance Assurance Activities: None Defined

Testing Assurance Activities: None Defined

Component TSS Assurance Activities: 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.

Section 6.1 of the ST states that the TOE uses SHA-1, SHA-256, SHA-384, and SHA-512 hashing when verifying the TLS server's authentication signature. The TOE also uses SHA-1 when hashing the user's password (combined with a 64-bit random salt).

Component Guidance Assurance Activities: None Defined

Component Testing Assurance Activities: 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.

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 TOE TNMS Client includes the CAVP-certified BC-FJA (Bouncy Castle FIPS Java API). See Section 1.2 for a listing of CAVP certificates.

2.1.6 Cryptographic Operation - Keyed-Hash Message Authentication - per TD0626 (ASPP14:FCS_COP.1/KeyedHash)

2.1.6.1 ASPP14:FCS_COP.1/KeyedHash.1

TSS Assurance Activities: None Defined

Guidance Assurance Activities: None Defined

Testing Assurance Activities: None Defined

Component TSS Assurance Activities: None Defined

Component Guidance Assurance Activities: None Defined

Component Testing Assurance Activities: 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.

The TOE TNMS Client includes the CAVP-certified BC-FJA (Bouncy Castle FIPS Java API). See Section 1.2 for a listing of CAVP certificates.

2.1.7 Cryptographic Operation - Signing (ASPP14:FCS_COP.1/Sig)

2.1.7.1 ASPP14:FCS_COP.1/Sig.1

TSS Assurance Activities: None Defined

Guidance Assurance Activities: None Defined

Testing Assurance Activities: None Defined

Component TSS Assurance Activities: None Defined

Component Guidance Assurance Activities: None Defined

Component Testing Assurance Activities: 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 TOE TNMS Client includes the CAVP-certified BC-FJA (Bouncy Castle FIPS Java API). See Section 1.2 for a listing of CAVP certificates.

2.1.8 Cryptographic Operation - Encryption/Decryption (ASPP14:FCS_COP.1/SKC)

2.1.8.1 ASPP14:FCS_COP.1/SKC.1

TSS Assurance Activities: None Defined

Guidance Assurance Activities: None Defined

Testing Assurance Activities: None Defined

Component TSS Assurance Activities: None Defined

Component Guidance Assurance Activities: 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 TOE generates 128 and 256-bit AES keys during the TLS handshake and uses its Bouncy Castle cryptographic library to generate the random values used during the handshake. Section *3.5.4 TLS Configuration* notes that "Other than setting the trusted root certificates, TLS settings in TNMS Client – such as supported cipher suites and key sizes - are fixed and not configurable by the user." Additionally, *Section 3.4 Enabling FIPS mode* indicates that the TOE installation requires setting up the cryptographic library into FIPS mode in order to be in compliance with the Common Criteria evaluated configuration. Additionally, this section also states that the use of any other cryptographic engines, or configurations other than what is described in this section was not evaluated nor tested during the Common Criteria Evaluation of TNMS Client

Component Testing Assurance Activities: 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].

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:

```
# Input: PT, IV, Key

for i = 1 to 1000:

if i == 1:

CT[1] = AES-CBC-Encrypt(Key, IV, PT)

PT = IV

else:

CT[i] = AES-CBC-Encrypt(Key, PT)

PT = CT[i-1]
```

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

AES-CCM Tests

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

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

Keys: All supported and selected key sizes (e.g., 128, 256 bits).

Associated Data: Two or three values for associated data length: The minimum (. 0 bytes) and maximum (. 32 bytes) supported associated data lengths, and 2^16 (65536) bytes, if supported.

Payload: Two values for payload length: The minimum (. 0 bytes) and maximum (. 32 bytes) supported payload lengths.

Nonces: All supported nonce lengths (7, 8, 9, 10, 11, 12, 13) in bytes.

Tag: All supported tag lengths (4, 6, 8, 10, 12, 14, 16) in bytes.

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

Variable Associated Data Test

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

Variable Payload Test

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

Variable Nonce Test

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

Variable Tag Test

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

Decryption-Verification Process Test

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

AES-CTR Tests

Test 1: Known Answer Tests (KATs)

There are four Known Answer Tests (KATs) described below. For all KATs, the plaintext, IV, and ciphertext values shall be 128-bit blocks. The results from each test may either be obtained by the validator 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.

To test the encrypt functionality, the evaluator shall supply a set of 10 plaintext values and obtain the ciphertext value that results from 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, the evaluator shall perform the same test as for encrypt, using 10 ciphertext values as input.

To test the encrypt functionality, the evaluator shall supply a set of 10 key values and obtain the ciphertext value that results from encryption of an all zeros plaintext using the given key value and an IV of all zeros. Five of the key values shall be 128-bit keys, and the other five shall be 256-bit keys. To test the decrypt functionality, the evaluator shall perform the same test as for encrypt, using an all zero ciphertext value as input.

To test the encrypt functionality, the evaluator shall supply the two sets of key values described below and obtain the ciphertext values that result from AES encryption of an all zeros plaintext using the given key values an an IV of all zeros. The first set of keys shall have 128 128-bit keys, and the second 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, the evaluator shall supply the two sets of key and ciphertext value pairs described below and obtain the plaintext value that results from decryption of the given ciphertext using the given key values 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 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.

To test the encrypt functionality, the evaluator shall supply the set of 128 plaintext values described below and obtain the two ciphertext values that result from encryption of the given plaintext using a 128-bit key value of all zeros and using a 256 bit key value of all zeros, respectively, and an IV of all zeros. Plaintext value i in each set shall have the leftmost bits be ones and the rightmost 128-i bits be zeros, for i in [1, 128]. To test the decrypt functionality, 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.

Test 2: Multi-Block Message Test

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

Test 3: Monte-Carlo Test

For AES-CTR mode perform the Monte Carlo Test for ECB Mode on the encryption engine of the counter mode implementation. There is no need to test the decryption engine.

The evaluator shall test the encrypt functionality using 200 plaintext/key pairs. 100 of these shall use 128 bit keys, and 100 of these shall use 256 bit keys. The plaintext values shall be 128-bit blocks. For each pair, 1000 iterations shall be run as follows:

For AES-ECB mode

```
# Input: PT, Key

for i = 1 to 1000:

CT[i] = AES-ECB-Encrypt(Key, PT)

PT = CT[i]
```

The ciphertext computed in the 1000th iteration 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 TOE TNMS Client includes the CAVP-certified BC-FJA (Bouncy Castle FIPS Java API). See Section 1.2 for a listing of CAVP certificates.

2.1.9 Random Bit Generation Services (ASPP14:FCS_RBG_EXT.1)

2.1.9.1 ASPP14:FCS_RBG_EXT.1.1

TSS Assurance Activities: None Defined

Guidance Assurance Activities: None Defined

Testing Assurance Activities: None Defined

Component TSS Assurance Activities: 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.

The ST claims that the TOE both invokes the platform-provided DRBG functionality and implements DRBG functionality as Section 6.1 states that the TOE invokes the platforms-provided DRBG functionality in order to obtain seeding material for its DRBG. As a result, the additional SFR FCS_RBG_EXT.2 was included in the ST.

Section 6.1 also states that the TOE implements a DRBG in its Bouncy Castle library and uses that DRBG when generating per-user salts as well as when generating random values used in TLS handshakes and PKBDF credential transformation.

Further information is shared under FCS_RBG_EXT.2 under Section 6.1 of the ST including the API used as the ST states the TOE obtains entropy to seed its DRBG from the platform through the BCryptGenRandom Windows API.

Component Guidance Assurance Activities: None Defined

Component Testing Assurance Activities: 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 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:

Platforms: 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.

Platforms: Microsoft 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.

Platforms: Apple iOS....

The evaluator shall verify that the application invokes either SecRandomCopyBytes, CCRandomGenerateBytes or CCRandomCopyBytes, or uses /dev/random directly to acquire random.

Platforms: Linux....

The evaluator shall verify that the application collects random from /dev/random or /dev/urandom.

Platforms: Oracle Solaris....

The evaluator shall verify that the application invokes either CCRandomGenerateBytes or CCRandomCopyBytes, or collects random from /dev/random.

Platforms: Apple macOS....

The evaluator shall verify that the application invokes either CCRandomGenerateBytes or CCRandomCopyBytes, or 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 evaluator examined the Java Virtual Machine to view the list of loaded modules and found the BCrypt.dll module loaded. The evaluator noted that this is the module that contains the Windows API for BCryptGenRandom referenced in the test assurance activity. The evaluator used an additional tool that scanned the Java application and listed each function that was imported from the referenced module and found that Java was importing BCryptGenRandom.

2.1.10 Random Bit Generation from Application (ASPP14:FCS_RBG_EXT.2)

2.1.10.1 ASPP14:FCS_RBG_EXT.2.1

TSS Assurance Activities: None Defined

Guidance Assurance Activities: None Defined

Testing Assurance Activities: 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 NISTRecommended 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 â€“ 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 â€“ 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.

Additional input: the additional input bit lengths have the same defaults and restrictions as the personalization string lengths.

The TOE TNMS Client includes the CAVP-certified BC-FJA (Bouncy Castle FIPS Java API). See Section 1.2 for a listing of CAVP certificates.

2.1.10.2 ASPP14:FCS_RBG_EXT.2.2

TSS Assurance Activities: Documentation shall be produced - and the evaluator shall perform the activities - in accordance with Appendix D - Entropy Documentation and Assessment and the Clarification to the Entropy Documentation and Assessment Annex.

Entropy documentation has been provided to NIAP. Additionally, section 6.1 of the ST summarizes the DRGB functionality.

Guidance Assurance Activities: None Defined

Testing Assurance Activities: In the future, specific statistical testing (in line with NIST SP 800-90B) will be required to verify the entropy estimates.

No further testing activities are currently required.

Component TSS Assurance Activities: None Defined

Component Guidance Assurance Activities: None Defined

Component Testing Assurance Activities: None Defined

2.1.11 Storage of Credentials (ASPP14:FCS_STO_EXT.1)

2.1.11.1 ASPP14:FCS_STO_EXT.1.1

TSS Assurance Activities: None Defined

Guidance Assurance Activities: None Defined

Testing Assurance Activities: None Defined

Component TSS Assurance Activities: 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 this requirement, the ST claims that the TOE application shall not store any credentials to non-volatile memory. Section 6.1 of the ST is consistent with claim as it states the TOE does not store any credentials.

Component Guidance Assurance Activities: None Defined

Component Testing Assurance Activities: For all credentials for which the application implements functionality, the evaluator shall verify credentials are encrypted according to FCS_COP.1/SKC or conditioned according to FCS_CKM.1.1/AK and FCS_CKM.1/PBKDF. For all credentials for which the application invokes platform-provided functionality, the evaluator shall perform the following actions which vary per platform.

Platforms: Android....

The evaluator shall verify that the application uses the Android KeyStore or the Android KeyChain to store certificates.

Platforms: Microsoft 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.

Platforms: Apple iOS....

The evaluator shall verify that all credentials are stored within a Keychain.

Platforms: Linux....

The evaluator shall verify that all keys are stored using Linux keyrings.

Platforms: Oracle Solaris....

The evaluator shall verify that all keys are stored using Solaris Key Management Framework (KMF).

Platforms: Apple macOS....

The evaluator shall verify that all credentials are stored within Keychain

The TOE does not claim to store any credentials. As a result, there are no credentials for which this test applies.

2.1.12 TLS Protocol (PKGTLS11:FCS_TLS_EXT.1)

2.1.12.1 PKGTLS11:FCS_TLS_EXT.1.1

TSS Assurance Activities: None Defined

Guidance Assurance Activities: The evaluator shall ensure that the selections indicated in the ST are consistent with selections in the dependent components.

The evaluator examined both the ST and the AGD and only found claims to the TOE operating as a TLS Client. As a result, PKGTLS11:FCS_TLSC_EXT.1 is claimed in the ST. Additionally, since the TOE claims elliptic curve under PKGTLS11:FCS_TLSC_EXT.1.1, the evaluator ensured that PKGTLS11:FCS_TLSC_EXT.5 was included in the ST and in testing. The evaluator found no references in the ST or AGD to indicate support for mutual authentication (PKGTLS11:FCS_TLSC_EXT.2), limiting of hashing algorithms under the Client Hello signature algorithms extension (PKGTLS11:FCS_TLSC_EXT.3), renegotiation (PKGTLS11:FCS_TLSC_EXT.4), TLS as a server

(PKGTLS11:FCS_TLSS_EXT.*), or DTLS (PKGTLS11:FCS_DTLS*). The evaluator further examined the TSS and AGD and found no claims that contradicted any of the more specific claims under this SFR.

Testing Assurance Activities: None Defined

Component TSS Assurance Activities: None Defined

Component Guidance Assurance Activities: None Defined

Component Testing Assurance Activities: None Defined

2.1.13 TLS Client Protocol (PKGTLS11:FCS_TLSC_EXT.1)

2.1.13.1 PKGTLS11:FCS_TLSC_EXT.1.1

TSS Assurance Activities: The evaluator shall check the description of the implementation of this protocol in the TSS to ensure that the cipher suites supported are specified. The evaluator shall check the TSS to ensure that the cipher suites specified include those listed for this component.

Section 6.1 of the ST states the TOE supports the cipher suites selected in section **Error! Reference source not found.** (which is a reference to the cipher suite claim under PKGTLS11:FCS_TLSC_EXT.1.1).

Guidance Assurance Activities: The evaluator shall also check the operational guidance to ensure that it contains instructions on configuring the product so that TLS conforms to the description in the TSS.

The evaluator followed the TLS configuration described in the AGD under *Section 3.5.4 TLS Configuration*. As stated in that section, Section 3.5.4 TLS Configuration notes that "Other than setting the trusted root certificates, TLS settings in TNMS Client – such as supported cipher suites and key sizes - are fixed and not configurable by the user." The evaluator also referred to *Section 3.5 Managing Root Certificates* in general as this section was directly referenced by the previous quote. The evaluator followed the above sections to ensure that the list of trusted credentials was correctly configured, however, no additional steps were required to ensure that TLS conforms to the description provided in the TSS or the configuration used for testing.

Testing Assurance Activities: The evaluator shall also perform the following tests:

Test 1: The evaluator shall establish a TLS connection using each of the cipher suites specified by the requirement. This connection may be established as part of the establishment of a higher-level protocol, e.g., as part of an EAP session. It is sufficient to observe the successful negotiation of a cipher suite to satisfy the intent of the test; it is not necessary to examine the characteristics of the encrypted traffic in an attempt to discern the cipher suite being used (for example, that the cryptographic algorithm is 128-bit AES and not 256-bit AES).

Test 2: The goal of the following test is to verify that the TOE accepts only certificates with appropriate values in the extendedKeyUsage extension, and implicitly that the TOE correctly parses the extendedKeyUsage extension as part of X.509v3 server certificate validation. The evaluator shall attempt to establish the connection using a server with a server certificate that contains the Server Authentication purpose in the extendedKeyUsage extension and verify that a connection is established. The evaluator shall repeat this test using a different, but otherwise valid and trusted, certificate that lacks the Server Authentication purpose in the extendedKeyUsage extension and ensure that a connection is not established. Ideally, the two certificates should be similar in structure, the types of identifiers used, and the chain of trust.

Test 3: The evaluator shall send a server certificate in the TLS connection that does not match the server-selected cipher suite (for example, send a ECDSA certificate while using the TLS_RSA_WITH_AES_128_CBC_SHA cipher suite or send a RSA certificate while using one of the ECDSA cipher suites.) The evaluator shall verify that the product disconnects after receiving the server's Certificate handshake message.

Test 4: The evaluator shall configure the server to select the TLS_NULL_WITH_NULL_NULL cipher suite and verify that the client denies the connection.

Test 5: The evaluator shall perform the following modifications to the traffic:

Test 5.1: Change the TLS version selected by the server in the Server Hello to an undefined TLS version (for example 1.5 represented by the two bytes 03 06) and verify that the client rejects the connection.

Test 5.2: Change the TLS version selected by the server in the Server Hello to the most recent unsupported TLS version (for example 1.1 represented by the two bytes 03 02) and verify that the client rejects the connection.

Test 5.3: [conditional] If DHE or ECDHE cipher suites are supported, modify at least one byte in the server's nonce in the Server Hello handshake message, and verify that the client does not complete the handshake and no application data flows.

Test 5.4: Modify the server's selected cipher suite in the Server Hello handshake message to be a cipher suite not presented in the Client Hello handshake message. The evaluator shall verify that the client does not complete the handshake and no application data flows.

Test 5.5: [conditional] If DHE or ECDHE cipher suites are supported, modify the signature block in the server's Key Exchange handshake message, and verify that the client does not complete the handshake and no application data flows. This test does not apply to cipher suites using RSA key exchange. If a TOE only supports RSA key exchange in conjunction with TLS, then this test shall be omitted.

Test 5.6: Modify a byte in the Server Finished handshake message, and verify that the client does not complete the handshake and no application data flows.

Test 5.7: Send a message consisting of random bytes from the server after the server has issued the Change Cipher Spec message and verify that the client does not complete the handshake and no application data flows. The message must still have a valid 5-byte record header in order to ensure the message will be parsed as TLS.

Test 1 – The evaluator made a TLS connection using each of the claimed ciphersuites. The evaluator was able to capture each ciphersuite using a packet capture

Test 2 – The evaluator configured a server to use TLS. The evaluator then attempted to connect to the TLS server using a server certificate with a valid serverAuth EKU and showed this connection was accepted. The evaluator then attempted to connect to a server using a certificate that lacks server authentication purpose in the EKU field and the attempt was rejected.

Test 3 – The evaluator configured a server to use an RSA TLS server certificate, but negotiate the incompatible TLS_ECDHE_RSA_WITH_AES_128_GCM_SHA256 ciphersuite. The evaluator attempted to connect the TOE to this server and show that the connection was rejected.

Test 4 – The evaluator configured a server to use TLS and to require the TLS_NULL_WITH_NULL_NULL ciphersuite. The evaluator attempted to connect the TOE to the server and the attempt was rejected.

Test 5.1 – The evaluator configured the TOE to connect to a TLS server that only supported a non-TOE-supported TLS version (version 1.5 represented by two byes 0x0306 in the TLS Server Hello Message). The connection attempt was rejected.

Test 5.2 – The TOE supports TLS 1.2. The evaluator attempted to connect the TOE to a TLS server that only supported the most-recent non-TOE-supported TLS version, version 1.1 represented by 0x0302 in the TLS Server Hello Message. The evaluator also attempted to connect the TOE to a TLS server that only supported TLS version 1.0, represented by 0x0301 in the TLS Server Hello Message. Both connection attempts were rejected.

Test 5.3 – The evaluator attempted to connect the TOE to a TLS server using Corrupting the first byte of sent server hello nonce. The connection attempt was rejected.

Test 5.4 – The evaluator attempted to connect the TOE to a TLS server that was configured to ignore the TLS Client Hello cipher suites and only use a non-supported cipher suite, TLS_KRB5_EXPORT_WITH_DES_CBC_40_MD5. The connection attempt was rejected.

Test 5.5 – The evaluator attempted to connect the TOE to a TLS server. The evaluator modified the TLS server's key exchange message to change the first byte from 0x5f to 0xb4. The evaluator found that the connection attempt was rejected.

Test 5.6 – The evaluator attempted to connect the TOE to a TLS server. The evaluator modified the TLS server's finished handshake message MAC (0x17 to 0xe8). The evaluator found that the connection attempt was rejected.

Test 5.7 – The evaluator attempted to connect the TOE to a TLS server. The evaluator modified the TLS server's finished handshake message to consist of random data. The evaluator found that the connection attempt was rejected and no application data flowed.

2.1.13.2 PKGTLS11:FCS_TLSC_EXT.1.2

TSS Assurance Activities: The evaluator shall ensure that the TSS describes the client's method of establishing all reference identifiers from the application-configured reference identifier, including which types of reference identifiers are supported (e.g. Common Name, DNS Name, URI Name, Service Name, or other application-specific Subject Alternative Names) and whether IP addresses and wildcards are supported. The evaluator shall ensure that this description identifies whether and the manner in which certificate pinning is supported or used by the product.

Section 6.1 of the ST states the TOE uses the user provided server name (either a FQDN or an IP address) when checking the server's certificate for the expected reference identifier. The TOE supports Common Name, SAN:FQDN SAN:IP reference identifiers, and wildcards and does not support certificate/public key pinning.

Guidance Assurance Activities: The evaluator shall verify that the AGD guidance includes instructions for setting the reference identifier to be used for the purposes of certificate validation in TLS.

The TOE is only able to initiate connections to the external TNMS Server through its login prompts and all following connections are protected by TLS. The evaluator followed the information in *Section 3.9 TNMS Login* where the AGD notes how to initiate connections, including the required "Server name", "user name", and "password" fields. The evaluator noted that the AGD claims support for IP address or FQDN identifiers for the server's name.

Testing Assurance Activities: The evaluator shall configure the reference identifier according to the AGD guidance and perform the following tests during a TLS connection. If the TOE supports certificate pinning, all pinned certificates must be removed before performing Tests 1 through 6. A pinned certificate must be added prior to performing Test 7. (TD0499 applied)

Test 1: The evaluator shall present a server certificate that contains a CN that does not match the reference identifier and does not contain the SAN extension. The evaluator shall verify that the connection fails. Note that some systems might require the presence of the SAN extension. In this case the connection would still fail but for the reason of the missing SAN extension instead of the mismatch of CN and reference identifier. Both reasons are acceptable to pass Test 1.

Test 2: The evaluator shall present a server certificate that contains a CN that matches the reference identifier, contains the SAN extension, but does not contain an identifier in the SAN that matches the reference identifier. The evaluator shall verify that the connection fails. The evaluator shall repeat this test for each supported SAN type.

Test 3: [conditional] If the TOE does not mandate the presence of the SAN extension, the evaluator shall present a server certificate that contains a CN that matches the reference identifier and does not contain the SAN extension. The evaluator shall verify that the connection succeeds. If the TOE does mandate the presence of the SAN extension, this Test shall be omitted.

Test 4: The evaluator shall present a server certificate that contains a CN that does not match the reference identifier but does contain an identifier in the SAN that matches. The evaluator shall verify that the connection succeeds.

Test 5: The evaluator shall perform the following wildcard tests with each supported type of reference identifier. The support for wildcards is intended to be optional. If wildcards are supported, the first, second, and third tests below shall be executed. If wildcards are not supported, then the fourth test below shall be executed.

Test 5.1: [conditional]: If wildcards are supported, the evaluator shall present a server certificate containing a wildcard that is not in the left-most label of the presented identifier (e.g. foo.*.example.com) and verify that the connection fails.

Test 5.2: [conditional]: If wildcards are supported, the evaluator shall present a server certificate containing a wildcard in the left-most label but not preceding the public suffix (e.g. *.example.com). The evaluator shall configure the reference identifier with a single left-most label (e.g. foo.example.com) and verify that the connection succeeds. The evaluator shall configure the reference identifier without a left-most label as in the certificate (e.g. example.com) and verify that the connection fails. The evaluator shall configure the reference identifier with two left-most labels (e.g. bar.foo.example.com) and verify that the connection fails.

Test 5.3: [conditional]: If wildcards are supported, the evaluator shall present a server certificate containing a wildcard in the left-most label immediately preceding the public suffix (e.g. *.com). The evaluator shall configure the reference identifier with a single left-most label (e.g. foo.com) and verify that the connection fails. The evaluator shall configure the reference identifier with two left-most labels (e.g. bar.foo.com) and verify that the connection fails.

Test 5.4: [conditional]: If wildcards are not supported, the evaluator shall present a server certificate containing a wildcard in the left-most label (e.g. *.example.com). The evaluator shall configure the reference identifier with a single left-most label (e.g. foo.example.com) and verify that the connection fails.

Test 6: [conditional] If URI or Service name reference identifiers are supported, the evaluator shall configure the DNS name and the service identifier. The evaluator shall present a server certificate containing the correct DNS name and service identifier in the URIName or SRVName fields of the SAN and verify that the connection succeeds. The evaluator shall repeat this test with the wrong service identifier (but correct DNS name) and verify that the connection fails.

Test 7: [conditional] If pinned certificates are supported the evaluator shall present a certificate that does not match the pinned certificate and verify that the connection fails.

Test 1 – The evaluator attempted to connect the TOE to a TLS server using a certificate that contains a CN that does not match the reference identifier and does not contain the SAN extension. This evaluator observed that this connection was rejected.

Test 2 – This was tested alongside the previous test, FCS_TLSC_EXT.1-t1. The evaluator attempted to connect the TOE to a TLS server using a certificate that contains a CN that matches the reference identifier, contains the SAN extension, but does not contain an identifier in the SAN that matches the reference identifier. The evaluator observed that this connection was rejected.

Test 3 - This was tested alongside the previous test, FCS_TLSC_EXT.1-t1. The evaluator attempted to connect the TOE to a TLS server using a certificate that contains a CN that matches the reference identifier and does not contain the SAN extension. The evaluator observed that this connection was accepted.

Test 4 - This was tested alongside the previous test, FCS_TLSC_EXT.1-t1. The evaluator attempted to connect the TOE to a TLS server using a certificate that does not match the reference identifier but does contain an identifier in the SAN that matches. The evaluator observed that this connection was accepted.

Test 5 - This was tested alongside the previous test, FCS_TLSC_EXT.1-t1. The evaluator attempted to connect the TOE to a TLS server using a certificate that met the following requirements. The evaluator alternated between testing wildcard tests between CN and SAN identifiers and found identical behaviors for both.

Test 5.1 – The evaluator connected the TOE to a TLS server using certificates containing a wildcard that is not in the left-most label of the presented identifier (foo.*.example.com) and both connections failed.

Test 5.2 - The evaluator connected the TOE to a TLS server using certificates containing a wildcard in the left-most label but not preceding the public suffix (*.example.com). The evaluator shall configure the reference identifier with a single left-most label (foo.example.com) and both connections succeeded.

Using the same TLS server certificates, the evaluator attempted to connect to the reference identifier without a left-most label as in the certificate (example.com) and both connections failed.

Using the same TLS server certificates, the evaluator attempted to connect to the reference identifier with two left-most labels (bar.foo.example.com) and both that the connection failed.

Test 5.3 - The evaluator connected the TOE to a TLS server using certificates containing a wildcard in the left-most label immediately preceding the public suffix (*.com). The evaluator attempted to connect the TOE using a reference identifier with a single left-most label (foo.com) and observed both connections failed. The evaluator attempted to configure the reference identifier with two left-most labels (bar.foo.com) and observed both connections failed.

Test 5.4 – Not applicable, the TOE does support wildcards in both the CN and SAN fields.

Test 6 – Not applicable, the TOE does not claim support for URI or Service name reference identifiers.

Test 7 – Not applicable, the TOE does not support pinned certificates.

2.1.13.3 PKGTLS11:FCS_TLSC_EXT.1.3

TSS Assurance Activities: If the selection for authorizing override of invalid certificates is made, then the evaluator shall ensure that the TSS includes a description of how and when user or administrator authorization is obtained. The evaluator shall also ensure that the TSS describes any mechanism for storing such authorizations, such that future presentation of such otherwise-invalid certificates permits establishment of a trusted channel without user or administrator action.

The ST does not claim any exceptions for overriding invalid certificates. As a result, the TSS does not provided additional descriptions on how authorization is obtained or how authorizations are stored.

Guidance Assurance Activities: None Defined

Testing Assurance Activities: The evaluator shall demonstrate that using an invalid certificate results in the function failing as follows, unless excepted:

Test 1a: The evaluator shall demonstrate that a server using a certifcate with a valid certification path successfully connects.

Test 1b: The evaluator shall modify the certificate chain used by the server in test 1a to be invalid and demonstrate that a server using a certificate without a valid certification path to a trust store element of the TOE results in an authentication failure.

Test 1c [conditional]: If the TOE trust store can be managed, the evaluator shall modify the trust store element used in Test 1a to be untrusted and demonstrate that a connection attempt from the same server used in Test 1a results in an authentication failure.

(TD0513 applied)

Test 2: The evaluator shall demonstrate that a server using a certificate which has been revoked results in an authentication failure.

Test 3: The evaluator shall demonstrate that a server using a certificate which has passed its expiration date results in an authentication failure.

Test 4: The evaluator shall demonstrate that a server using a certificate which does not have a valid identifier results in an authentication failure.

Test 1a – The evaluator attempted to connect the TOE to a TLS server using a certificate with a valid certification path. The evaluator observed the connection was successful.

Test 1b – The evaluator attempted to connect the TOE to a TLS server using a certificate with an invalid certification path. The evaluator observed that the connection failed.

Test 1c – The evaluator returned to the valid certificate from Test 1a and managed the TOE's trust store to remove the trusted root so that the TOE could not validate the certification path. The evaluator attempted to connect the TOE to the TLS server and observed the connection failed.

Test 2 – The evaluator tested this under ASPP14:FIA_X509_EXT.1.1-t3. Under the referenced test, the evaluator attempted to connect the TOE to a TLS server using a revoked server certificate. The resulting connection was rejected.

Test 3: The evaluator tested this under ASPP14:FIA_X509_EXT.1.1-t2. Under the referenced test, the evaluator attempted to connect the TOE to a TLS server using an expired server certificate. The resulting connection was rejected.

Test 4: The evaluator tested this under PKGTLS11:FCS_TLSC_EXT.1.2. Under these referenced tests, the evaluator attempted to connect the TOE to a TLS server using a variety of configured identifiers and server certificates. The evaluator found that all cases where the TLS server was using an invalid certificate identifier, the connection was rejected.

Component TSS Assurance Activities: None Defined

Component Guidance Assurance Activities: None Defined

Component Testing Assurance Activities: None Defined

2.1.14 TLS Client Support for Supported Groups Extension (PKGTLS11:FCS_TLSC_EXT.5)

2.1.14.1 PKGTLS11:FCS_TLSC_EXT.5.1

TSS Assurance Activities: The evaluator shall verify that TSS describes the Supported Groups Extension.

Section 6.1 of the ST states the TOE supports the ECDHE groups secp256r1, secp384r1, and secp521r1. This is consistent with the claims for the same curves under PKGTLS11:FCS_TLSC_EXT.5.1 for supported group extension.

Guidance Assurance Activities: None Defined

Testing Assurance Activities: The evaluator shall also perform the following test:

Test 1: The evaluator shall configure a server to perform key exchange using each of the TOE's supported curves and/or groups. The evaluator shall verify that the TOE successfully connects to the server.

The evaluator made a TLS connection using each of the claimed curves. The evaluator was able to capture each key size or curve using a packet capture.

Component TSS Assurance Activities: None Defined

Component Guidance Assurance Activities: None Defined

Component Testing Assurance Activities: None Defined

2.2 User data protection (FDP)

2.2.1 Encryption Of Sensitive Application Data (ASPP14:FDP_DAR_EXT.1)

2.2.1.1 ASPP14:FDP_DAR_EXT.1.1

TSS Assurance Activities: None Defined

Guidance Assurance Activities: None Defined

Testing Assurance Activities: None Defined

Component TSS Assurance Activities: 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.

The TOE claims to not store any sensitive data to non-volatile memory. Section 6.2 of the ST also is consistent with this claim as it describes how sensitive data cannot be written to non-volatile memory as it states the TOE does not store any sensitive data and that sensitive data is processed by the external TNMS server.

Component Guidance Assurance Activities: None Defined

Component Testing Assurance Activities: 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.

Platforms: 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.

Platforms: Microsoft 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.

Platforms: Apple 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.

Platforms: 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.

Platforms: Oracle 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.

Platforms: Apple 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.

Not applicable, the TOE does not store any sensitive data to non-volatile memory and as a result the evaluation activity applies to no files.

2.2.2 Access to Platform Resources (ASPP14:FDP_DEC_EXT.1)

2.2.2.1 ASPP14:FDP_DEC_EXT.1.1

TSS Assurance Activities: None Defined

Guidance Assurance Activities: 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.

Section 2.2 Other Hardware and Software resources of the AGD indicates that the "TNMS Client does not require access to any sensitive information repositories or special hardware resources other than network connectivity used to communicate with the TNMS Server or check for updates.."

Testing Assurance Activities: Platforms: Android....

The evaluator shall verify that each uses-permission entry in the AndroidManifest.xml file for access to a hardware resource is reflected in the selection.

Platforms: Microsoft 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.

Platforms: Apple iOS....

The evaluator shall verify that either the application or the documentation provides a list of the hardware resources it accesses.

Platforms: Linux....

The evaluator shall verify that either the application software or its documentation provides a list of the hardware resources it accesses.

Platforms: Oracle Solaris....

The evaluator shall verify that either the application software or its documentation provides a list of the hardware resources it accesses.

Platforms: Apple macOS....

The evaluator shall verify that either the application software or its documentation provides a list of the hardware resources it accesses.

The TOE only claims access to network connectivity and does not make any claims for access to additional hardware resources. The evaluator never saw any special requests for hardware resources during any of the installation or operational screens for the TOE.

2.2.2.2 ASPP14:FDP_DEC_EXT.1.2

TSS Assurance Activities: None Defined

Guidance Assurance Activities: 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.

Section 2.2 Other Hardware and Software resources of the AGD indicates that the TNMS Client does not require access to any sensitive information repositories or special hardware resources other than network connectivity used to communicate with the TNMS Server or check for updates..

Testing Assurance Activities: Platforms: Android....

The evaluator shall verify that each uses-permission entry in the AndroidManifest.xml file for access to a sensitive information repository is reflected in the selection.

Platforms: Microsoft 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.

Platforms: Apple iOS....

The evaluator shall verify that either the application software or its documentation provides a list of the sensitive information repositories it accesses.

Platforms: Linux....

The evaluator shall verify that either the application software or its documentation provides a list of sensitive information repositories it accesses.

Platforms: Oracle Solaris....

The evaluator shall verify that either the application software or its documentation provides a list of sensitive information repositories it accesses.

Platforms: Apple macOS....

The evaluator shall verify that either the application software or its documentation provides a list of sensitive information repositories it accesses.

The TOE does not make any claims for access to additional sensitive information repositories. The evaluator never saw any special requests for sensitive data repositories during any of the installation or operational screens for the TOE.

Component TSS Assurance Activities: None Defined

Component Guidance Assurance Activities: None Defined

Component Testing Assurance Activities: None Defined

2.2.3 Network Communications (ASPP14:FDP_NET_EXT.1)

2.2.3.1 ASPP14:FDP_NET_EXT.1.1

TSS Assurance Activities: None Defined

Guidance Assurance Activities: None Defined

Testing Assurance Activities: None Defined

Component TSS Assurance Activities: None Defined

Component Guidance Assurance Activities: None Defined

Component Testing Assurance Activities: 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).

Platforms: 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.

Test 1 – The evaluator captured traffic while the TOE was used to login to the external TNMS server, manage server configurations, and log out. The evaluator analyzed the packet capture and found only the mentioned network traffic from the TOE process.

Test 2 – The evaluator performed a port scan and verified that the TOE does not open any port for listening.

2.3 Identification and Authentication (FIA)

2.3.1 X.509 Certificate Validation (ASPP14:FIA_X509_EXT.1)

2.3.1.1 ASPP14:FIA_X509_EXT.1.1

TSS Assurance Activities: 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.

Section 6.3 of the ST states the TOE performs certificate validity checking within its Bouncy Castle cryptographic library.

Additionally, Section 6.3 describes the certificate path validation algorithm as follows:

The TOE examines each certificate in the path (starting with the peer's certificate) and first checks for validity of that certificate (e.g., has the certificate expired; or if not yet valid, whether the certificate contains the appropriate X.509 extensions [e.g., the CA flag in the basic constraints extension for a CA certificate, or that a server certificate contains the Server Authentication purpose in the ExtendedKeyUsagefield]), then verifies each certificate in the chain (applying the same rules as above, but also ensuring that the Issuer of each certificate matches the Subject in the next rung "up" in the chain and that the chain ends in a self-signed certificate present in either the TOE'S trusted anchor database or matches a specified Root CA), and finally the TOE performs revocation checking for all certificates in the chain. The TOE supports CRL as specified for RFC 5280 Section 6.3 for RSA certificates and CRL as specified in RFC 8603 for ECDSA certificates used in revocation checks under TLS connections.

Guidance Assurance Activities: None Defined

Testing Assurance Activities: 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, for each of the following reasons, in turn:

- by establishing a certificate path in which one of the issuing certificates is not a CA certificate,

- by omitting the basicConstraints field in one of the issuing certificates,

- by setting the basicConstraints field in an issuing certificate to have CA=False,

- by omitting the CA signing bit of the key usage field in an issuing certificate, and

- by setting the path length field of a valid CA field to a value strictly less than the certificate path.

The evaluator shall then establish a valid certificate path consisting of valid CA certificates, and demonstrate that the function succeeds. The evaluator shall then remove trust in one of the CA 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 â€“ conditional on whether CRL, OCSP, OCSP Stapling, or OCSP Multi-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. If OCSP stapling per RFC6066 is the only supported revocation method, this test is omitted.

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.

Test 4: f any OCSP option is selected, the evaluator shall ensure the TSF has no other source of revocation information available and configure the OCSP server or use a man-in-the-middle tool to present an OCSP response signed by a certificate that does not have the OCSP signing purpose and which is the only source of revocation status information advertised by the CA issuing the certificate being validated. The evaluator shall verify that validation of the OCSP response fails and that the TOE treats the certificate being checked as invalid and rejects the connection. If CRL is selected, the evaluator shall likewise configure the CA to be the only source of revocation status information, and sign a CRL with a certificate that does not have the cRLsign key usage bit set, and . The evaluator shall verify that validation of the CRL fails and that the TOE treats the certificate being checked as invalid and rejects the connection.

Note: The intent of this test is to ensure a TSF does not trust invalid revocation status information. A TSF receiving invalid revocation status information from the only advertised certificate status provider should treat the certificate whose status is being checked as invalid. This should generally be treated differently from the case where the TSF is not able to establish a connection to check revocation status information, but it is acceptable that the TSF ignore any invalid information and attempt to find another source of revocation status (another advertised provider, a locally configured provider, or cached information) and treat this situation as not having a connection to a valid certificate status provider. (TD0669 applied)

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.)

Test 8: (Conditional on support for EC certificates as indicated in FCS_COP.1/Sig). The evaluator shall establish a valid, trusted certificate chain consisting of an EC leaf certificate, an EC Intermediate CA certificate not designated as a trust anchor, and an EC certificate designated as a trusted anchor, where the elliptic curve parameters are specified as a named curve. The evaluator shall confirm that the TOE validates the certificate chain.

Test 9: (Conditional on support for EC certificates as indicated in FCS_COP.1/Sig). The evaluator shall replace the intermediate certificate in the certificate chain for Test 8 with a modified certificate, where the modified intermediate CA has a public key information field where the EC parameters uses an explicit format version of the Elliptic Curve parameters in the public key information field of the intermediate CA certificate from Test 8, and the modified Intermediate CA certificate is signed by the trusted EC root CA, but having no other changes. The evaluator shall confirm the TOE treats the certificate as invalid.

Test 1 – The evaluator configured a server to use TLS with credential with a valid certification path. The evaluator connected the TOE to the server and the connection was accepted. The evaluator tested the case where a trusted CA needed to validate the server certificate was removed from the TOE's trusted certificate store under the test FCS_TLSC_EXT.1.3-t1-Part 3. In the referenced test, the evaluator found that the connection was refused. The evaluator attempted to connect the TOE to a TLS server that was using a server certificate whose issuing CA lacked the basicConstraints extension and the connection was refused. The evaluator attempted to connect the TOE to a TLS server that was using a server certificate whose issuing CA had the basicConstraints field to state CA=False and the connection was refused. The evaluator attempted to connect the TOE to a TLS server whose issuing CA was using a server certificate that had an incorrect path length under basicConstraints extension and the connection was refused. The evaluator attempted to connect to a TLS server that was using a server certificate whose issuing CA omitted the CA keyCertSign bit under the key usage field and the connection was refused.

Test 2 – The evaluator attempted to connect the TOE to a TLS server with a valid, unexpired certificate and demonstrate that the TOE could properly validate this certificate. Then the evaluator attempted to connect to a TLS server using a leaf-node server certificate that was expired and found the connection was refused. The evaluator then tried to connect to a TLS server using a server certificate that was issued by a CA that was expired and found that this was also refused.

Test 3 – The evaluator configured the TOE to connect to a TLS server using certificates with accurate CRL information. The evaluator attempted to connect to the server using a server certificate that was not revoked and saw that this connection was accepted. The evaluator attempted to connect to a server using a revoked server certificate and saw that this connection was rejected. The evaluator then attempted to connect to the server using a server certificate that was issued by a revoked intermediate CA and saw that this connection was also rejected.

Test 4 – The evaluator configured the TOE to connect to a TLS server using certificates with accurate CRL information. The evaluator then attempted to connect to the server using a server certificate that had its revocation information signed by a CA that lacked the CRL Signing key usage and saw that this connection was rejected. The evaluator then attempted to connect to the server using a server certificate that had its issuing CA's revocation information signed by a certificate that lacked the CRL Signing key usage and saw that this connection was also rejected.

Test 5- The evaluator configured the TOE to use TLS. The evaluator modified a byte in the first eight bytes of the client certificate. A handshake error was received.

Test 6- The evaluator configured the TOE to use TLS. The evaluator modified a byte in the last byte of the client certificate. A decryption error message was received.

Test 7 - The evaluator configured the TOE to use TLS. The evaluator modified the public key of the client certificate. An invalid signature error message was received.

Test 8 - The evaluator configured the TOE to use TLS. The evaluator attempted to connect to a TLS server with a valid EC certificate chain where the elliptic curve parameters are specified as a named curve. The evaluator observed that the TLS connection was successful.

Test 9 - The evaluator configured the TOE to use TLS. The evaluator attempted to connect to a TLS server with a valid EC certificate chain where the elliptic curve parameters in the intermediate CA are specified as an explicit curve. The evaluator observed that the TLS connection was rejected.

2.3.1.2 ASPP14:FIA_X509_EXT.1.2

TSS Assurance Activities: None Defined

Guidance Assurance Activities: None Defined

Testing Assurance Activities: 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 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.

Test 1 – The evaluator tested this in conjunction with FIA_X509_EXT.1.1-t1. In that test, the evaluator attempted to connect the TOE to a server using a certificate without a basicConstraints extension and found that the validation of the certificate path fails as part of the validation of the peer certificate belonging to this chain.

Test 2 – The evaluator tested this in conjunction with FIA_X509_EXT.1.1-t1. In that test, the evaluator attempted to connect the TOE to a server using a certificate with the CA flag set to false in the basicConstraints extension and found that the validation of the certificate path fails as part of the validation of the peer certificate belonging to this chain.

Component TSS Assurance Activities: None Defined

Component Guidance Assurance Activities: None Defined

Component Testing Assurance Activities: None Defined

2.3.2 X.509 Certificate Authentication (ASPP14:FIA_X509_EXT.2)

2.3.2.1 ASPP14:FIA_X509_EXT.2.1

TSS Assurance Activities: None Defined

Guidance Assurance Activities: None Defined

Testing Assurance Activities: None Defined

2.3.2.2 ASPP14:FIA_X509_EXT.2.2

TSS Assurance Activities: None Defined

Guidance Assurance Activities: None Defined

Testing Assurance Activities: None Defined

Component TSS Assurance Activities: 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.

Section 6.3 of the ST states that the TOE validates the entire provided certificate chain and the TOE validates X.509 certificates through outbound TLS connections against the administrator configured trusted anchor database. The AGD contains instructions followed by the evaluation team to configured the referenced trusted anchor database.

Additionally, Section 6.3 states that TLS server certificates that cannot be validated will not be accepted by the TOE and the trusted channel will not be established.

Component Guidance Assurance Activities: None Defined

Component Testing Assurance Activities: 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.

Test 1 – The evaluator configured a client to use TLS with a client credential. The evaluator configured the CDP information for the necessary certificates used in this test so that the TOE would be able to fetch the CRLs and verify the client credential. The evaluator then attempted to connect to the TOE and showed that the connection was accepted. The evaluator then removed the CDP information and flushed the CRL cache so that the TOE no longer had access to the revocation information. The evaluator attempted to connect to the TOE again, this time showing that the connection failed.

Test 2 – the evaluator referred to ASPP12:FIA_X509_EXT.1.1-t3 where the evaluator tested this exact test. In this test, the evaluator tested that a certificate that was revoked by an external server who hosted a CRL from the certificate's issuing CA could not be accepted.

2.4 Security management (FMT)

2.4.1 Secure by Default Configuration (ASPP14:FMT_CFG_EXT.1)

2.4.1.1 ASPP14:FMT_CFG_EXT.1.1

TSS Assurance Activities: The evaluator shall check the TSS to determine if the application requires any type of credentials and if the application installs with default credentials.

Section 6.4 of the TSS states the TOE requires no credential of its own. As a result, the TOE does not install with default credentials.

The evaluation evidence makes reference to a username and password that belongs to an external TNMS server, so Section 6.4 further clarifies this by stating the TOE accepts a username and password, and the TOE forwards these to the TNMS server (through a TLS protected connection and after first hashing the user's password with SHA-1). This credential belongs to and is managed by the external server and is not claimed as a credential for the TOE under this requirement.

Guidance Assurance Activities: None Defined

Testing Assurance Activities: 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.

Test 1 – Not applicable, the TOE does not require any credentials of its own.

Test 2 – Not applicable, the TOE does not have any credentials.

Test 3 – Not applicable, the TOE does not have default credentials.

2.4.1.2 ASPP14:FMT_CFG_EXT.1.2

TSS Assurance Activities: None Defined

Guidance Assurance Activities: None Defined

Testing Assurance Activities: 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.

Platforms: Android....

The evaluator shall run the command find -L . -perm /002 inside the application's data directories to ensure that all files are not world-writable. The command should not print any files.

Platforms: Microsoft 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.

Platforms: Apple iOS....

The evaluator shall determine whether the application leverages the appropriate Data Protection Class for each data file stored locally.

Platforms: Linux....

The evaluator shall run the command find -L. -perm /002 inside the application's data directories to ensure that all files are not world-writable. The command should not print any files.

Platforms: Oracle 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.

Platforms: Apple 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 evaluator installed and ran the application. The evaluator then ran the Windows command to inspect the filesystem for permissions created by the application and the command returned no files.

Component TSS Assurance Activities: None Defined

Component Guidance Assurance Activities: None Defined

Component Testing Assurance Activities: None Defined

2.4.2 Supported Configuration Mechanism - per TD0624 (ASPP14:FMT_MEC_EXT.1)

2.4.2.1 ASPP14:FMT_MEC_EXT.1.1

TSS Assurance Activities: None Defined

Guidance Assurance Activities: None Defined

Testing Assurance Activities: None Defined

Component TSS Assurance Activities: 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.

Section 6.4 of the ST states the TOE makes use of its application storage area within the Windows filesystem to store the configured root CA. This is consistent with the claim to invoke mechanisms recommended by the platform vendor.

Implement functionality to encrypt and store configuration options is not claimed under this requirement.

Component Guidance Assurance Activities: None Defined

Component Testing Assurance Activities: 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:

Platforms: Android....

The evaluator shall run the application and make security-related changes to its configuration. The evaluator shall check that at least one file exists at location /data/data/package/shared_prefs/ (for SharedPreferences) and/or /data/data/package/files/datastore (for DataStore), where the package is the Java package of the application. For SharedPreferences the evaluator shall examine the XML file to make sure it reflects the changes made to the configuration to verify that the application used SharedPreferences and/or PreferenceActivity to store the configuration data. For DataStore the evaluator shall use a protocol buffer analyzer to examine the file to make sure it reflects the changes made to the configuration to verify that the application used DataStore to store the configuration data.

Platforms: Microsoft Windows....

The evaluator shall determine and verify that Windows Universal Applications use either the Windows.Storage namespace, 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 Windows Registry or C:\ProgramData\ directory.

Platforms: Apple iOS....

The evaluator shall verify that the app uses the user defaults system or key-value store for storing all settings.

Platforms: 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).

Platforms: Oracle 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).

Platforms: Apple 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 TOE invokes platform mechanisms for storing credentials. The evaluator ran ProcMon while configuring the trust store used for TLS authentication. The ProcMon scan confirmed that the public certificate was housed in the truststore which was configured according to the AGD to be under C:\ProgramData\.

2.4.3 Specification of Management Functions (ASPP14:FMT_SMF.1)

2.4.3.1 ASPP14:FMT_SMF.1.1

TSS Assurance Activities: None Defined

Guidance Assurance Activities: None Defined

Testing Assurance Activities: None Defined

Component TSS Assurance Activities: None Defined

Component Guidance Assurance Activities: 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 TOE claims the following management functions and the evaluator ensured that the AGD contained references to the information required to perform the function

Specify the TNMS Server (by FQDN or IP address):

Section 3.9 TNMS Login contains the information to connect to the external TNMS server. More specifically, the section identifies that the TNMS server address can be specified as the <server IP address> or <FQDN> format.

Manage trusted root CA certificates:

Section 3.5 Managing Root Certificates contains information about how to use the JRE keytool command to manage the provided trusted root store. This section also includes subsections to directly address how new certificates are installed, current trusted certificates can be queried, and existing trusted certificates can be removed.

Connect with to the TNMS Server:

Section 3.9 TNMS Login contains the information to connect to the external TNMS server and *Section 3.10 TNMS Logout* contains information to terminate the connection. The TOE is a UI client for the external TNMS server, so additional actions performed via the connection are not performed by the TOE, but rather the external TNMS server, and as such are not included under this evaluation.

Component Testing Assurance Activities: 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 TOE claims three management functions under this SFR.

Part 1 Specify the TNMS server (by FQDN or IP address) – The evaluator attempted to log on to an external TNMS server using a FQDN and an IP address and showed both connections were successful. Further testing of this same mechanism against ensuring that the configured identifier is used is performed under FCS_TLSC_EXT.1.2.

Part 2 Manage trusted root CA certificates – The evaluator attempted to connect to an external TNMS server using a server certificate with a non-configured CA and showed that the TOE could not connect. The evaluator then configured the same CA and repeated the connection and showed the TOE could now long in, demonstrating the configuration was a success. The evaluator then removed the certificate and showed the log in once again failed. Throughout the process, the evaluator queried the truststore and found the reported certificates matched the expected TLS behavior

Part 3 Connect with to the TNMS server – Testing was down in conjunction with the previous two management functions as the evaluator demonstrated the TOE could connect with an external TNMS server when specified with matching identifiers and trusted roots.

2.5 Privacy (FPR)

2.5.1 User Consent for Transmission of Personally Identifiable (ASPP14:FPR_ANO_EXT.1)

2.5.1.1 ASPP14:FPR_ANO_EXT.1.1

TSS Assurance Activities: None Defined

Guidance Assurance Activities: None Defined

Testing Assurance Activities: None Defined

Component TSS Assurance Activities: The evaluator shall inspect the TSS documentation to identify functionality in the application where PII can be transmitted.

Section 6.5 states the TOE does not transmit any PII.

Component Guidance Assurance Activities: None Defined

Component Testing Assurance Activities: 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.

Not applicable, the TOE does not transmit PII.

2.6 Protection of the TSF (FPT)

2.6.1 Anti-Exploitation Capabilities (ASPP14:FPT_AEX_EXT.1)

2.6.1.1 ASPP14:FPT_AEX_EXT.1.1

TSS Assurance Activities: The evaluator shall ensure that the TSS describes the compiler flags used to enable ASLR when the application is compiled.

The TOE is primarily a Java application that does not rely on compilation flags for most runtime protections including ASLR.

For native executables included with the TOE, section 6.6 of the ST states the corresponding compilation flags are used including /DYNAMICBASE, /GS, and /NXCOMPAT.

Guidance Assurance Activities: None Defined

Testing Assurance Activities: 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 those platforms requiring the same application running on two different systems, the evaluator may alternatively 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.

Platforms: 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.

Platforms: Microsoft 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.

Platforms: Apple 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.

Platforms: 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.

Platforms: Oracle 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.

Platforms: Apple 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 TOE is a java application that does not rely on the same ASLR protections to produce randomized memory addresses, however the evaluator still ran the Windows requirements above to show the platform-level protections were enabled. The evaluator ran the same application on two different Windows instances and gathered memory maps using the Microsoft Sysinternals tool VMMap. The evaluator compared the two memory maps and could not find any instance of an explicit memory address allocated to the same address. The evaluator also used Microsoft's BinScope Binary Analyzer to search through all of the binaries on the TOE and found that no binaries failed the DBCheck for /DYNAMICBASE, showing that ASLR was enabled. The evaluator finally ran the TOE and used Process Hacker to analyze the mitigation policies the application loads by default and observed platform-level ASLR was enabled.

2.6.1.2 ASPP14:FPT_AEX_EXT.1.2

TSS Assurance Activities: None Defined

Guidance Assurance Activities: None Defined

Testing Assurance Activities: The evaluator shall verify that no memory mapping requests are made with write and execute permissions. The method of doing so varies per platform.

Platforms: Android....

The evaluator shall perform static analysis on the application to verify that

o mmap is never invoked with both the PROT_WRITE and PROT_EXEC permissions, and

o mprotect is never invoked.

Platforms: Microsoft 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.

Platforms: Apple iOS....

The evaluator shall perform static analysis on the application to verify that mprotect is never invoked with the PROT_EXEC permission.

Platforms: Linux....

The evaluator shall perform static analysis on the application to verify that both

- o mmap is never be invoked with both the PROT_WRITE and PROT_EXEC permissions, and
- o mprotect is never invoked with the PROT_EXEC permission.

Platforms: Oracle Solaris....

The evaluator shall perform static analysis on the application to verify that both

- o mmap is never be invoked with both the PROT_WRITE and PROT_EXEC permissions, and
- o mprotect is never invoked with the PROT_EXEC permission.

Platforms: Apple macOS....

The evaluator shall perform static analysis on the application to verify that mprotect is never invoked with the PROT_EXEC permission.

The TOE is primarily a Java application that does not rely on any data execution protection being enabled as Java manages its own memory and does not have a method of opting out of its runtime exception checking.

For native executables bundled with this app, this was tested alongside ASPP14:FPT_AEX_EXT.1.1-t1 where the evaluator obtained a list of all TOE executables that could be processed with BinScope and ran them through a variety of different checks. During this test, the evaluator found that the /NXCOMPAT flag was used during compilation.

2.6.1.3 ASPP14:FPT_AEX_EXT.1.3

TSS Assurance Activities: None Defined

Guidance Assurance Activities: None Defined

Testing Assurance Activities: The evaluator shall configure the platform in the ascribed manner and carry out one of the prescribed tests:

Platforms: Android....

Applications running on Android cannot disable Android security features, therefore this requirement is met and no evaluation activity is required.

Platforms: Microsoft 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/threatprotection/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).

Platforms: Apple iOS....

Applications running on iOS cannot disable security features, therefore this requirement is met and no evaluation activity is required.

Platforms: Linux....

The evaluator shall ensure that the application can successfully run on a system with either SELinux or AppArmor enabled and in enforce mode.

Platforms: Oracle Solaris....

The evaluator shall ensure that the application can run with Solaris Trusted Extensions enabled and enforcing.

Platforms: Apple macOS....

The evaluator shall ensure that the application can successfully run on macOS without disabling any security features.

Windows 10 no longer supports Windows Enhanced Mitigation Experience Toolkit in favor of Windows Defender. The evaluator ran the TOE on a platform that had Windows Defender Exploit Guard configured and enabled with Control Flow Guard, Randomize Memory allocations, Export address filtering, Import address filtering, and Data Execution Prevention and found no issues with TOE functionality.

2.6.1.4 ASPP14:FPT_AEX_EXT.1.4

TSS Assurance Activities: None Defined

Guidance Assurance Activities: None Defined

Testing Assurance Activities: The evaluator shall run the application and determine where it writes its files. For files where the user does not choose the destination, the evaluator shall check whether the destination directory contains executable files. This varies per platform:

Platforms: 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.

Platforms: Microsoft 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.

Platforms: Apple iOS....

The evaluator shall consider the requirement met because the platform forces applications to write all data within the application working directory (sandbox).

Platforms: 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.

Platforms: Oracle 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.

Platforms: Apple 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 evaluator used the Windows SysInternals tool AccessCheck with the '-w users' flag set to search through the directories used by the application, including the directories which housed executables, searching for user-writeable files. The evaluator found no user-writeable files in any of the directories.

2.6.1.5 ASPP14:FPT_AEX_EXT.1.5

TSS Assurance Activities: None Defined

Guidance Assurance Activities: None Defined

Testing Assurance Activities: The evaluator will inspect every native executable included in the TOE to ensure that stack-based buffer overflow protection is present.

Platforms: Microsoft 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 evaluator ensured that section 6.6 of the ST specified that the correct /GS flag was used during compilation. The evaluator then surveyed the directories used by the TOE and found a series of PE native executables. The evaluator copied these to a Linux system in order to run the recommended Canary Detector tool and found that all of the reported files were found that all of them were compiled with the applicable sequences for stack canaries.

Component TSS Assurance Activities: None Defined

Component Guidance Assurance Activities: None Defined

Component Testing Assurance Activities: None Defined

2.6.2 Use of Supported Services and APIs (ASPP14:FPT_API_EXT.1)

2.6.2.1 ASPP14:FPT_API_EXT.1.1

TSS Assurance Activities: None Defined

Guidance Assurance Activities: None Defined

Testing Assurance Activities: None Defined

Component TSS Assurance Activities: The evaluator shall verify that the TSS lists the platform APIs used in the application.

Section 6.6 of the ST contains a list of various APIs used in the TOE application.

Component Guidance Assurance Activities: None Defined

Component Testing Assurance Activities: 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.

The evaluator pulled this of supported APIs used by the TOE from the ST. The evaluator compared the list with online documentation for the Open Java Development Kit (OpenJDK) and found references to all of the included APIs.

2.6.3 Software Identification and Versions (ASPP14:FPT_IDV_EXT.1)

2.6.3.1 ASPP14:FPT_IDV_EXT.1.1

TSS Assurance Activities: None Defined

Guidance Assurance Activities: None Defined

Testing Assurance Activities: None Defined

Component TSS Assurance Activities: If 'other version information' is selected the evaluator shall verify that the TSS contains an explanation of the versioning methodology.

Section 6.6 of the TSS states that the TOE is versioned with a major and minor version number, as well as a build number that is incremented with every update to the TOE.

Component Guidance Assurance Activities: None Defined

Component Testing Assurance Activities: 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.

This is tested in conjunction with FPT_TUD_EXT.1-t1 where the evaluator checked the current version as well as for any available updates. During this test, the evaluator observed the version of the TOE.

2.6.4 Use of Third Party Libraries (ASPP14:FPT_LIB_EXT.1)

2.6.4.1 ASPP14:FPT_LIB_EXT.1.1

TSS Assurance Activities: None Defined

Guidance Assurance Activities: None Defined

Testing Assurance Activities: None Defined

Component TSS Assurance Activities: None Defined

Component Guidance Assurance Activities: None Defined

Component Testing Assurance Activities: 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 evaluator inventories the location of where the application was installed for files with extensions that matched dynamic libraries including .so, .a, .dll, .la, .jar, and .java. The resulting files were then individually identified to their corresponding library in the ST and ensured that no additional libraries were included.

2.6.5 Integrity for Installation and Update (ASPP14:FPT_TUD_EXT.1)

2.6.5.1 ASPP14:FPT_TUD_EXT.1.1

TSS Assurance Activities: None Defined

Guidance Assurance Activities: The evaluator shall check to ensure the guidance includes a description of how updates are performed.

Section 3.8 Updating TNMS Client of the AGD contains details of how updates should be performed.

Testing Assurance Activities: 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 evaluator followed the procedures in section 3.6 the AGD to check for an update. The update script reported back the current version, the latest version, and whether the client was considered up to date. Since the client was up to date and no error was issued, this requirement was considered to be met.

2.6.5.2 ASPP14:FPT_TUD_EXT.1.2

TSS Assurance Activities: None Defined

Guidance Assurance Activities: The evaluator shall verify guidance includes a description of how to query the current version of the application.

Section 3.6 Checking the Installed Version contains information about how the end-user can check if there are any updates available. The same procedure for checking for updates also will report back the current version of the product and if it is up to date.

Testing Assurance Activities: 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 evaluator tested this in conjunction with FPT_TUD_EXT.1.1-t1. The evaluator verified the installed and documented versions are the same.

2.6.5.3 ASPP14:FPT_TUD_EXT.1.3

TSS Assurance Activities: None Defined

Guidance Assurance Activities: None Defined

Testing Assurance Activities: The evaluator shall verify that the application's executable files are not changed by the application.

Platforms: Apple 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 perform 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 evaluator installed the application and recorded the timestamps and MD5 hash of all the executables bundled with the TOE. The evaluator then ran the application and ensured functionality. The evaluator then compared the current executable attributes with the previous record and found that they were the same

2.6.5.4 ASPP14:FPT_TUD_EXT.1.4

TSS Assurance Activities: The evaluator shall verify that the TSS identifies how updates to the application 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.

Section 6.6 of the ST contains information regarding TOE updates. As stated in this section, “The vendor packages updates to the TOE in an EXE format (as opposed to being bundled with the Windows platform itself) and relies upon the Windows 10 operating system to verify the installation package’s signature before installing. Updates are signed by Infinera’s Window developer key and display under the Windows platform with the Name of Signer as www.infinera.com.”

The same section also identifies the following for how candidate updates are obtained: “Updates are obtained through Infinera’s Customer Service Portal (https://support.infinera.com/) and are installed using the same process as the initial installations.”

Guidance Assurance Activities: None Defined

Testing Assurance Activities: None Defined

2.6.5.5 ASPP14:FPT_TUD_EXT.1.5

TSS Assurance Activities: The evaluator shall verify that the TSS identifies how the application is distributed.

Section 6.6 of the ST states the vendor packages updates to the TOE in an EXE installer format which details that format and source of application updates. Updates are in the same format as initial initializations as updates are processed through an uninstall and reinstall of the TOE application.

Guidance Assurance Activities: None Defined

Testing Assurance Activities: 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 TOE claims ‘as an additional package’ and therefore FPT_TUD_EXT.2 is included and no additional testing activity is performed.

Component TSS Assurance Activities: None Defined

Component Guidance Assurance Activities: None Defined

Component Testing Assurance Activities: None Defined

2.6.6 Integrity for Installation and Update - per TD0664 (ASPP14:FPT_TUD_EXT.2)

2.6.6.1 ASPP14:FPT_TUD_EXT.2.1

TSS Assurance Activities: None Defined

Guidance Assurance Activities: None Defined

Testing Assurance Activities: The evaluator shall verify that application updates are distributed in the format supported by the platform. This varies per platform:

Platforms: Android....

The evaluator shall ensure that the application is packaged in the Android application package (APK) format.

Platforms: Microsoft 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/enus/library/ms537364(v=vs.85).aspx for details regarding Authenticode signing.

Platforms: Apple iOS....

The evaluator shall ensure that the application is packaged in the IPA format.

Platforms: 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.

Platforms: Oracle Solaris....

The evaluator shall ensure that the application is packaged in the PKG format.

Platforms: Apple macOS....

The evaluator shall ensure that application is packaged in the DMG format, the PKG format, or the MPKG format.

The TOE was provided to the evaluator in the form of an .EXE executable file for all of testing. The evaluator verified the signature by using the platform to view the signature details.

2.6.6.2 ASPP14:FPT_TUD_EXT.2.2

TSS Assurance Activities: None Defined

Guidance Assurance Activities: None Defined

Testing Assurance Activities: Platforms: Android....

The evaluator shall consider the requirement met because the platform forces applications to write all data within the application working directory (sandbox).

Platforms: Apple iOS....

The evaluator shall consider the requirement met because the platform forces applications to write all data within the application working directory (sandbox).

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 evaluator recorded every file on the filesystem prior to the installation of the application, and then installed and ran the application. The evaluator then followed AGD guidance to uninstall the TOE and compared the resulting filesystem to the initial record. The only differences in the file scans were related to things not attributable to the TOE or intentionally left behind as it was considered configuration files which are allowed per this testing requirement.

2.6.6.3 ASPP14:FPT_TUD_EXT.2.3

TSS Assurance Activities: The evaluator shall verify that the TSS identifies how the application installation package is signed by an authorized source. The definition of an authorized source must be contained in the TSS.

Section 6.6 of the ST contains the following about how the TOE updates are signed by an authorized source: The vendor packages updates to the TOE in an EXE installer format (as opposed to being bundled with the Windows platform itself) and relies upon the Windows 10 operating system to verify the installation package's signature before installing. Updates are signed by Infinera's Window developer key and display under the Windows platform with the Name of Signer as www.infinera.com. Updates are in the same format as initial initializations as updates are processed through an uninstall and reinstall of the TOE application.

Guidance Assurance Activities: None Defined

Testing Assurance Activities: None Defined

Component TSS Assurance Activities: None Defined

Component Guidance Assurance Activities: None Defined

Component Testing Assurance Activities: None Defined

2.7 Trusted path/channels (FTP)

2.7.1 Protection of Data in Transit - per TD0655 (ASPP14:FTP_DIT_EXT.1)

2.7.1.1 ASPP14:FTP_DIT_EXT.1.1

TSS Assurance Activities: None Defined

Guidance Assurance Activities: None Defined

Testing Assurance Activities: None Defined

Component TSS Assurance Activities: For platform-provided functionality, the evaluator shall verify the TSS contains the calls to the platform that TOE is leveraging to invoke the functionality.

Not applicable, the TOE does not invoke platform-provided functionality.

Component Guidance Assurance Activities: None Defined

Component Testing Assurance Activities: 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, 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.

Platforms: 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.

Platforms: Apple 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.

Test 1 – The evaluator reanalyzed the packet capture from FDP_NET_EXT.1-t1. The TOE claims to encrypt all sensitive data. All data transmitted by the TOE is either encrypted with TLS (for communicating with the external

TNMS server) or is not considered sensitive data (as is the case with version information to the vendor's public versioning server). The evaluator did not observe any undocumented traffic form the TOE.

Test 2 – The evaluator reanalyzed the packet capture from FDP_NET_EXT.1-t1. The evaluator attempted to search for the credentials used by the external TNMS server in plaintext form and could not find them. The evaluator found no other sensitive data in packet capture

Test 3 – The TOE does not have any credentials of its own, however the evaluator tested the TNMS server's credentials in Test 2. The evaluator found no evidence of any credentials in the network traffic.

3. Protection Profile SAR Assurance Activities

The following sections address assurance activities specifically defined in the ASPP14/PKGTLS11 that correspond with Security Assurance Requirements.

3.1 Development (ADV)

3.1.1 Basic Functional Specification (ADV_FSP.1)

Assurance Activities: There are no specific assurance activities associated with these SARs, except ensuring the information is provided. The functional specification documentation is provided to support the evaluation activities described in Section 5.1, and other activities described for AGD, ATE, and AVA SARs. The requirements on the content of the functional specification information is implicitly assessed by virtue of the other assurance activities being performed; if the evaluator is unable to perform an activity because there is insufficient interface information, then an adequate functional specification has not been provided.

The assurance activities from section 5.1 of ASPP14 have been performed and the analysis of the evaluator is documented in the previous sections of this document.

3.2 Guidance documents (AGD)

3.2.1 Operational User Guidance (AGD_OPE.1)

Assurance Activities: Some of the contents of the operational guidance will be verified by the assurance activities in Section 5.1 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 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 hash/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.

Section 3.4 Enabling FIPS mode of the AGD indicates that the TOE installation requires setting up the cryptographic library into FIPS mode in order to be in compliance with the Common Criteria evaluated configuration. Additionally, this section also states that “the use of any other cryptographic engines, or configurations other than what is described in this section was not evaluated nor tested during the Common Criteria Evaluation of TNMS Client

Section 3.8 Updating TNMS Client contains all required information including where to obtain new updates, instructions to update the TOE, verifying signatures, and determining if it was successful. Updates are obtained from Infinera's Customer Service portal. This section states that updates are installed in the same manner as the initial installation which is described in section 3.2 TNMS Client Installation. In that referenced section, any necessary instructions for making the installation accessible to the platform are included. Similarly, the success of the update can be checked in the same manner as the initial update, by "run[ning] the client and connect[ing] and login to a TNMS server as described in the next sections [*Section 3.9 TNMS Login*]" The signature is verified by the Windows platform and the end user can ensure that the "Name of signer" on the platform signature matches "www.infinera.com"

The TOE does not contain any obvious functionality that is not included under the scope of this evaluation aside from any mention of the external TNMS server. *Section 1 Introduction* states "the full TNMS system consists of a TNMS server and a TNMS client. This guidance is provided as a supplement to the TNMS Customer Documentation provided with every TNMS installation and describes how to install and configure the TNMS Client Component as the evaluated configuration compliant with the Common Criteria for Information Technology Security Evaluation version 3.1." As detailed, this document and evaluation only cover the TNMS Client as the TNMS server is external to the TOE and a product covered in a separate on-going evaluation.

3.2.2 Preparative Procedures (AGD_PRE.1)

Assurance Activities: 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 ST claims that TOE is capable of running on a Microsoft Windows 10 (64-bit) platform with an Amazon Corretto JDK/JRE version 11.0.6 and Oracle Java JRE 8u201. The AGD is in agreement with this claim as the list of supported OS's listed in *Section 2.1 Supported Operating Systems* and the list of software prerequisites in *Section 3.1 Software Pre-Requisites* identify that the TOE runs on Windows 10 (64 bit) with Amazon Corretto JDK 11.0.6.10.1 and Oracle Java JRE 8u201.

3.3 Life-cycle support (ALC)

3.3.1 Labelling of the TOE (ALC_CMC.1)

Assurance Activities: 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 title of the AGD document specifies a specific version of the TNMS Client and the evaluator ensure that was consistent with the samples received during testing and the version contained in the ST.

3.3.2 TOE CM Coverage (ALC_CMS.1)

Assurance Activities: 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 assurance activity for ALC_CMC.1), the evaluator implicitly confirms the information required by this component. Life-cycle support is targeted aspects of the developer's life-cycle and instructions to providers of applications for the developer's devices, rather than an in-depth examination of the TSF manufacturer's development and configuration management process. This is not meant to diminish the critical role that a developer's practices play in contributing to the overall trustworthiness of a product; rather, it's a reflection on the information to be made available for evaluation.

The evaluator 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 evaluator noted that the AGD identifies that the subject of the document is for the matching version in the ST. *Section 2 System Requirements* of the AGD goes over the different development environments including the hardware and software claims. Included in the subsection *Section 2.1 Supported Operating Systems,* the AGD states "TNMS Client has been evaluated in in a Microsoft Windows 10 (64 bit) environment. The TNMS client is primarily a Java application and features some native level libraries. For all components, the TNMS client is compiled with all necessary compilation flags to ensure that all required environmental protections are enabled by default and require no further configuration under Windows."

3.3.3 Timely Security Updates (ALC_TSU_EXT.1)

Assurance Activities: 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.

Section 6.6 of the ST claims the vendor provides timely security updates for the TOE and third party libraries included in the TOE. The vendor aims for updates as soon as possible with a maximum of 30 days. Updates are packaged in the same RPM format as the original installation of the TOE. The vendor maintains a customer portal where updates can be found and new issues can be reported.

3.4 Tests (ATE)

3.4.1 Independent Testing - Conformance (ATE_IND.1)

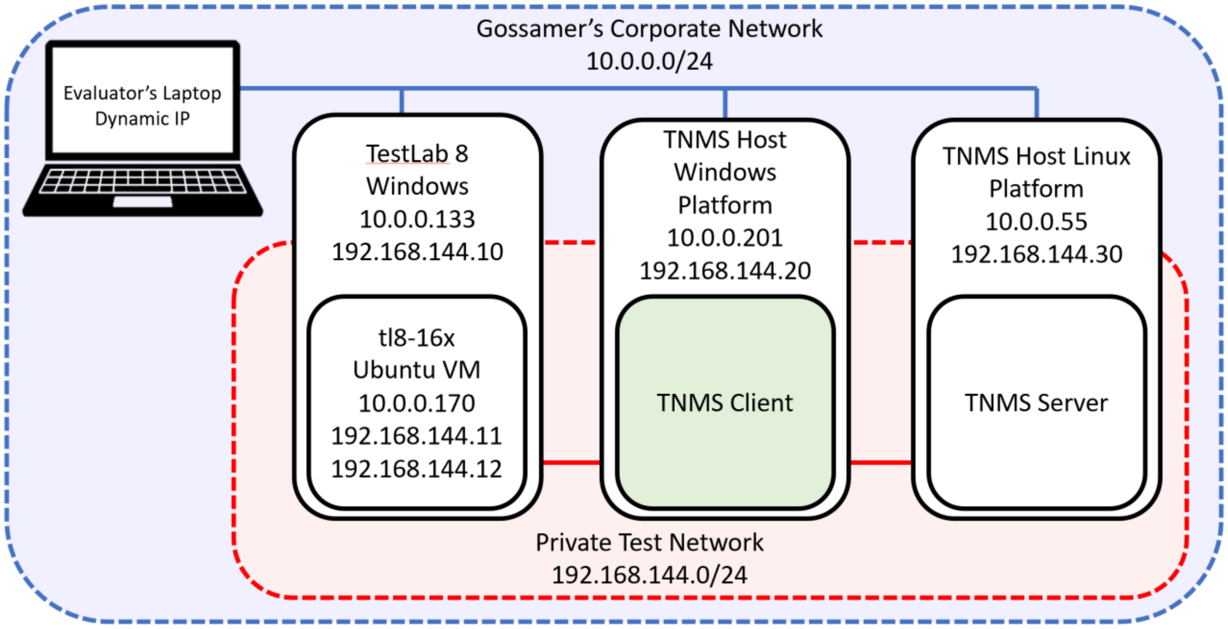
Assurance Activities: The evaluator shall prepare a test plan and report documenting the testing aspects of the system, including any application crashes during testing. The evaluator shall determine the root cause of any application crashes and include that information in the report. The test plan covers all of the testing actions contained in the [CEM] and the body of this PP's Assurance Activities.

While it is not necessary to have one test case per test listed in an Assurance Activity, the evaluator must document in the test plan that each applicable testing requirement in the ST is covered. The test plan identifies the platforms to be tested, and for those platforms not included in the test plan but included in the ST, the test plan provides a justification for not testing the platforms. This justification must address the differences between the tested platforms and the untested platforms, and make an argument that the differences do not affect the testing to be performed. It is not sufficient to merely assert that the differences have no affect; rationale must be provided. If all platforms claimed in the ST are tested, then no rationale is necessary. The test plan describes the composition of each platform to be tested, and any setup that is necessary beyond what is contained in the AGD documentation. It should be noted that the evaluator is expected to follow the AGD documentation for installation and setup of each platform either as part of a test or as a standard pre-test condition. This may include special test drivers or tools. For each driver or tool, an argument (not just an assertion) should be provided that the driver or tool will not adversely affect the performance of the functionality by the 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 (IPsec, TLS, 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 evaluator created a proprietary Detailed Test Report (DTR) to address all aspects of this requirement. The DTR discusses the test configuration, test cases, expected results, and test results. The following diagram depicts the test environment



3.5 Vulnerability assessment (AVA)

3.5.1 Vulnerability Survey (AVA_VAN.1)

Assurance Activities: 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.

The virus definition search and vulnerability analysis are in the proprietary Detailed Test Report (DTR) prepared by the evaluator. The evaluator ran a Windows Defender virus scan with current virus definitions against all of the TOE components and verified that no files are flagged as malicious. The vulnerability analysis includes a public search for vulnerabilities. The evaluator searched on 12/01/2022 the National Vulnerability Database (NVD) from the NIST website and the Vulnerability Notes Database (VND) from the CERT Knowledgebase using the following terms:

- Infinera
- Infinera Corporation
- Transcend Network Management System
- TNMS
- Apache Xerces
- AdventNet
- AOP Alliance
- Apache Active MQ
- Apache Avalon
- Apache Commons
- Apache FOP
- Apache FTP server
- Apache HttpClient
- Apache ORO
- Apache Tomcat
- Apache Velocity
- Apache XML
- Apached Commons Codec
- ASM
- AspectJ
- Batik
- Bouncy Castle
- Castor
- cglib
- Disruptor
- Docking Frames
- Dom4j
- EdDSA
- edftpj
- EHCache
- Ganymed
- Guice
- google-collection
- gson
- Guava
- image4j
- InstallAnywhere
- io.grpc
- istack common
- Jakarta Activation
- Jakarta Mail
- Java Architecture for XML Binding
- Java Communications
- Java FX
- Java Swing
- javaee/glassfish V2 Milestone
- JavaHelp
- Javax Inject
- JavaZoom Basic Player
- JaxB Runtime
- JCalendar
- JDOM
- Jersey
- JFreeChart
- JFreeReport
- jgraphx
- JIDE
- Jlayer
- Jscape
- JSch
- JSR305
- JUnit
- jzlib
- Log4J
- MiGLayout
- MP3SPI
- OpenCensus
- OpenMap
- OpenProps
- Oracle Database JDBC Drivers
- Plexus
- reactive-streams
- reflections
- rxjava
- SLF4J
- snappy
- SNMP4j
- SSHJ
- Tritonus
- VorbisSPI
- VT Dictionary
- VT Password
- webdavilb
- webdavlib
- Wildfly
- Xalan
- XBean
- XML Commons.

The public vulnerability search did not return back any unresolved vulnerabilities.