www.GossamerSec.com

Assurance Activity Report for Cyber Reliant Mobile Data Defender for Android SDK Version 4.0

Version 0.3
03/16/23

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

Revision History

Revision	Date	Authors	Summary
Version 0.1	02/15/23	Smoley	Initial draft
Version 0.2	03/09/23	Smoley	First-round comments
Version 0.3	03/16/23	Smoley	Addressed ECR Comments

The TOE Evaluation was Sponsored by:

Cyber Reliant Corporation
National Security & Justice Group
180 Admiral Cochrane Drive Suite 310
Annapolis, MD 21401 U.S.A.

Evaluation Personnel:

- Yoel Fortaleza
- 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 Cyber Reliant Mobile Data Defender for Android SDK Version 4.0 ASPP14/FE10 evaluation. This document contains a description of the assurance activities and associated results as performed by the evaluators.

1.1 Device Equivalence

The Target of Evaluation (TOE) is the Cyber Reliant Mobile Data Defender for Android SDK Version 4.0 software application package residing on evaluated mobile devices running Android 11. The TOE is a software solution providing the capability to handle file encryption on mobile devices. The TOE is capable of running on Android 11 devices under VID11160, VID11211, or VID11315 with the same application supporting any of the devices. The TOE was tested on the following mobile devices representing one device from each VID.

Device Name	Chipset/CPU	Architecture	Android Version
Samsung S20 (VID11160)	Snapdragon 865	A64	11
Samsung Tab Active 3 (VID11211)	Exynos 9810	A64	11
Panasonic ToughBook FZN1 (VID11315)	Snapdragon 660 (SDM660)	A64	11

The devices above were identified based on availability for testing, however these devices were previously evaluated to exhibit the same behavior from a security function standpoint and any device could have been used in place for testing. Since the TOE is the same for the remaining devices under these evaluations, all other devices from these evaluations are claimed as equivalent.

VID11160

- Samsung S21 (S21 5G / S21+ 5G / S21 Ultra 5G)
- Samsung S21 (S21 5G / S21+ 5G / S21 Ultra 5G / S21 5G FE / Z Fold3 5G / Z Flip3 5G)
- Samsung S20 (S20 5G / S20+ 5G / S20 Ultra 5G / S20 LTE 5G / S20 5G FE / Note20 5G / Note20 LTE / Note20 Ultra LTE / Note20 Ultra 5G)
- Samsung S20 (S20+ 5G / S20 FE / S20 Ultra 5G / Z Flip 5G / Tab S7 / Tab S7+ / Note20 5G / Note20 Ultra 5G / Z Fold2 5G)
- Samsung XCover Pro / Samsung A51
- Samsung Note10 (Note10+ 5G / Note10+ / Note 10 5G / Note 10)
- Samsung S10e (S10+ / S10 5G / S10)
- Samsung S10+_(Note10+ 5G / Note10+ / Note 10 / Tab S6 / S10 5G / S10 / S10e / Fold 5G / Fold / Z Flip)

VID11211

- Samsung A52 5G (A52 5G / A42 5G)
- Samsung A71 5G (A71 5G / A51 5G)
- Samsung Tab Active 3

VID11315

- Panasonic ToughBook FZN1 (FZN1 / FZS1 / FZA3)

The TOE is composed of the two parts, the TOE management service and the TOE security library. The TOE management service is an Android APK that is used to manage (lock, unlock, change password, check for updates) the TOE encryption/decryption service. The TOE security library is a shared object file that is distributed by the vendor to be included in 3rd party libraries. The TOE security library is invoked by the 3rd party library to internally perform any encryption/decryption process as long as the TOE service is running and unlocked.

The evaluator also fully tested the TOE software on 3 separate platforms, the Samsung S20, Samsung Tab Active 3 and the Panasonic ToughBook FZ-N1. Similarly, the TOE's cryptographic library is CAVP-certified for the exact devices matching the tests results. Since the test platforms have been thoroughly evaluated within their VIDs to demonstrate the same behavior between the different devices claimed in their evaluations, the evaluation team concludes that the results are sufficient for any instance of the TOE running on the additional platforms claimed in the VID outside of the ones explicitly tested.

1.2 CAVP Algorithms

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

Functions	Standards	CAVP Certificates
Encryption/Decryption		
AES-CBC (256 bits)	FIPS PUB 197, NIST SP 800-38E	A3240
Cryptographic Hashing		
SHA-256, SHA-384, SHA-512	FIPS Pub 180-4	A3240
Keyed-Hash Message Authentication		
HMAC-SHA-512	FIPS Pub 198-1, FIPS Pub 180-4	A3240
Key Wrapping		
AES Key Wrap (256-bit)	NIST SP 800-38F	A3240
Key Transport		
RSA KTS-OAEP	NIST SP 800-56B, Revision 1	Vendor-Affirmed
Random Bit Generation		
AES-HASH-DRBG (256 bits)	NIST SP 800-90A	A3240

2. Protection Profile SFR Assurance Activities

This section of the AAR identifies each of the assurance activities included in the claimed Protection Profile and Extended Packages. This section also describes the findings for each activity.

The evidence identified below was used to perform these Assurance Activities:

- Cyber Reliant Mobile Data Defender for Android SDK Version 4.0 Security Target, version 0.6, 03/16/2023 (ST/KMD)
- User Guide Cyber Reliant Defender Installation and Use, March 2023

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 the TOE invokes the platform for asymmetric key generation.

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.1.1/AK

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 both the Management Service and Application's 2048-bit RSA keys used to wrap the FEKEK are generated using the KeyGenerator API. The ST only identifies one asymmetric scheme.

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 cryptography used by the TOE needs no configuration. As such, the User Guide contains no instructions related to RSA keys.

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 has been CAVP tested. Refer to the CAVP certificates identified in Section 1.2.

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

2.1.3.1 ASPP14:FCS_CKM.1.1/SK

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 the TOE generates file encryption keys using the CRC Encryption and Shredding engine cryptographic module which uses an SP 800-90A HASH- DRBG and the platform uses an SP 800-90A AES-256 CTR DRBG. Both DRBGs are seeded with sufficient entropy to generate keys with 256 bits of security strength by using seeding material collected by the evaluated platform.

Component Guidance Assurance Activities: None Defined

Component Testing Assurance Activities: None Defined

2.1.4 File Encryption Key (FEK) Generation (FE10:FCS_CKM_EXT.2)

2.1.4.1 FE10:FCS_CKM_EXT.2.1

TSS Assurance Activities: The evaluator shall review the TSS to determine that a description covering how and when the FEKs are generated exists. The description must cover all environments on which the TOE is claiming conformance, and include any preconditions that must exist in order to successfully generate the FEKs. The evaluator shall verify that the description of how the FEKs are generated is consistent with the instructions in the AGD guidance, and any differences that arise from different platforms are taken into account.

Conditional: If 'using a Random Bit Generator' was selected, the evaluator shall verify that the TSS describes how the functionality described by FCS_RBG_EXT.1 (from the [AppPP]) is invoked to generate FEK. To the extent

possible from the description of the RBG functionality in FCS_RBG_EXT.1 (from [AppPP]), the evaluator shall determine that the key size being requested is identical to the key size and mode to be used for the decryption/encryption of the user data (FCS_COP.1(1)) (from [AppPP]). (Per TD0455)

Conditional: If 'derived from a password/passphrase' is selected, the examination of the TSS section is performed as part of FCS_CKM_EXT.6 evaluation activities.

Section 6.1 of the ST states the TOE generates file encryption keys using the CRC Encryption and Shredding engine cryptographic module which uses an SP 800-90A HASH- DRBG and the platform uses an SP 800-90A AES-256 CTR DRBG. Both DRBGs are seeded with sufficient entropy to generate keys with 256 bits of security strength by using seeding material collected by the evaluated platform. The FEKs are generated every time a new file is going to be encrypted. The TOE associates a FEK with an individual file that is being encrypted by storing the wrapped FEK in the same hidden directory as the encrypted file. The TOE automatically generates a FEK (without user action) whenever the application attempts to encrypt a new file.

Guidance Assurance Activities: None Defined

Testing Assurance Activities: None Defined

2.1.4.2 FE10:FCS_CKM_EXT.2.2

TSS Assurance Activities: The evaluator shall verify the TSS describes how a FEK is used for a protected resource and associated with that resource. The evaluator confirms that-per this description-the FEK is unique per resource (file or set of files) and that the FEK is established using the mechanisms specified in FCS_CKM_EXT.2.1).

Section 6.1 of the ST states the FEKs are generated every time a new file is going to be encrypted. The TOE associates a FEK with an individual file that is being encrypted by storing the wrapped FEK in the same hidden directory as the encrypted file. The TOE automatically generates a FEK (without user action) whenever the application attempts to encrypt a new file.

The evaluator concludes the ST description of the FEK is unique per file and that the FEK is established using the DRBG mechanism specified in the SFR.

Guidance Assurance Activities: None Defined

Testing Assurance Activities: None Defined

Component TSS Assurance Activities: None Defined

Component Guidance Assurance Activities: The evaluator shall review the instructions in the AGD guidance to determine that any explicit actions that need to be taken by the user to establish a FEK exist-taking into account any differences that arise from different platforms-and are consistent with the description in the TSS.

Section 6.1 of the TSS states the TOE automatically generates a FEK (without user action) whenever the application attempts to encrypt a new file. As such, no information is required in the Guidance.

Component Testing Assurance Activities: None Defined

2.1.5 Key Encrypting Key (KEK) Support (FE10:FCS_CKM_EXT.3)

2.1.5.1 FE10:FCS_CKM_EXT.3.1

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 a description covering how and when KEK(s) are generated exists. The description must cover all environments on which the TOE is claiming conformance, and include any preconditions that must exist in order to successfully generate the KEKs. The evaluator shall verify that the description of how the KEK(s) are generated is consistent with the instructions in the AGD guidance, and any differences that arise from different platforms are taken into account.

Conditional: If using a RBG was selected the evaluator shall examine the TSS and verify that it describes how the functionality described by FCS_RBG_EXT.1 (from the [AppPP]) is invoked to generate KEK(s). To the extent possible from the description of the RBG functionality in FCS_RBG_EXT.1 (from [AppPP]), the evaluator shall determine that the key size being requested is identical to the key size selected.

Conditional: If derived from a password/passphrase is selected the examination of the TSS section is performed as part of FCS_CKM_EXT.6 evaluation activities.

Section 6.1 of the ST explains the platform DRBG is used to generate 256-bit AES FEKEKs. The file encryption key encryption key (FEKEK) is generated using the KeyGenerator API. The FEKEK is stored in the Management Service's WolfCrypt keystore. The FEKEK is wrapped twice, once using RSA and one more time using AES. The AES key used to wrap the FEKEK is derived from the DaR password using PBKDF2. Both the Management Service and Application's RSA keys used to wrap the FEKEK are generated using the KeyGenerator API. If the DaR password provided by the user is not correct, then the Management Service's keystore will not properly load, preventing the Management Service from accessing its keystore. Furthermore, an incorrect DaR password results in the incorrect derivation of the PBKDF2 derived AES key, therefore the FEKEK will not be unwrapped properly. The CRC Encryption and Shredding Engine Cryptographic module DRBG is used to generate the FEKs. The FEK is wrapped by the FEKEK before being stored.

For the DRBG selection in the SFR, the ST discussion matches the key size identified in the SFR. The password derived portion of the SFR is addressed in FE10: FCS_CKM_EXT.6.

Component Guidance Assurance Activities: The evaluator shall review the instructions in the AGD guidance to determine that any explicit actions that need to be taken by the user to establish a KEK exist-taking into account any differences that arise from different platforms-and are consistent with the description in the TSS.

Section 6.1 of the ST states the CRC Encryption and Shredding Engine Cryptographic module DRBG is used to generate the FEKs and requires no user action other than file access. As such, no information is required in the Guidance.

Component Testing Assurance Activities: None Defined

2.1.6 Cryptographic Key Destruction - per TD0644 (FE10:FCS_CKM_EXT.4)

2.1.6.1 FE10:FCS_CKM_EXT.4.1

TSS Assurance Activities: None Defined

Guidance Assurance Activities: None Defined

Testing Assurance Activities: None Defined

2.1.6.2 FE10:FCS_CKM_EXT.4.2

TSS Assurance Activities: None Defined

Guidance Assurance Activities: None Defined

Testing Assurance Activities: None Defined

Component TSS Assurance Activities: The evaluator shall verify the TSS provides a high level description of what it means for keys and key material to be no longer needed and when they should be expected to be destroyed. The evaluator shall verify the TSS provides a high level description of what it means for keys and key material to be no longer needed and when then should be expected to be destroyed.

KMD

The evaluator examines the KMD to ensure it describes how the keys are managed in volatile memory. This description includes details of how each identified key is introduced into volatile memory (e.g. by derivation from user input, or by unwrapping a wrapped key stored in non-volatile memory) and how they are overwritten.

The evaluator shall check to ensure the KMD lists each type of key that is stored in in non-volatile memory, and identifies how the TOE interacts with the underlying platform to manage keys (e.g., store, retrieve, destroy). The

description includes details on the method of how the TOE interacts with the platform, including an identification and description of the interfaces it uses to manage keys (e.g., file system APIs, platform key store APIs).

The evaluator examines the interface description for each different media type to ensure that the interface supports the selection(s) and description in the KMD.

If the ST makes use of the open assignment and fills in the type of pattern that is used, the evaluator examines the KMD to ensure it describes how that pattern is obtained and used. The evaluator shall verify that the pattern does not contain any CSPs.

The evaluator shall check that the KMD identifies any configurations or circumstances that may not strictly conform to the key destruction requirement.

If the selection 'destruction of all KEKs protecting target key, where none of the KEKs protecting the target key are derived' is included the evaluator shall examine the TOE's keychain in the KMD and identify each instance when a key is destroyed by this method. In each instance the evaluator shall verify all keys capable of decrypting the target key are destroyed in accordance with a specified key destruction method in FCS_CKM_EXT.4.1. The evaluator shall verify that all of the keys capable of decrypting the target key are not able to be derived to reestablish the keychain after their destruction.

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

The evaluator shall verify the KMD includes a key lifecycle, that includes a description where key material reside, how the key material is used, how it is determined that keys and key material are no longer needed, and how the material is destroyed once it is not needed and that the documentation in the KMD follows FCS_CKM_EXT.4.1 for the destruction.

Section 6.1 of the ST states the TOE relies on the platform and CRC Encryption and Shredding Engine Cryptographic module for destroying keys. The platform utilizes Java Garbage Collection in order to clear memory. The TOE releases all references to objects (e.g. keys) when they are no longer needed, and the Java Garbage Collection clears out the memory that is no longer in use. The CRC Encryption and Shredding Engine Cryptographic module also utilizes platform functions to destroy FEKs. The CRC Encryption and Shredding Engine Cryptographic module destroys API overwrites the reference to the key directly and initiates a request for garbage collection for the destruction of non-persistent CSPs from volatile memory.

The ST provides a table identifying each key, where it is destroyed, when it is destroyed and how it is destroyed.

The evaluator examines the KMD (ST) to ensure it describes how the keys are managed in volatile memory. This description includes details of how each identified key is introduced into volatile memory (e.g. by derivation from user input, or by unwrapping a wrapped key stored in non-volatile memory) and how they are overwritten.

Component Guidance Assurance Activities: There are a variety of concerns that may prevent or delay key destruction in some cases.

The evaluator shall check that the guidance documentation identifies configurations or circumstances that may not strictly conform to the key destruction requirement, and that this description is consistent with the relevant parts of the TSS and any other relevant Required Supplementary Information.

The evaluator shall check that the guidance documentation provides guidance on situations where key destruction may be delayed at the physical layer and how such situations can be avoided or mitigated if possible.

Some examples of what is expected to be in the documentation are provided here.

When the TOE does not have full access to the physical memory, it is possible that the storage may be implementing wearleveling and garbage collection. This may create additional copies of the key that are logically inaccessible but persist physically. In this case, to mitigate this the drive should support the TRIM command and implements garbage collection to destroy these persistent copies when not actively engaged in other tasks.

Drive vendors implement garbage collection in a variety of different ways, as such there is a variable amount of time until data is truly removed from these solutions. There is a risk that data may persist for a longer amount of time if it is contained in a block with other data not ready for erasure. To reduce this risk, the operating system and file system of the OE should support TRIM, instructing the non-volatile memory to erase copies via garbage collection upon their deletion. If a RAID array is being used, only set-ups that support TRIM are utilized. If the drive is connected via PCI-Express, the operating system supports TRIM over that channel.

The drive should be healthy and contains minimal corrupted data and should be end of lifed before a significant amount of damage to drive health occurs, this minimizes the risk that small amounts of potentially recoverable data may remain in damaged areas of the drive.

The evaluator concludes that no Guidance information was required for this TOE. This TOE is a software application and memory is cleared with Java garbage collection. The TOE and its users are not aware of wear leveling, RAID, and other physical concerns on the device.

Component Testing Assurance Activities: These tests are only for key destruction provided by the application, test 2 does not apply to any keys using the selection 'new value of a key':

Test 1: [Conditional; applies when the application does not perform the zeroization (ie. garbage collecting) for each key held in volatile memory for FCS_CKM_EXT.4.1 (assuming the selection 'destruction of the reference followed by a request for garbage collection')] Applied to each key held in volatile memory and subject to destruction by overwrite by the TOE (whether or not the value is subsequently encrypted for storage in volatile or non-volatile memory). In the case where the only selection made for the key destruction method was removal of power, then this test is unnecessary.

The evaluator shall:

1. Record the value of the key in the TOE subject to clearing.

2. Cause the cause the TOE or the underlying platform to dump to perform a normal cryptographic processing with the key from Step #1.

3. Cause the TOE to clear the key.

4. Cause the TOE to stop the execution but not exit.

5. Cause the TOE to dump the entire memory of the TOE into a binary file.

6. Search the content of the binary file created in Step #5 for instances of the known key value from Step #1.

Steps #1-6 ensure that the complete key does not exist anywhere in volatile memory. If a copy is found, then the test fails.

Test 2: [Conditional; applies when instructing the underlying platform to destroy the key] If new value of a key is selected this test does not apply.

Applied to each key held in non-volatile memory and subject to destruction by the TOE.

The evaluator shall use special tools (as needed), provided by the TOE developer if necessary, to ensure the tests function as intended.

1. Identify the purpose of the key and what access should fail when it is deleted. (e.g. the file encryption key being deleted would cause data decryption to fail.)

2. Cause the TOE to clear the key.

3. Have the TOE attempt the functionality that the cleared key would be necessary for.

4. The test succeeds if Step #3 fails.

Tests 3 and 4 do not apply for the selection instructing the underlying platform to destroy the representation of the key, as the TOE has no visibility into the inner workings and completely relies on the underlying platform.

Test 3: Applied to each key held in non-volatile memory and subject to destruction by overwrite by the TOE. The evaluator shall use a tool that provides a logical view of the media (e.g., MBR file system):

1. Record the value of the key in the TOE subject to clearing.

2. Cause the TOE to perform a normal cryptographic processing with the key from Step #1.

3. Cause the TOE to clear the key.

4. Search the logical view that the key was stored in for instances of the known key value from Step #1. If a copy is found, then the test fails.

Test 4: Applied to each key held in non-volatile memory and subject to destruction by overwrite by the TOE. The evaluator shall use a tool that provides a logical view of the media:

1. Record the logical storage location of the key in the TOE subject to clearing.

2. Cause the TOE to perform a normal cryptographic processing with the key from Step #1.

3. Cause the TOE to clear the key.

4. Read the logical storage location in Step #1 of non-volatile memory to ensure the appropriate pattern is utilized.

The test succeeds if correct pattern is used to overwrite the key in the memory location. If the pattern is not found the test fails.

Test 1 – For keys held in volatile memory and subject to garbage collection (FEKEK, PBKDF derived key, and FEK), the evaluator used the keys for their intended purposes, logged their values, and then caused the TOE to clear the keys. The evaluator searched with and without the TOE service locked and did not find any of the keys in volatile memory.

Test 2 – The TOE offers no mechanism to clear keys, other than uninstalling their parent applications. The evaluator installed both the TOE management application and the example File Browser and ensure that they correctly operated. The evaluator noted that the only used for the volatile keys identified was to encrypt the FEKEK for long-term storage and to transport the FEKEK between applications. The evaluator then individually uninstalled each and showed that it removed access to the FEKEK and also the files which were encrypted using keys secured by the FEKEK.

Test 3 and 4 – Not applicable, as the TOE does not store plaintext values of its security keys in volatile memory except for the RSA keypairs, which are created in the platform keystore. The platform keystore provides no visibility into the inner workings and are not subject to overwrite by the TOE.

2.1.7 Cryptographic Password/Passphrase Conditioning (FE10:FCS_CKM_EXT.6)

2.1.7.1 FE10:FCS_CKM_EXT.6.1

TSS Assurance Activities: There are two aspects of this component that require evaluation: passwords/passphrases of the length specified in the requirement (at least 64 characters or a length defined by the platform) are supported, and that the characters that are input are subject to the selected conditioning function. These activities are separately addressed in the text below.

Support for minimum length: The evaluators shall check to ensure that the TSS describes the allowable ranges for password/passphrase lengths, and that at least 64 characters or a length defined by the platform may be specified by the user.

Support for character set: The evaluator shall check to ensure that the TSS describes the allowable character set and that it contains the characters listed in the SFR.

Support for PBKDF: The evaluator shall examine the TSS to ensure that the formation of all KEKs or FEKs (as decided in the FCS_CKM_EXT.3 selection) is described and that the key sizes match that described by the ST author.

The evaluator shall check that the TSS describes the method by which the password/passphrase is first encoded and then fed to the SHA algorithm. The settings for the algorithm (padding, blocking, etc.) shall be described, and the evaluator shall verify that these are supported by the selections in this component as well as the selections concerning the hash function itself. The evaluator shall verify that the TSS contains a description of how the output of the hash function is used to form the submask that will be input into the function and is the same length as the KEK as specified in FCS_CKM_EXT.4.

For the NIST SP 800-132-based conditioning of the password/passphrase, the required evaluation activities will be performed when doing the evaluation activities for the appropriate requirements (FCS_COP.1.1(4)). If any manipulation of the key is performed in forming the submask that will be used to form the FEK or KEK, that process shall be described in the TSS.

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

Section 6.1 of the ST describes the TOE allows the use of DaR passwords that support all special characters mentioned inFE10: FCS_CKM_EXT.6. The TOE encodes the DaR password using UTF-8 before the DaR password is passed into the evaluated Android platform's cryptographic APIs to perform Password-Based key derivation (SP 800-132 PBKDF2) using HMAC-SHA-512 pseudo-random function. Note that this DaR password does not derive the FEK. The password input into the PBKDF2 function results in a HMAC-SHA-512 output that is 512 bits long. The TOE truncates this output to a length of 256 bits without any further manipulation and uses that value as a derived key to perform a secondary AES wrap on the FEKEK in conjunction with an asymmetric key wrap using an RSA key pair stored in the Android keystore. The TOE enforces a minimum password length of 6 characters and can support a maximum password of 128 characters.

Guidance Assurance Activities: None Defined

Testing Assurance Activities: None Defined

2.1.7.2 FE10:FCS_CKM_EXT.6.2

TSS Assurance Activities: The ST author shall provide a description in the TSS regarding the salt generation. The evaluator shall confirm that the salt is generated using an RBG described in FCS_RBG_EXT.1 (from the [AppPP]).

Section 6.1 of the ST (ASPP14:FCS_COP.1/KeyedHash) explains the TOE uses the platform HMAC-SHA-512 to hash a PBKDF2 derived key with a second salt to produce a key used to further unwrap the FEKEK using AES key wrap, after the FEKEK has been unwrapped using RSA-OAEP.

Guidance Assurance Activities: None Defined

Testing Assurance Activities: None Defined

2.1.7.3 FE10:FCS_CKM_EXT.6.3

TSS Assurance Activities: None Defined

Guidance Assurance Activities: None Defined

Testing Assurance Activities: None Defined

2.1.7.4 FE10:FCS_CKM_EXT.6.4

TSS Assurance Activities: None Defined

Guidance Assurance Activities: None Defined

Testing Assurance Activities: None Defined

2.1.7.5 FE10:FCS_CKM_EXT.6.5

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: Support for minimum length: The evaluators shall check the Operational Guidance to determine that there are instructions on how to generate large passwords/passphrases, and instructions on how to configure the password/passphrase length to provide entropy commensurate with the keys that the authorization factor is protecting.

Section 4.1 of the User Guide states that for a user to create a strong passphrase, a passphrase with at least one uppercase letter, lowercase letter, special character, and number is recommended. Longer length passphrases provide increased security strength over shorter ones. Acceptable password strength should reflect the sensitivity of the data being encrypted. Higher data sensitivity should have increased password strength.

Component Testing Assurance Activities: Support for Password/Passphrase characteristics: In addition to the analysis above, the evaluator shall also perform the following tests on a TOE configured according to the Operational Guidance:

Test 1: Ensure that the TOE supports password/passphrase lengths as defined in the SFR assignments.

Test 2: Ensure that the TOE does not accept more than the maximum number of characters specified in FCS_CKM_EXT.6.1.

Test 3: Ensure that the TOE does not accept less than the minimum number of characters specified in FCS_CKM_EXT.6.4. If the minimum length is settable by the administrator, the evaluator determines the minimum length or lengths to test.

Test 4: Ensure that the TOE supports passwords consisting of all characters listed in FCS_CKM_EXT.6.2.

Conditioning: No explicit testing of the formation of the authorization factor from the input password/passphrase is required.

Test 1 – The evaluator configured a password matching the claimed minimum (6) and claimed maximum (128) character length. Both password changes were successful as both lengths were accepted.

Test 2 – The evaluator configured a password that exceeded the maximum value (129) specified in FCS_CKM_EXT.6.1 in the ST and confirmed that the TOE displayed an error and would not allow the user to continue with the password changing process until a smaller password was entered.

Test 3 – The evaluator configured a password that was below the minimum value (6) claimed in FCS_CKM_EXT.5.4 in the ST and confirmed the TOE displayed an error and would not allow the user to continue with the password changing process until a larger password was entered.

Test 4 – The evaluator set the password using each of the characters in the ST. The evaluator was able to use each claimed character in a password.

2.1.8 Cryptographic operation (Key Wrapping) (FE10:FCS_COP.1(5))

2.1.8.1 FE10:FCS_COP.1.1(5)

TSS Assurance Activities: None Defined

Guidance Assurance Activities: None Defined

Testing Assurance Activities: None Defined

Component TSS Assurance Activities: Conditional: If use platform provided functionality was selected, then the evaluator shall examine the TSS to verify that it describes how the FEK encryption/decryption is invoked.

Conditional: If implement functionality was selected, The evaluator shall check that the TSS includes a description of encryption function(s) used for key wrapping. The evaluator should check that this description of the selected

encryption function includes the key sizes and modes of operations as specified in the selection above. The evaluator shall check that the TSS describes the means by which the TOE satisfies constraints on algorithm parameters included in the selections made for 'cryptographic algorithm' and 'list of standards'.

The evaluator shall verify the TSS includes a description of the key wrap function(s) and shall verify the key wrap uses an approved key wrap algorithm according to the appropriate specification.

KMD

The evaluator shall review the KMD to ensure that all keys are wrapped using the approved method and a description of when the key wrapping occurs.

The TOE claims to implement functionality to perform key wrapping.

Section 6.1 of the ST states the TOE uses its WolfSSL Module 256 bit AES key wrapping to wrap the FEK with the FEKEK. The TOE performs AES key wrapping in accordance with SP 800-38F and RSA key wrapping in accordance with SP 800-56B. The descriptions match the SFR selection.

All descriptions of FEK key wrapping are using an approved method and the KMD (ST) includes no additional unapproved methods of key wrapping.

Component Guidance Assurance Activities: If multiple encryption modes are supported, the evaluator examines the guidance documentation to determine that the method of choosing a specific mode/key size by the end user is described.

Multiple encryption modes are not supported. As such, no Guidance is required.

Component Testing Assurance Activities: The evaluation activity tests specified for AES in GCM mode in the underlying [AppPP] shall be performed in the case that 'GCM' is selected in the requirement.

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

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

parameter lengths:

128 and 256 bit key encryption keys (KEKs)

Three plaintext lengths. One of the plaintext lengths shall be two semi-blocks (128 bits). One of the plaintext lengths shall

be three semi-blocks (192 bits). The third data unit length shall be the longest supported plaintext length less than or

equal to 64 semi-blocks (4096 bits).

using a set of 100 key and plaintext pairs and obtain the ciphertext that results from AES-KW authenticated encryption. To

determine correctness, the evaluator will use the AES-KW authenticated-encryption function of a known good implementation.

The evaluator will test the authenticated-decryption functionality of AES-KW using the same test as for authenticatedencryption, replacing plaintext values with ciphertext values and AES-KW authenticated-encryption with AES-KW

authenticated-decryption.

The evaluator will test the authenticated-encryption functionality of AES-KWP using the same test as for AES-KW

authenticated-encryption with the following change in the three plaintext lengths:

One plaintext length shall be one octet. One plaintext length shall be 20 octets (160 bits).

One plaintext length shall be the longest supported plaintext length less than or equal to 512 octets (4096 bits).

The evaluator will test the authenticated-decryption functionality of AES-KWP using the same test as for AES-KWP

authenticated-encryption, replacing plaintext values with ciphertext values and AES-KWP authenticated-encryption with AESKWP authenticated-decryption.

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 Text

For each supported key size and payload length, and any supported associated data length, nonce length, and tag length, the

evaluator shall supply one key value, one nonce value, and 10 pairs of associated data and payload values, and obtain the

resulting ciphertext.

Variable Nonce Test

For each supported key size and nonce length, and any supported associated data length, payload length, and tag length, the

evaluator shall supply one key value, one nonce value, and 10 pairs of associated data and payload values, and obtain the

resulting ciphertext.

Variable Tag Test

For each supported key size and tag length, and any supported associated data length, payload length, and nonce length, the

evaluator shall supply one key value, one nonce value, and 10 pairs of associated data and payload values, and obtain the

resulting ciphertext.

Decryption-Verification Process Test

To test the decryption-verification functionality of AES-CCM, for each combination of supported associated data length,

payload length, nonce length, and tag length, the evaluator shall supply a key value and 15 sets of input plus ciphertext, and

obtain the decrypted payload. Ten of the 15 input sets supplied should fail verification and five should pass.

The TOE has been CAVP tested. Refer to the CAVP certificates identified in Section 1.2.

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

2.1.9.1 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.

No user configuration is available for the cryptographic features of the TOE.

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 has been CAVP tested. Refer to the CAVP certificates identified in Section 1.2.

2.1.10 Cryptographic operation (Key Transport) (FE10:FCS_COP.1(6))

2.1.10.1 FE10:FCS_COP.1.1(6)

TSS Assurance Activities: None Defined

Guidance Assurance Activities: None Defined

Testing Assurance Activities: None Defined

Component TSS Assurance Activities: The evaluator shall verify the TSS provides a high level description of the RSA scheme and the cryptographic key size that is being used, and that the asymmetric algorithm being used for key transport is RSA. If more than one scheme/key size are allowed, then the evaluator shall make sure and test all combinations of scheme and key size. There may be more than one key size to specify - an RSA modulus size (and/or encryption exponent size), an AES key size, hash sizes, MAC key/MAC tag size.

If the KTS-OAEP scheme was selected, the evaluator shall verify that the TSS identifies the hash function, the mask generating function, the random bit generator, the encryption primitive and decryption primitive. If the KTS-KEM-KWS scheme was selected, the evaluator shall verify that the TSS identifies the key derivation method,

Section 6.1 of the ST states the TOE uses its WolfSSL Module 256 bit AES key wrapping to wrap the FEK with the FEKEK. The TOE performs AES key wrapping in accordance with SP 800-38F and RSA key wrapping in accordance with SP 800-56B. The TOE leverages platform APIs to perform RSA-OAEP key transport using 3072-bit

RSA/ECB/NoPadding, SHA-512 for hashing and mask generation, and the platform's invoked DRBG source though Java's SecureRandom API's.

Component Guidance Assurance Activities: None Defined

Component Testing Assurance Activities: For each supported key transport schema, the evaluator shall initiate at least 25 sessions that require key transport with an independently developed remote instance of a key transport entity, using known RSA key-pairs. The evaluator shall observe traffic passed from the sender-side and to the receiver-side of the TOE, and shall perform the following tests, specific to which key transport scheme was employed. If the KTS-OAEP scheme was selected, the evaluator shall perform the following tests:

Test 1: The evaluator shall inspect each cipher text, C, produced by the RSA-OAEP encryption operation of the TOE and make sure it is the correct length, either 256 or 384 bytes depending on RSA key size. The evaluator shall also feed into the TOE's RSA-OEAP decryption operation some cipher texts that are the wrong length and verify that the erroneous input is detected and that the decryption operation exits with an error code.

Test 2: The evaluator shall convert each cipher text, C, produced by the RSA-OAEP encryption operation of the TOE to the correct cipher text integer, c, and use the decryption primitive to compute em = RSADP((n,d),c) and convert em to the encoded message EM. The evaluator shall then check that the first byte of EM is 0x00. The evaluator shall also feed into the TOE's RSA-OEAP decryption operation some cipher texts where the first byte of EM was set to a value other than 0x00, and verify that the erroneous input is detected and that the decryption operation exits with an error code.

Test 3: The evaluator shall decrypt each cipher text, C, produced by the RSA-OAEP encryption operation of the TOE using RSADP, and perform the OAEP decoding operation (described in NIST SP 800-56B section 7.2.2.4) to recover HA' || X. For each HA', the evaluator shall take the corresponding A and the specified hash algorithm and verify that HA' = Hash(A). The evaluator shall also force the TOE to perform some RSA-OAEP decryption where the A value is passed incorrectly, and the evaluator shall verify that an error is detected.

Test 4: The evaluator shall check the format of the 'X' string recovered in OAEP.Test.3 to ensure that the format is of the form PS || 01 || K, where PS consists of zero or more consecutive 0x00 bytes and K is the transported keying material. The evaluator shall also feed into the TOE's RSA-OEAP decryption operation some cipher texts for which the resulting 'X' strings do not have the correct format (i.e., the leftmost non-zero byte is not 0x01). These incorrectly formatted 'X' variables shall be detected by the RSA-OEAP decrypt function.

Test 5: The evaluator shall trigger all detectable decryption errors and validate that the returned error codes are the same and that no information is given back to the sender on which type of error occurred. The evaluator shall also validate that no intermediate results from the TOE's receiver-side operations are revealed to the sender.

If the KTS-KEM-KWS scheme was selected, the evaluator shall perform the following tests:

Test 1: The evaluator shall inspect each cipher text, C, produced by KTS-KEM-KWS encryption operation of the TOE and make sure the length (in bytes) of the cipher text, cLen, is greater than nLen (the length, in bytes, of the modulus of the RSA public key) and that cLen - nLen is consistent with the byte lengths supported by the key wrapping algorithm. The evaluator shall feed into the KTS-KEM-KWS decryption operation a cipher text of unsupported length and verify that an error is detected and that the decryption process stops.

Test 2: The evaluator shall separate the cipher text, C, produced by the sender-side of the TOE into its C0 and C1 components and use the RSA decryption primitive to recover the secret value, Z, from C0. The evaluator shall check that the unsigned integer represented by Z is greater than 1 and less than n-1, where n is the modulus of the RSA public key. The evaluator shall construct examples where the cipher text is created with a secret value Z = 1 and make sure the KTS-KEM-KWS decryption process detects the error. Similarly, the evaluator shall construct examples where the cipher text is created with a secret value Z = n - 1 and make sure the KTS-KEM-KWS decryption process detects the error.

Test 3: The evaluator shall attempt to successfully recover the secret value Z, derive the key wrapping key, KWK, and unwrap the KWA-cipher text following the KTS-KEM-KWS decryption process given in NISP SP 800-56B section 7.2.3.4. If the key-wrapping algorithm is AES-CCM, the evaluator shall verify that the value of any (unwrapped) associated data, A, that was passed with the wrapped keying material is correct The evaluator shall feed into the TOE's KTS-KEM-KWS decryption operation examples of incorrect cipher text and verify that a decryption error is detected. If the key-wrapping algorithm is AES-CCM, the evaluator shall attempt at least one decryption where the wrong value of A is given to the KTS-KEM-KWS decryption operation and verify that a decryption error is detected. Similarly, if the key-wrapping algorithm is AES-CCM, the evaluator shall attempt at least one decryption where the wrong nonce is given to the KTSKEM-KWS decryption operation and verify that a decryption error is detected.

Test 4: The evaluator shall trigger all detectable decryption errors and validate that the resulting error codes are the same and that no information is given back to the sender on which type of error occurred. The evaluator shall also validate that no intermediate results from the TOE's receiver-side operations (in particular, no Z values) are revealed to the sender.

The TOE supports KTS-OAEP with cryptographic keys of size 3072. The evaluator initiated at least 25 sessions that require key transport with an independently developed remote instance of a key transport entity in conjunction with the following tests, using RSA key-pairs of size 3072.

Test 1 – The evaluator processed 10 correct and 10 incorrect cipher texts using the TOE's RSA-OAEP test harness. The evaluator inspected each of the correct ciphertexts and found that they were 256 bytes corresponding to the claimed 3072 RSA key size. The evaluator inspected the incorrect length cipher texts and verified that the erroneous input was detected and that the decryption operation exits with an InvalidCipherTextException.

Test 2: The evaluator converted each cipher text, C, produced by the RSA-OAEP encryption operation of the TOE to the correct cipher text integer, c, and used the decryption primitive to compute em = RSADP((n,d),c) and convert em to the encoded message EM. The evaluator repeated this process for 10 correct cipher texts and checked that the first byte of EM is 0x00. The evaluator fed into the TOE's RSA-OEAP decryption operation 10 cipher texts where

the first byte of EM was set to a value other than 0x00, and verify that the erroneous input is detected and that the decryption operation exits with an InvalidCipherTextException.

Test 3: The evaluator decrypted each cipher text, C, produced by the RSA-OAEP encryption operation of the TOE using RSADP, and performed the OAEP decoding operation (described in NIST SP 800-56B section 7.2.2.4) to recover HA' || X. For each of the 10 correct HA' tested by the evaluator, the evaluator took the corresponding A and the specified hash algorithm and verified that HA' = Hash(A). The evaluator forced the TOE to perform 10 additional RSA-OAEP decryptions where the A value is passed incorrectly, and verified that an InvalidCipherTextExcpetion error is detected.

Test 4: The evaluator checked the format of the 'X' string to ensure that the format is of the form PS || 01 || K, where PS consists of zero or more consecutive 0x00 bytes and K is the transported keying material. The evaluator fed into the TOE's RSA-OEAP decryption operation 10 cipher texts for which the resulting 'X' strings did have the correct format and verified that these operations succeeded. The evaluator then fed 10 incorrect cipher texts where the 'X' strings did not have the correct format (i.e., the leftmost non-zero byte is not 0x01) and verified these incorrectly formatted 'X' variables were detected by the RSA-OEAP decrypt function with an InvalidCipherTextException.

Test 5 – This was tested in conjunction with the previous 4 tests. The evaluator triggered all known detectable decryption errors and validated that the returned error codes are the same (InvalidCipherTextExcpetion) and that no information is given back to the sender on which type of error occurred. The evaluator witnessed no intermediate results from the TOE's receiver-side operations are revealed to the sender at any point.

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

2.1.11.1 ASPP14:FCS_COP.1.1/Hash

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 the TOE performs SHA hashing using its WolfSSL Module. The TOE supports hash sizes of SHA-256, SHA-384, SHA-512 and message digest sizes 256, 384, and 512 bits. The TOE uses its SHA_256 function to support the HASH-DRBG. The TOE uses its WolfSSL Module SHA-384 to hash the DaR password with a salt generated by the platform's DRBG. The TOE uses the 512 hash function to support the keyed hash function.

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 has been CAVP tested. Refer to the CAVP certificates identified in Section 1.2.

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

2.1.12.1 ASPP14:FCS_COP.1.1/KeyedHash

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 has been CAVP tested. Refer to the CAVP certificates identified in Section 1.2.

2.1.13 Initialization Vector Generation (FE10:FCS_IV_EXT.1)

2.1.13.1 FE10:FCS_IV_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 ensure the TSS describes how nonces are created uniquely and how IVs and tweaks are handled (based on the AES mode). The evaluator shall confirm that the nonces are unique and the IVs and tweaks meet the stated requirements.

Section 6.1 of the ST states the TOE calls the CRC Encryption and Shredding Engine Cryptographic module's API to access that module's Hash_DRBG to generate IV values. By using a Hash_DRBG to generate its IVs, the TOE guarantees those IVs to be non-repeating and unpredictable. The TOE's [AGD] contains the full list of APIs used by the TOE. The TOE only uses IVs when performing AES-256 encryption/decryption of user data using the FEK (IVs are not used as part of any key wrap/unwrap process).

Component Guidance Assurance Activities: None Defined

Component Testing Assurance Activities: None Defined

2.1.14 Key Chaining and Key Storage (FE10:FCS_KYC_EXT.1)

2.1.14.1 FE10:FCS_KYC_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 the TSS contains a high level description of all keychains and authorization methods selected in FIA_AUT_EXT.1 that are used to protect the KEK or FEK.

KMD

The evaluator shall examine the KMD to ensure it describes each key chain in detail, and these descriptions correspond with the selections of the requirement. The description of each key chain shall be reviewed to ensure the options for maintaining the key chain are documented.

The evaluator shall verify the KMD to ensure that it describes how each key chain process functions, such that it does not expose any material that might compromise any key in the chain. A high-level description should include a diagram illustrating the keychain(s) implemented and detail where all keys and keying material is stored or how the keys or key material are derived. The evaluator shall examine the primary key chain to ensure that at no point the chain could be broken without a cryptographic exhaust or knowledge of the KEK or FEK and the effective strength of the FEK is maintained throughout the Key Chain as specified in the requirement.

Section 6.1 of the ST states the DaR password is used in two places. First, it is hashed using SHA-384 with a salt (generated using the platform's DRBG) and used to load the Management Service's keystore. The DaR password is then used as input to PBKDF2 with 4096 iterations and HMAC-SHA-512 PRF along with a second salt value (also generated using the platform's DRBG).

The platform retrieves the double-wrapped FEKEK from the Management Service's keystore. The double wrapped FEKEK is AES unwrapped using the 256-bit PBKDF2 derived key, resulting in an RSA wrapped FEKEK.

The RSA wrapped FEKEK is unwrapped using the Management Service's RSA keypair, resulting in a cleartext FEKEK that is re-encrypted using the Application's RSA public key. The Management Service passes this single wrapped FEKEK to the application. The Management Service obtains the Application's RSA public key as well as the Application's certificate fingerprint when the Application registers itself to the Management Service (performed once). The Application's RSA public key is kept within Android's SharedPreferences under MODE_PRIVATE.

The Management Service uses the Application's certificate fingerprint to read/write to the Application's keystore.

All selections made in the SFRs are reflected in the ST discussion. The KMD includes a key hierarchy diagram. The evaluator has examined it for possible points of vulnerability and not identified any weaknesses where the chain could be broken.

Component Guidance Assurance Activities: None Defined

Component Testing Assurance Activities: None Defined

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

2.1.15.1 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 requirement selected both invoke and implement.

For the implement selection, see ASPP14:FCS_RBG_EXT.2

For invoke, section 6.1 of the ST states the TOE generates the FEKEK using the platform's AES-256 AES CTR DRBG. It also states (in ASPP14:FCS_CKM.1) that the platform's API KeyGenerator is used.

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 requirement selected both invoke and implement. The testing activity for this SFR covers invoked behavior.

Test 1 – The evaluator decompiled the TOE application and security library and found references to the javax.crypto.KeyGenerator and java.security.SecureRandom classes.

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

2.1.16.1 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 has been CAVP tested. Refer to the CAVP certificates identified in Section 1.2.

2.1.16.2 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.

The Entropy description is provided in a separate (non-ST and non-Guidance) document that has been approved by NIAP.

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.

The entropy description is provided in a separate (non-ST and non-Guidance) document that has been delivered to NIAP for approval.

Component TSS Assurance Activities: None Defined

Component Guidance Assurance Activities: None Defined

Component Testing Assurance Activities: None Defined

2.1.17 Storage of Credentials (ASPP14:FCS_STO_EXT.1)

2.1.17.1 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.

Section 6.1 of the ST states the All claimed keystores live in Android's protected directory /data/misc/keystore, which is also on the platform's flash storage. The TOE uses the evaluated Android platform's Android keystore to store all RSA keypairs.

Upon registration, each Application passes its public key to the Management Service, which is stored in Android's SharedPreferences under MODE_PRIVATE. The TOE uses the Application's public key to wrap the FEKEK for unwrapping and use by the application.

- The TOE uses the Management Service's keypair to keywrap the FEKEK, which is again wrapped by the PBKDF-derived key to form the double wrapped FEKEK for storage.
- The wrapped FEK and IV are stored in the same hidden directory as the encrypted file, which is all stored on flash

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

Test 1 – The TOE only invokes platform functionality to securely store keys to non-volatile memory. To verify the correct API invocations were used, the evaluator decompiled the TOE application and TOE security library and found several references to Keystore and Keychain.

2.1.18 Validation (FE10:FCS_VAL_EXT.1)

2.1.18.1 FE10:FCS_VAL_EXT.1.1

TSS Assurance Activities: None Defined

Guidance Assurance Activities: None Defined

Testing Assurance Activities: None Defined

2.1.18.2 FE10:FCS_VAL_EXT.1.2

TSS Assurance Activities: None Defined

Guidance Assurance Activities: None Defined

Testing Assurance Activities: None Defined

Component TSS Assurance Activities: Conditional: If 'validating' is selected in FCS_VAL_EXT.1.1, the evaluator shall examine the TSS to determine which authorization factors support validation.

The evaluator shall examine the TSS to ensure that it contains a high-level description of how the submasks are validated. If multiple submasks are used within the TOE, the evaluator shall verify that the TSS describes how each is validated (e.g., each submask validated before combining, once combined validation takes place).

Conditional: If 'receiving assertion' is selected in FCS_VAL_EXT.1.1, the evaluator shall examine the TSS to verify that it describes the enviroments that can be leveraged with the TOE and how each claims to perform validation. The evaluator shall ensure that none of the stated platform validation mechanisms weaken the key chain of the product.

Section 6.1 of the ST states the TOE only allows for a single password to be used for authentication

Section 6.1 of the ST states The DaR password is used in two places. First, it is hashed using SHA-384 with a salt (generated using the platform's DRBG) and used to load the Management Service's keystore. The DaR password is then used as input to PBKDF2 with 4096 iterations and HMAC-SHA-512 PRF along with a second salt value (also generated using the platform's DRBG).

The platform retrieves the double-wrapped FEKEK from the Management Service's keystore. The double wrapped FEKEK is AES unwrapped using the 256-bit PBKDF2 derived key, resulting in an RSA wrapped FEKEK.

The RSA wrapped FEKEK is unwrapped using the Management Service's RSA keypair, resulting in a cleartext FEKEK that is re-encrypted using the Application's RSA public key. The Management Service passes this single wrapped FEKEK to the application. The Management Service obtains the Application's RSA public key as well as the Application's certificate fingerprint when the Application registers itself to the Management Service (performed once). The Application's RSA public key is kept within Android's SharedPreferences under MODE_PRIVATE.

The Management Service uses the Application's certificate fingerprint to read/write to the Application's keystore.

Component Guidance Assurance Activities: If the validation functionality is configurable, the evaluator shall examine the operational guidance to ensure it describes how to configure the TOE to ensure the limits regarding validation attempts can be established.

The validation function is not configurable. As such, no Guidance is needed.

Component Testing Assurance Activities: None Defined

2.1.19 Validation Remediation (FE10:FCS_VAL_EXT.2)

2.1.19.1 FE10:FCS_VAL_EXT.2.1

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 determine which remediation options are supported for which authentication options.

Section 6.1 of the ST explains the TOE only allows for a single password to be used for authentication. After 5 attempts at authenticating with the wrong password, the TOE will lock out the service for 3600 seconds (60 minutes) before allowing the user to attempt a password authentication again. This effectively sets a limit at a total of 120 attempts per 24 hour period.

Component Guidance Assurance Activities: If the remediation functionality is configurable, the evaluator shall examine the operational guidance to ensure it describes how to configure the TOE to ensure the limits regarding validation attempts can be established.

The remediation function is not configurable. As such, no Guidance is needed

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

Test 1: The evaluator shall determine the limit on the average rate of the number of consecutive failed authorization attempts. For each authentication factor supported, the evaluator will test the TOE by entering that number of incorrect authorization factors in consecutive attempts to access the protected data. If the limit mechanism includes any 'lockout' period, the time period tested should include at least one such period. Then the evaluator will verify that the TOE behaves as described in the TSS.

Test 1 – The evaluator attempted 5 password attempts on the TOE management application with an incorrect password. The evaluator found that a failed attempt caused an error on the TOE and five failed attempts caused an additional message signifying the user was locked out from making additional attempts. After waiting 59 minutes the evaluator found that the user was still locked out. After an additional minute, the evaluator was able to attempt another password attempt. The evaluator attempted another 5 failed passwords again and observed the user was locked out for another additional hour. Following this, and vendor guidance, the evaluator concluded

that all subsequent lockout periods should follow the same timeout period, meaning a user can attempt at most 5 failed password attempts per hour, or 120 in a 24-hour period.

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.

Section 6.2 of the ST states the TOE offers two groups of API: one set to manipulate files and one set to manipulate SQLite databases. While the API groups provide different abstractions for the read and write operations, they both are ultimately simply reading and writing a single file. The TOE implements functionality to encrypt data and store it securely on the evaluated platform. The TOE uses the CRC Encryption and Shredding engine cryptographic module to generate AES-256 bit keys for file encryption (a.k.a., FEK).

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.

Test 1 - The evaluator created a test file under the external storage directory using the TOE application and the sample File Browser application that included the TOE library. The evaluator then ensured the test file was encrypted. The evaluator then tried to access the encrypted file through the platform's shell. The resulting file was visible, however its contents were not readable as the file was encrypted and could not be deciphered.

The TOE only claims to implement functionality to encrypt sensitive data as defined in the PP-module for File Encryption, and as a result, none of the other tests apply

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 4.7 of the User's Guide identifies each permission requested at installation time and explains why it is needed. The Guide has the following table.

Permissions	Explanation
android.permission.READ_PHONE_STATE	Phone status and identity access is requested so that the phone's ID can be used for identification purposes.
android.permission.QUERY_ALL_PACKAGES	This permission is requested to enable the management service to monitor application install/uninstall for third party app registration/removal.
android.permission.ACCESS_WIFI_STATE **android.permission.CHANGE_WIFI_STATE**	The ability to modify the Wi-Fi connection is requested so that the device's MAC address can be used for identification purposes. The MAC address may only be available from one of a few sources, so multiple Wi-Fi-related permissions are requested to make sure all of these sources are available if needed.
android.permission.INTERNET **android.permission.ACCESS_NETWORK_STATE**	Network access is requested for making network requests to the Trivalent licensing server, using the HTTPS.
android.permission.BLUETOOTH	Bluetooth permission is requested so that the device's Bluetooth adapter's name and address can be used for identification.
android.permission.REORDER_TASKS	The ability to reorder apps is used because the Management Service needs the ability to launch and display authentication prompts from the background.
android.permission.RECEIVE_BOOT_COMPLETED	The ability to run at startup is used so that the Management Service's authentication service can begin running on Boot.
android.permission.WAKE_LOCK	The Management Service uses this permission because time-based authentication methods need to remain accurate in their de-authentication triggers, even if a device's screen is off.
android.permission.BIND_DEVICE_ADMIN	Necessary for implementing a device administration component.
android.permission.FOREGROUND_SERVICE	Needed for Authentication task as well as status bar notifications
android.permission.WRITE_EXTERNAL_STORAGE	Needed to write externally accessible Logs in future versions.
android.permission.AURICFS_ADMIN	Added for future updates.

Additionally, the user guide also has a similar table for the File Browser application (section 5.4)

Permissions	Explanation
android.permission.MANAGE_EXTERNAL_STORAGE	Allows use of the ACTION_MANAGE_ALL_FILES_ACCESS_PERMISSION intent action to direct users to a system settings page where they can enable the following option for your app: Allow access to manage all files.
android.permission.WRITE_EXTERNAL_STORAGE **android.permission.READ_EXTERNAL_STORAGE**	Enable application to read and write files into the device shared drives.
android.permission.QUERY_ALL_PACKAGES **android.permission.USE_CREDENTIALS**	Allows communication with Management Service.
android.support.PARENT_ACTIVITY	Added make it easy for users to find their way back to the app's main screen on older Android versions.

This list matches the requirement claim of network, USB, SD card, and Bluetooth. There are other non-hardware permission in the list (e.g., android.permission.BIND_DEVICE_ADMIN).

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.

Test 1 – The evaluator extracted the Android Manifest file from the TOE application and verified the permissions. All hardware permissions were consistent with the claims in the ST.

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.

The TOE does not have any sensitive repositories. The evaluator reviewed the list of permissions required by the application under section 4.7 and 5.4 of the AGD and found no claims for sensitive information repositories which is consistent with the TOE's claims.

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.

Test 1 – The evaluator extracted the Android Manifest file from the TOE application and verified the permissions. All sensitive information repository permissions were consistent with the claims in the ST.

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 TOE restricts its network communication to checking for updates. The evaluator sniffed the network to capture traffic while the TOE was used to unlock the TOE service, encrypt a file, decrypt a file, lock the TOE service, and then check for updates. The evaluator examined the packet capture and found no traffic that could be attributed to the TOE other than checking for an update.

Test 2- The evaluator performed a port scan on the test platform while the TOE service was running. The evaluator did not find any open or responding ports and concluded that the TOE did not open any ports.

2.2.4 Protection of Data in Power Managed States (FE10:FDP_PM_EXT.1)

2.2.4.1 FE10:FDP_PM_EXT.1.1

TSS Assurance Activities: None Defined

Guidance Assurance Activities: None Defined

Testing Assurance Activities: None Defined

2.2.4.2 FE10:FDP_PM_EXT.1.2

TSS Assurance Activities: None Defined

Guidance Assurance Activities: None Defined

Testing Assurance Activities: None Defined

2.2.4.3 FE10:FDP_PM_EXT.1.3

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 state(s) that are supported by this capability. For each state, the evaluator ensures that the TSS contains a description of how the state is entered, and the actions of the TSF on entering the state, specifically addressing how multiple open resources (of each type) are protected, and how keying material associated with these resources is protected (if different from that described elsewhere). The TSF shall also describe how the state is exited, and how the requirements are met during this transition to an operational state.

The evaluator shall verify the TSS provides a description of what keys and key material are destroyed when entering any protected state.

KMD

The evaluator shall verify the KMD includes a description of the areas where keys and key material reside. The evaluator shall verify the KMD includes a key lifecycle that includes a description where key material reside, how the key material is used, and how the material is destroyed once a claimed power state is entered and that the documentation in the KMD follows FCS_CKM_EXT.4.1 for the destruction.

Section 6.2 of the ST explains the TOE has 3 states: Power-off, Data Locked, and Data Unlocked. In the Power-off state, the device is off and TOE is not available.

The TOE spends most of its time between the Data Lock and Data Unlock states. It is not possible to move directly from Power-off to Data Unlock; the TOE always starts in the Data Lock state on startup.

During the Power-off state, all keys that may be in use are removed from volatile storage automatically, regardless of whether this state was by user shutdown/restart or by power failure.

On startup, the device places the TOE into the Data Lock state. From the Data Lock state, the user must authenticate successfully to the CRC Encryption and Shredding Engine Cryptographic module to transition to the Data Unlock state. In the Data Unlock phase, the passphrase is hashed using SHA-384 with a salt (generated using the platform's DRBG) and used to load the Management Service's keystore. The DaR password is then used as input to PBKDF2 with 4096 iterations and HMAC-SHA-512 PRF along with a second salt value (also generated using the platform's DRBG).

The "Data Locked" state is when the screen is locked. In this state the TOE deletes the 256-bit HASH KEK and encrypted 256-bit Master KEK from volatile memory, leaving only the encrypted files. A screen can be screen locked, password locked, or data locked after a period of inactivity. The TOE requires the user to re-authenticate (i.e., provide the DaR password) to decrypt files following a mobile device entering the "Data Locked" state

The proprietary KMD (TSS) includes a key hierarchy diagram. The diagram supports the description in the ST and the identification of key destruction in the table for FE10:FCS_CKM_EXT.4

Component Guidance Assurance Activities: The evaluator shall check the Operational Guidance to determine that it describes the states that are supported by the TOE, and provides information related to the correct configuration of these modes and the TOE.

The evaluator shall validate that guidance documentation contains clear warnings and information on conditions in which the TOE may end up in a non-protected state. In that case it must contain mitigation instructions on what to do in such scenarios.

Section 4 of the AGD explains the different power states (Authenticated and Locked/Unauthenticated) supported by the TOE, specifically sections 4.3 through 4.5. As stated, only once authenticated can additional applications using the CRC Data IO Protection Plugin tart initializing their protection core. When not authenticated integrated applications will not start up. Additionally, section 5.1 repeats this information from the File Browser App's perspective as the application offers no functionality while the Management service is locked and further subsections under Section 5 File Browser Apps detail the functionality available to the application while the TOE service is unlocked. The TOE protects its configured files across power states, so there is no condition in which the TOE may end up in a non-protected state.

Component Testing Assurance Activities: The following tests must be performed by the evaluator for each supported State, type of resource, platform, and authorization factor:

Test 1: Following the Operational guidance, configure the Operational Environment and the TOE so that the lower power state of the platform is enabled and protected by the TOE. Open several resources (documented in the test report) that are protected. Invoke the lower power state. On resumption of normal power attempt to access a previously-opened protected resource, observe that an incorrect entry of the authorization factor(s) does not result in access to the system, and that correct entry of the authorization factor(s) does result in access to the resources.

Test 1 – While unlocked, the evaluator used the TOE to encrypt several files. The evaluator then transitioned the TOE to the lower power state, or data locked the TOE. The evaluator then attempted to resume the normal power state with an incorrect authorization factor (password) and observed the TOE failed to transition power states. While in its current data locked state, the evaluator attempted to access (decrypt) a resource using the sample File Browser application, however this process failed. The evaluator then used the correct authorization factor to transition to the normal power state and observed that the decryption process now succeeded.

2.2.5 Protection of Selected User Data (FE10:FDP_PRT_EXT.1)

2.2.5.1 FE10:FDP_PRT_EXT.1.1

TSS Assurance Activities: None Defined

Guidance Assurance Activities: None Defined

Testing Assurance Activities: None Defined

2.2.5.2 FE10:FDP_PRT_EXT.1.2

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 determine that it lists each type of resource that can be encrypted (e.g., file, directory) and what 'encrypted' means in terms of the resource (e.g., 'encrypting a directory' means that all of the files contained in the directory are encrypted, but the data in the directory itself (which are filenames and pointers to the files) are not encrypted).

The evaluator shall also confirm that the TSS describes how each type of resource listed is encrypted and decrypted by the TOE. The evaluator shall ensure that this description includes the case where an existing file or set of files is encrypted for the first time; a new file or set of files is created and encrypted; an existing file or set of files is re-encrypted (that is, it had been initially encrypted; it was decrypted (by the TOE) for use by the user, and is then subsequently re-encrypted); and corresponding decryption scenarios. If other scenarios exist due to product implementation/features, the evaluator shall ensure that those scenarios are covered in the TSS as well.

The evaluator shall examine the TSS to ensure there is a high-level description of how the FEK is protected.

The evaluator shall examine the TSS to ensure there is a description of how the FEK is protected.

The evaluator shall examine the TSS to ensure that it describes all temporary files/resources created or memory used during the decryption/encryption process and when those files/resources or memory is no longer needed.

The TSS shall describe how the TSF or TOE platform deletes the non-volatile memory (for example, files) and volatile memory locations after the TSF is done with its decryption/encryption operation.

Section 6.2 of the ST states the TOE encrypts each individual file separately. The encrypted files are stored in a hidden directory at the same directory level as the original files. Before encryption, the TOE splits a file into chunks to more easily access specific parts within an encrypted file. Using the CRC Encryption and Shredding Engine Cryptographic Module's API, each chunk is then encrypted separately. The original file is replaced with a directory structure of different files after chunking and encryption. The directory structure of the encrypted file includes a metadata file that describes the chunking structure, a hidden folder for every chunk that includes a header file, and the encrypted file chunks split into encrypted pieces. The TOE implements functionality to ensure that sensitive data is destroyed in volatile and non-volatile memory upon completion of either a decrypt or encrypt operation of the sensitive files. The Cyber Reliant Mobile Data Defender for Android SDK returns the plaintext data returned by the API to the application for processing. The Cyber Reliant Mobile Data Defender for Android SDK clears all internal buffers of the data. The CRC Encryption and Shredding Engine Cryptographic Module also has routines to destroy keys stored inside. The TOE does not create any temporary resources.

Each chunk is decrypted separately (using the CRC Encryption and Shredding Engine Cryptographic module's decryption API). This allows much faster access to read/write encrypted data to the encrypted file (e.g. random access files). For example, if data is added to the middle of an encrypted file that has not been chunked, the entire file needs to be decrypted, and the data needs to be inserted before the file is re-encrypted. In the same scenario, if the encrypted file was chunked, then the first half of the chunks can be skipped before reaching the point at which the data needs to be encrypted and inserted. The Cyber Reliant Mobile Data Defender for Android SDK chunks the data set on 10MB boundaries, so if only a subset of the data is needed, the system will only encrypt the 10MB chunk that contains the desired data set. If the data in the file wishes to be changed, then the whole file must be re-encrypted. Decrypted pieces are retained for caching purposes for up to 30 seconds, before they are purged and the memory is wiped. Each file has a unique FEK and IV, which is used to encrypt/decrypt each chunk. The wrapped FEK and IV are stored in a hidden directory that resides in the same directory as the encrypted file.

The TOE programmatically destroys keys in volatile memory per the table in FE10:FCS_CKM_EXT.4 (such as keys stored in RAM and used by the TOE) by calling Android's Arrays.fill method in order to zero out the key array.

The CRC Encryption and Shredding Engine Cryptographic Module also has its own zeroization. The module_destroy API zeroes the FEK and FEKEK from volatile memory the table in FE10:FCS_CKM_EXT.4.

Component Guidance Assurance Activities: If the TOE creates temporary objects and these objects can be protected through administrative measures (e.g., the TOE creates temporary files in a designated directory that can be protected through configuration of its access control permissions), then the evaluator shall check the Operational Guidance to ensure that these measures are described.

If there are special measures necessary to configure the method by which the file or set of files are encrypted (e.g., choice of algorithm used, key size, etc.), then those instructions shall be included in the Operational Guidance and verified by the evaluator. In these cases, the evaluator checks to ensure that all non-TOE products used to satisfy the requirements of the ST that are described in the Operational Guidance are consistent with those listed in the ST, and those tested by the evaluation activities of this PP-Module.

There are no additional guidance evaluation activities for FDP_PRT_EXT.1.2.

There are no special configuration steps for the encryption method for files. Thus, there is no need for the guidance to contain instructions.

Component Testing Assurance Activities: The evaluator shall also perform the following tests. All instructions for configuring the TOE and each of the environments must be included in the Operational Guidance and used to establish the test configuration.

For each resource and decryption/encryption scenario listed in the TSS, the evaluator shall ensure that the TSF is able to successfully encrypt and decrypt the resource using the following methodology:

Monitor the temporary resources being created (if any) and deleted by the TSF-the tools used to perform the monitoring (e.g., procmon for a Windows system) shall be identified in the test report. The evaluator shall ensure that these resources are consistent with those identified in the TSS, and that they are protected as specified in the Operational Guidance and are deleted when the decryption/encryption operation is completed.

Test 1: This test only applies for application provided functionality.

1. Using a file editor, create and save a text file that is encrypted per the evaluation configured encryption policy. The contents of the file will be limited to a known text pattern to ensure that the text pattern will be present in all encryption/decryption operations performed by the TOE.

2. Exit the file editor so that the file (including its known text pattern) has 'completed the decryption/encryption operation' and process memory containing the known text pattern is released.

3. The evaluator will take a dump of volatile memory and search the retrieved dump for the known pattern. The test fails if the known plaintext pattern is found in the memory dump.

4. The evaluator will search the underlying non-volatile storage for the known pattern. The test fails if the known plaintext pattern is found in the search.

Test 1 – The evaluator used a file editor to create and save a text file. Once created, the evaluator configured the TOE to encrypt the text file, ensuring both encryption and decryption operations could complete, however leaving the file in a protected encrypted state. The evaluator then searched non-volatile memory for the plaintext data and could not find any evidence of the data. The evaluator repeated the search in volatile memory and also did not find any evidence of the data.

2.2.6 Destruction of Plaintext Data (FE10:FDP_PRT_EXT.2)

2.2.6.1 FE10:FDP_PRT_EXT.2.1

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 all temporary file (or set of files) that are created in the filesystem of the host during the decryption/encryption process, and that the TSS describes how these files are deleted after the TSF is done with its decryption/encryption operation. Note that if other objects/resources are created on the host that are 1) persistent and 2) visible to other processes (users) on that host that are not filesystem objects, those objects shall be identified and described in the TSS as well.

Section 6.2 of the ST states the TOE does not create any temporary resources.

Component Guidance Assurance Activities: None Defined

Component Testing Assurance Activities: Test 1: If the TSS creates temporary files/resources during file decryption/encryption, the evaluator shall perform the following tests to verify that the temporary files/resources are destroyed. If the product supported shared files per FIA_FCT_EXT.2, this test must be repeated with a shared file. The evaluator shall use a tool (e.g., procmon for a Windows system) that is capable of monitoring the creation and deletion of files during the decryption/encryption process is performed. A tool that can search the contents of the hard drive (e.g., winhex) will also be needed. The tools used to perform the monitoring shall be identified in the test report.

(Creating an encrypted document)

Open an editing application.

Create a special string inside the document. The string could be 5-10 words. It is recommended to remove the spaces. This will create a one page document.

Start the file monitoring tool.

Save and close the file.

Encrypt the file using the TOE (if the TOE does not encrypt automatically for the user).

Analysis Steps

If needed, exit/close the TOE.

Stop the file monitoring tool. View the results. Identify any temporary files that were created during the encryption process. Examine to see if the temporary files were destroyed when the TOE closed.

If temporary files remain, these temporary files should be examined to ensure that no plaintext data remains. If plaintext data is found in these files, the test fails.

Search the contents of the hard drive (using the second tool) for the plaintext string used above. (The search should be performed using both ASCII and Unicode formats.)

If the string is found, this means that plaintext from the test fails.

(Creating, encrypting a blank document and then adding text):

Encrypt a blank document using the tool.

Create a special string inside the document. The string could be 5-10 words. It is recommended to remove the spaces. This will create a one page document.

Start the file monitoring tool.

Save and close the file.

Perform the 'Analysis Steps' listed above.

Test 1 – Not applicable, the TSS does not cerate temporary files/resources during file decryption/encryption nor does the product support shared files per FIA_FCS_EXT.2.

2.3 Identification and Authentication (FIA)

2.3.1 Subject Authorization (FE10:FIA_AUT_EXT.1)

2.3.1.1 FE10:FIA_AUT_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 how user authentication is performed. The evaluator shall verify that the authorization methods listed in the TSS are specified and included in the requirements in the ST.

Requirement met by the TOE:

The evaluator shall first examine the TSS to ensure that the authorization factors specified in the ST are described. For password-based factors the examination of the TSS section is performed as part of FCS_CKM_EXT.6 Evaluation Activities.

Additionally in this case, the evaluator shall verify that the operational guidance discusses the characteristics of external authorization factors (e.g., how the authorization factor must be generated; format(s) or standards that the authorization factor must meet) that are able to be used by the TOE. If other authorization factors are specified, then for each factor, the TSS specifies how the factors are input into the TOE.

Requirement met by the platform:

The evaluator shall examine the TSS to ensure a description is included for how the TOE is invoking the platform functionality and how it is getting an authorization value that has appropriate entropy.

Section 6.3 of the ST states that the TOE provides a DaR password-based authorization factor in order to authenticate to the Cyber Reliant Mobile Data Defender for Android SDK service.

There is only a single, password/passphrase-based authorization method offered by the TOE. Section 4.1 of the User Guide explains the characteristics of the password – must be 6 characters and must contain characters, upper, lower and special characters. It also explains, for a user to create a strong passphrase, a passphrase with at least one uppercase letter, lowercase letter, special character, and number is recommended. Longer length passphrases provide increased security strength over shorter ones. Acceptable password strength should reflect the sensitivity of the data being encrypted. Higher data sensitivity should have increased password strength.

Component Guidance Assurance Activities: The evaluator shall verify that the AGD guidance includes instructions for all of the authorization factors. The AGD will discuss the characteristics of external authorization factors (e.g., how the authorization factor is generated; format(s) or standards that the authorization factor must meet, configuration of the TPM device used) that are able to be used by the TOE.

There is only a single, password/passphrase-based authorization method offered by the TOE. The “Installation” section of the User Guide describes the process used to initially configure the TOE, including initializing the user’s password/passphrase. The User Guide indicates that once configured, it is necessary for the TOE’s service to be started, before encryption and decryption features are available. Section 4.2 entitled “Start Service” in User Guide describes that it is necessary to start the service and that the user must authenticate prior to the service being able to encrypt and decrypt files.

Component Testing Assurance Activities: The evaluator shall ensure that authorization using each selected method is tested during the course of the evaluation, setting up the method as described in the operational

guidance and ensuring that authorization is successful and that failure to provide an authorization factor results in denial to access to plaintext data.

[conditional]: If there is more than one authorization factor, ensure that failure to supply a required authorization factor does not result in access to the decrypted plaintext data.

Test 1 – This has been tested as a side effect of other tests. The authorization factor is required to use the TOE and the correctness of the authorization factor is tested under FE10:FCS_CKM_EXT.6. Correctness of the access control of plaintext data based on this authorization factor is tested under FD10:FDP_PM_EXT.1

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 ST states the TOE has a default password but provides only enough functionality to set new credentials when first installed. The encryption services are greyed out until configuration file is created.

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 – The evaluator installed the TOE on the test platform and ran the application for the first time. Before being able to start the TOE service, the evaluator was forced to change the default password.

Test 2 – The evaluator has shown that the TOE does not allow credentials to be cleared in test 1 and therefore the test is met.

Test 3 – The evaluator installed the TEO on the test platform and ran the application for the first time. After changing the default password, the evaluator attempted to use the default password to start the TOE service, however this failed as the password was incorrect.

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.

Test 1 – The evaluator installed and ran the TOE application before inspecting the filesystem. The evaluator ran the prescribed command in the TOE's data directory which did not return any files. The evaluator did not find any world-writable files in the locations where the application could create data.

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's evaluated Android platform automatically uses /data/data/package/shared_prefs/ to store configuration options and settings. The evaluated TOE is distributed with an embedded configuration which includes a list of apps to encypt, shredding configurations, IP address of the TCM (N/A), password requirements and settings, and authentication lockout settings, however these are not configurable post-installation. The Time-outs, Lock-outs, and SALTs for PBKDFv2 are stored in private files in the file directory.

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: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.

Test 1 – the TOE invokes the platform mechanisms for storing configuration options. The evaluator installed the application and recorded the XML values under the shared_prefs directory. The evaluator then changed the password on the TOE and tested it by making an incorrect authorization attempt. The evaluator observed that NUM_AUTH_ATTEMPTS changed from 0 to 1.

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.

Section 6.4 of ST claims that the TOE includes four management functions. Each is described in the User Guide as identified below:

1. Change password/passphrase authentication credential – Section 4.1 explains the change password screen
2. Lock/unlock the TOE management service – Section 4.5 explains the Utility screen that shows if the management service has been locked/unlocked
3. Check for updates – Section 4.2 shows the check version option and explains it checks the app version.
4. Configure encryption/decryption file status - Section 4.5 explains the Utility screen that shows if the management service has been locked/unlocked which permits encryption. Section 5 describes the file browser app that performs file operations once registered.

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.

Test 1 – The TOE claims 4 management functions. The evaluator configured and tested each separately and observed the expected behavior in each case

Change password/passphrase authentication credentials – See FE10:FCS_CKM_EXT.6 where this is tested extensively.

Lock/unlock the TOE management service – The evaluator showed that the TOE can transition from the unlocked to locked state and from the locked to unlocked state (which requires a password)

Check for updates – See ASPP14:FPT_TUD_EXT.1 where this is tested.

Configure encryption/decryption file status – The evaluator verified the TOE service was unlocked and then used the TOE security library to configure encryption/decryption status. Further testing showing the file is actually encrypted/decrypted is performed under ASPP14:FDP_DAR_EXT.1

2.4.4 Specification of File Encryption Management Functions (FE10:FMT_SMF.1(2))

2.4.4.1 FE10:FMT_SMF.1.1(2)

TSS Assurance Activities: None Defined

Guidance Assurance Activities: None Defined

Testing Assurance Activities: None Defined

Component TSS Assurance Activities: Conditional Activities: The evaluator shall examine the TSS to ensure that it describes the sequence of activities that take place from an implementation perspective when this activity is performed (for example, how it determines which resources are associated with the KEK, the decryption and re-encryption process), and ensure that the KEK and FEK are not exposed during this change.

Cryptographic Configuration: None for this requirement.

Section 6.4 of the ST identifies the follow description for the claimed management function, “change password/passphrase authentication credential”:

- Upon a password/passphrase change, the TOE will load the double-wrapped FEKEK, rederive the old PBKDF-derived key to singularly unwrap the double-wrapped FEKEK, and then use the new password to rewrap this and store this into the embedded WolfCrypt keystore. During this process, the FEKEK is never

unwrapped with the TOE management service's RSA keypair, meaning the cleartext FEKEK and any resulting FEK plaintext values are never exposed.

Component Guidance Assurance Activities: Conditional Activities: The evaluator shall examine the Operational Guidance to ensure that it describes how the password/passphrase-based authorization factor is to be changed.

Cryptographic Configuration: The evaluator shall determine from the TSS for other requirements (FCS_*, FDP_PRT_EXT, FIA_AUT_EXT) what portions of the cryptographic functionality are configurable. The evaluator shall then review the AGD documentation to determine that there are instructions for manipulating all of the claimed mechanisms.

Section 4.1 of the User Guide explains the password change screen and requirements. No other cryptographic functionality is configurable so no guidance is necessary.

Component Testing Assurance Activities: Cryptographic Erase: If the TOE uses stored FEKS or KEKs, the evaluator shall examine the key chain to determine that the keys destroyed by a cryptographic erase will result in the data becoming unrecoverable. Testing for this activity is performed for other components in this PP-Module.

Test 1 – The Toe only claims the ability to change password/passphrase authentication credentials under this SFR which is tested under ASPP14:FMT_SMF.1. The TOE does not claim perform cryptographic erase and therefore it is not possible to cryptographic erase keys needed derive the FEKs or KEKs.

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 of the ST states the TOE does not transmit any PII over a network.

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.

Test - Not applicable – the TOE does not transmit any PII information over a network

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.

Section 6.6 of the ST states the application is compiled with "-v -DBUILD_JNI -DANDROID -DCyber -O2 -fstack-protector-all -fexceptions" in order to enable ASLR and stack-based buffer overflow protection.

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.

Test 1 – The evaluator performed a dynamic analysis to determine that the TOE uses address space layout randomization. The evaluator ran the application on a test platform and then grabbed the process maps for the running TOE application. The evaluator then uninstalled the application, rebooted the device, and reinstalled the TOE application. Whie running the evaluator grabbed a second memory mapping. The evaluator compared the two outputs and found that they were sufficiently randomized.

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.

Test 1 – The evlautor decompiled the TOE application and the TOE security library. The evaluator then performed a search for mmap and mprotect. The evaluator found no instances of mprotect or mmap, specifically with the parameters PROT_WRITE and PROT_EXEC, being invoked.

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.

Test 1 – The evaluator ran the TOE application on an Android platform for all of testing which does not have security features that can be disabled and therefore this requirement is met and no evaluation activity is required.

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.

Test 1 – The evaluator ran the application and determined where it could write files. The evaluator determined that the TOE could write files to any area where a normal application can write files (external storage directory). In these scenarios, the user is choosing the location. The evaluator also searched the data directory for the application and found no executables located under that location.

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.

Test 1 – The TOE by default is an Android (Java application) and a TOE security library. The evaluator inventoried all of the native executables within the TOE boundary and verified that stack-based buffer overflow protection was present in each one.

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 ST references the User Guide for a list of the platform APIs that are used by the TOE. The evaluator compared the list in the User Guide with the java docs online to ensure each method, object and exception was documented in the public. The evaluator relied on the https://docs.oracle.com/javase/7/docs website for the Java API definitions.

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.

Test 1 – The evaluator compared the list of APIs in the TSS with a list of supported APIs through online documentation. The evaluator found that the various Android APIs matched online documentation provided by Android and that the various java.security and javax.crypto APIs matched online documentation provided by Java.

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 ST states the TOE labels its software using a unique release number that can be queried by users.

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.

Test 1 – The evaluator installed the application and used the TOE's option to check version. The TOE reported back a unique application version. The TOE does not claim SWID tags.

2.6.4 Protection of Keys and Key Material (FE10:FPT_KYP_EXT.1)

2.6.4.1 FE10:FPT_KYP_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 the TSS for a high level description of the method(s) used to protect keys stored in non-volatile memory.

KMD

The evaluator shall verify the KMD to ensure it describes the storage location of all keys and the protection of all keys stored in non-volatile memory. The description of the key chain shall be reviewed to ensure FCS_COP.1(5) is followed for the storage of wrapped or encrypted keys in non-volatile memory and plaintext keys in non-volatile memory meet one of the criteria for storage.

Section 6.6 of the ST states the TOE stores plaintext keys in non-volatile memory by relying on the Android platform keystores. The TOE uses the Management Service's WolfCrypt keystore to store the double wrapped FEKEK. The double-wrapped FEKEK is unwrapped using AES key wrap and the PBKDF derived value from the user's password. The TOE uses the Android keystore to store all RSA public/private keys, which are used to unwrap the single-wrapped FEKEK. In order to pass the plaintext FEKEK between the Management service and the registered application, the plaintext FEKEK is wrapped using an RSA public/private keypair stored in the application's platform keystore. The TOE wraps the file encryption keys (FEK, used for encrypting files) with another AES key, and then stores the wrapped FEK on the evaluated device's flash memory. The TOE utilizes the evaluated Android platform's AES key wrap and RSA OAEP key wrap functions to protect keys.

The evaluator reviewed the key chain in FE10:FCS_COP.1(5) and all keys are included in the description.

Component Guidance Assurance Activities: None Defined

Component Testing Assurance Activities: None Defined

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

2.6.5.1 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.

Test 1 – the evaluator installed and queried the dynamic libraries included in the TOE's installation directory. The evaluator found libCRCWolf.so, libdataprotectCRC.so between the TOE management service and the sample application which leverages the TOE security library. libdataprotectCRC.so is the name of the TOE security library and libCRCWolf.so is the vendor's version of WolfCrypt used for cryptographic operations.

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

2.6.6.1 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 2.1 of the AGD states CRC will email all customers using outdated software as soon as the new version release is available. CRC will also list the current software version numbers on their website (www.cyberreliant.com). Customers are able to request the newest updated versions by emailing their request to: contact@cyberreliant.com . Upon a customer request CRC sends new software release version and Release Notes to the customer, usually via email, containing a download link to the customer installable software version .apk file hosted by CRC's ShareFile portal.

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.

Test 1 – The evaluator used the app to check its version. The app reported it was up to date.

2.6.6.2 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 2.1.1 of the User Guide denies how a user can check their current installation version and the last full release version using the Management Service Main Window.

Section 4.2 of the User Guide identifies the check version button the management service. The explanation states the button performs a version check of Cyber Reliant Protect to determine if the installed version 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.

Test 1 – The evaluator queried the version for the application under ASPP14:FDP_IDV_EXT.1.1-t1. The resulting version was found to match the documented and installed version.

2.6.6.3 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.

Test 1 – The evaluator installed the TOE and used the test platform's shell to inventory the directory where the TOE executables are saved. The evaluator then saved a copy of a timestamp and md5 hash of each file in the directory. The evaluator then used the TOE to encrypt a file, decrypt a file, change the password, and restarted the

TOE service. Afterward, the evaluator returned to the test platform's shell and obtained a new timestamp and MD5 hash for each executable in the installation directory. The evaluator verified that these were identical

2.6.6.4 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 states the TOE's software is digitally signed by Cyber Reliant. Candidate updates are obtained directly from the vendor through email or other agreed upon delivery mechanism.

Guidance Assurance Activities: None Defined

Testing Assurance Activities: None Defined

2.6.6.5 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 that the application is distributed as APK files that are digitally signed by Cyber Reliant. Cyber Reliant delivers any installation material via email or other agreed upon method.

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.

Test 1 – The TOE claims to be distributed as an additional software package to the platform OS. As a result, the 'with the platform' test does not apply and the evaluator selected and performed the tests under FPT_TUD_EXT.2

Component TSS Assurance Activities: None Defined

Component Guidance Assurance Activities: None Defined

Component Testing Assurance Activities: None Defined

2.6.7 Integrity for Installation and Update - per TD0628 (ASPP14:FPT_TUD_EXT.2)

2.6.7.1 ASPP14:FPT_TUD_EXT.2.1

TSS Assurance Activities: None Defined

Guidance Assurance Activities: None Defined

Testing Assurance Activities: FPT_TUD_EXT.2.1: If a container image is claimed the evaluator shall verify that application updates are distributed as container images.

If the format of the platform-supported package manager is claimed, the evaluator shall verify that application updates are distributed in the correct format. 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.

Test 1 – The TOE claims to be distributed using the format (APK) of the platform (Android)-supported package manager. The vendor provided the application in the form of an APK file for Android and this was used for the entirety of testing.

2.6.7.2 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. (TD0664 applied)

Test 1 – This requirement is inherently met as the platform forces applications to write data within the application working directory.

2.6.7.3 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 states the TOE's software is digitally signed by Cyber Reliant.

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.

Section 6.7 of the ST states the TOE does not transmit any sensitive data between itself and other products.

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 reexamined the packet capture from FDP_NET_EXT.1.1-t1 as it contained network traffic of the test platform while the TOE was being exercised for its various operations. The TOE does not transmit any sensitive data which is consistent with the evidence observed.

Test 2 – The evaluator reexamined the packet capture from FDP_NET_EXT.1.1-t1. Although there is no transmitted sensitive data, the evaluator analyzed the packet capture and observed no plaintext protocols that involved the TOE application. The evaluator observed traffic to the TOE's update server, however this was not considered sensitive data.

Test 3 – The TOE does not transmit any credentials. The evaluator searched the packet capture from FDP_NET_EXT.1.1_t1 for the password that was configured at the time, however could not find any evidence of it in plaintext in the packet capture.

3. Protection Profile SAR Assurance Activities

The following sections address assurance activities specifically defined in the ASPP14/FE10 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.

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.

No user configuration is available for the cryptographic features of the TOE.

Section 2 of Admin Guide discusses the topic of a security update procedure. Updates to the TOE (i.e., Trivalent Management Service) are provided via an APK which must be obtained from Trivalent and manually installed. Section 4.2 of the User Guide also explains how to check the version and contract Trivalent if an update is available (email address provided).

Section 1.2 of the User Guide describes the evaluated product. The product does not contain extra functionality as it is a focused application.

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.

Section 1.2 of the User Guide states the TOE is an Android application. This matches the ST claims. Additionally, this section also states "See Section 1.3 of the Security Target for a list of mobile devices included in the scope of evaluation for the Cyber Reliant Defender"

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 User Guide document identifies a specific version of the TOE for which it is relevant (section 1.1). This version matches 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 User Guide document identifies a specific version of the TOE for which it is relevant (section 1.1). This version matches the ST. The User Guide also states the TOE is an Android application (1.2).

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 states if a security vulnerability was found by a user, then the user must report it to Cyber Reliant via email at support@cyberreliant.com. In the case that the vulnerability impacts the Cyber Reliant Mobile Data Defender for Android SDK: Cyber Reliant will deliver updated API Code, as well as additional developer documentation outlining any potential changes in the implementation, within a maximum of 30 days from time of vulnerability disclosure. In order to deliver the final resolution to the end-user, the partner developer or customer will need to implement the updated code into their target applications. The time for final delivery will be dependent on their ability to update the end-user application, and to distribute to users via Mobile Device Management Service, application store, or other delivery mechanism.

In case the vulnerability directly relates to the Management Service APK: Cyber Reliant shall deliver, via email or other agreed upon method, an updated application with security vulnerabilities addressed. The delivered software shall be accompanied by documentation outlining changes to the overall service, as well as compatible versions of the API. Once delivered to the customer or partner, the application can be delivered to end-users via Internal MDM instances, Internal 'App Stores' or other agreed upon methodologies

3.4 Tests (ATE)

3.4.1 Independent Testing - Conformance (ATE_IND.1)

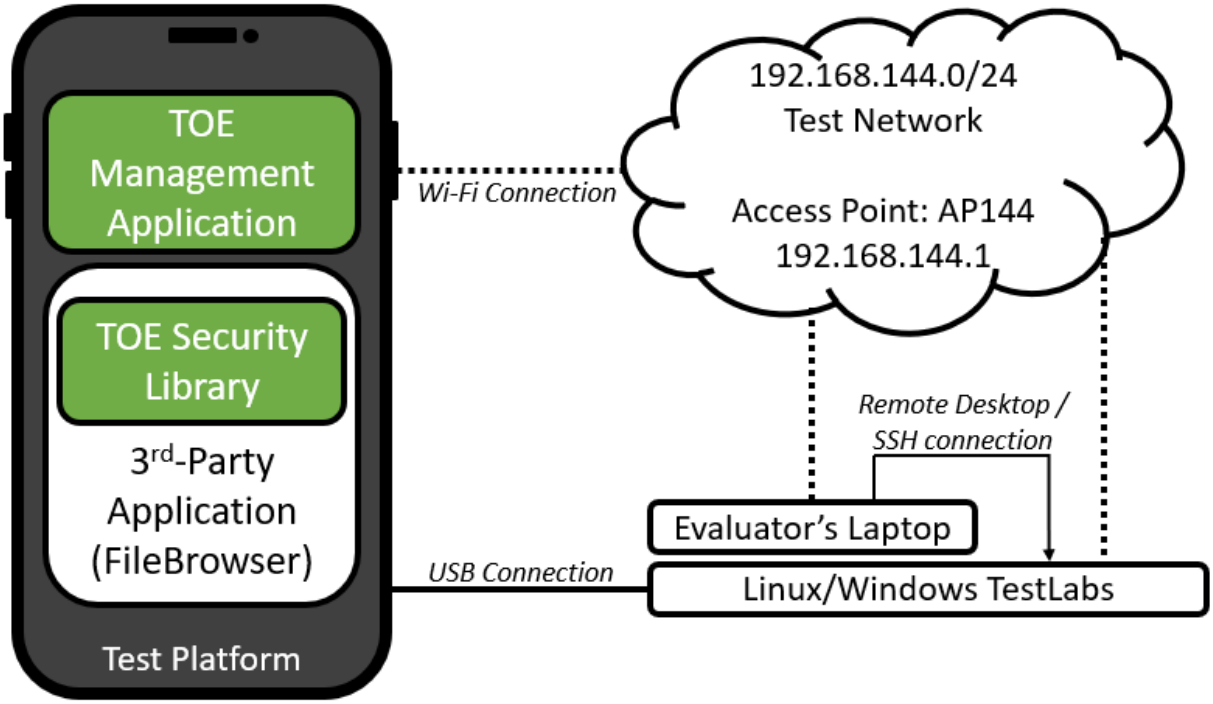
Assurance Activities: The evaluator shall prepare a test plan and report documenting the testing aspects of the system, including any application crashes during testing. The evaluator shall determine the root cause of any application crashes and include that information in the report. The test plan covers 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 evaluator used the following test configuration with the boundaries of the TOE highlighted in green:



The TOE was fully tested on a Samsung S20, Samsung Tab Active 3, and a Panasonic Toughbook FZ-N1 all on Android 11. The TOE can be installed on any evaluated Android 11 device as identified in the Equivalency Section of this report. The apk and corresponding binaries are the same among all devices.

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.

For Windows, Linux, macOS and Solaris: 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 vulnerability analysis is in the proprietary Detailed Test Report (DTR) prepared by the evaluator. The vulnerability analysis includes a public search for vulnerabilities. The evaluator searched the National Vulnerability Database (NVD) from the NIST website and the Vulnerability Notes Database (VND) from the CERT Knowledgebase on 03/07/2023 using the following terms: "Cyber Reliant Corp", "Cyber Reliant", "CRC", "Trivalent", "Mobile Data Defender", "WolfCrypt".

The TOE does not run on Windows, Linux, macOS, or Solaris so a virus scan is not applicable.