Trivalent Data at Rest (DaR) Service (Inside) (ASPP11/FEEP10) Security Target

Version 0.6
12/21/15

Prepared for:

Trivalent

175 Admiral Cochrane Drive
Suite 404
Annapolis. MD 21401
CyberReliant Corporation dba Trivalent

Prepared By:

www.gossamersec.com

LIST OF TABLES

1. Security Target Introduction

This section identifies the Security Target (ST) and Target of Evaluation (TOE) identification, ST conventions, ST conformance claims, and the ST organization. The TOE is Data at Rest (DaR) Service (128-bit) provided by Trivalent. The TOE is being evaluated as a software application with file encryption capabilities.

The Security Target contains the following additional sections:

- Conformance Claims (Section 2)
- Security Objectives (Section 3)
- Extended Components Definition (Section 4)
- Security Requirements (Section 5)
- TOE Summary Specification (Section 6)

Conventions

The following conventions have been applied in this document:

- Security Functional Requirements – Part 2 of the CC defines the approved set of operations that may be applied to functional requirements: iteration, assignment, selection, and refinement.
    - Iteration: allows a component to be used more than once with varying operations. In the ST, iteration is indicated by a letter placed at the end of the component. For example FDP_ACC.1a and FDP_ACC.1b indicate that the ST includes two iterations of the FDP_ACC.1 requirement, a and b.
    - Assignment: allows the specification of an identified parameter. Assignments are indicated using bold and are surrounded by brackets (e.g., [**assignment**]). Note that an assignment within a selection would be identified in italics and with embedded bold brackets (e.g., [***[selected-assignment]***]).
    - Selection: allows the specification of one or more elements from a list. Selections are indicated using bold italics and are surrounded by brackets (e.g., [***selection***]).
    - Refinement: allows the addition of details. Refinements are indicated using bold, for additions, and strike-through, for deletions (e.g., "… **all** objects …" or "… ~~some~~ **big** things …").
- The NDPP uses an additional convention – the 'case' – which defines parts of an SFR that apply only when corresponding selections are made or some other identified conditions exist. Only the applicable cases are identified in this ST and they are identified using **bold** text.
- Other sections of the ST – Other sections of the ST use bolding to highlight text of special interest, such as captions.

1.1 Security Target Reference

ST Title – Trivalent Data at Rest (DaR) Service (Inside) (ASPP11/FEEP10) Security Target

ST Version – Version 0.6

ST Date – 12/21/2015

1.2 TOE Reference

TOE Identification – Trivalent Data at Rest (DaR) Service (Inside) Version 1.0.0 (Version Code 2)

TOE Developer – Trivalent

Evaluation Sponsor – Trivalent

1.3 TOE Overview

The Target of Evaluation (TOE) is Trivalent's Data at Rest (DaR) Service (Inside) Version 1.0.0 (Version Code 2) software application package residing on evaluated mobile devices running Android 4.4 on the Snapdragon 800-family processors. The TOE is a software solution providing the capability to handle file encryption on mobile devices. The TOE is tested on a Samsung Galaxy Note 3. Below are the current evaluated platforms:

- Samsung Note 3 and NotePRO tablet
- Samsung Galaxy S5 & Note 10.1 2014 Edition
- LG G3 Smartphone
- Samsung Galaxy Note 4, Note Edge, Galaxy Alpha, Galaxy Tab S 8.4 LTE & 10.5 LTE
- Samsung Galaxy S5 with KNOX 2
- Samsung Galaxy Note 4, Note Edge, Alpha, Galaxy Tab S 8.4 LTE & 10.5 LTE, Galaxy Tab Active with KNOX 2
- Samsung Galaxy Note Edge

Any of the above products utilize the Snapdragon 800 family processor and are appropriate for use with the TOE.

1.4 TOE Description

Trivalent's DaR Service (Inside) provides file level encryption through an APK and a library implementation. DaR Management Service (128-bit) contains both Java and native (c/c++) interfaces in order to support majority of android application storage requirements. The same implementation and functionality for both java and c/c++ are provided by the TOE.

The DaR's Management Service implementation is a straight Java DaR Service (Inside) APK and a library to be included into one's mobile application (and then one can use the API). The DaR Management Service (Inside) runs in the background and uses both Android and BouncyCastle keystores to provide the File Encryption Key Encryption Key (FEKEK) to each of the applications. The DaR Service (Inside) also uses the Android keystore to store an RSA key pair used by the Management service, and a per application android keystore to store each application's RSA keypair to wrap the AES-wrapped file encryption key encryption key. The Management Service handles necessary authentication and key management. The file level encryption suite is an API designed to support the use of specialized file level encryption for Android applications. Encryption is provided by the SPX Core (Security First, Secure Parser Library).

1.4.1 TOE Architecture

The TOE is software installed on an evaluated mobile device running Android 4.4. The TOE software is installed as a Management Service as well as a TSF interface library that is compiled into other applications. References to applications noted in this Security Target are regarded as applications that are compiled with the TSF interface API library. The Management Service is responsible for handling the File Encryption Key Encryption Keys (FEKEKs) necessary to unwrap the FEK. The Management Service obtains the DaR password from the user and double wraps the FEKEK by using RSA-2048 first and then wrapping it again using AES-256.

The TOE's interface library is compiled into another application's package. The library allows the other application to invoke the TOE's services. This library allows the application to call the TOE's file encryption services. The application must register itself with the DaR Management Service. Applications registered to the TOE have a unique RSA public/private keypair, so applications pass their RSA public keys to the DaR Management Service along with a certificate fingerprint (which is what the application uses as the password to the application's BouncyCastle key store). Android's keystore protects keys by storing them in a container with limited access to the keys through Android's keystore API. The TOE allows only a single user at a time.

The TOE stores the double wrapped FEKEKs in the Management Service's BouncyCastle keystore and the single wrapped FEKEKs in the Application's specific BouncyCastle keystore. The keys are protected by requiring a password to load both the Management Service and Application's BouncyCastle keystore. In order for other applications to access its FEK, the application must use the TOE's interface library API in order to request the Management Service's functions. The management service uses the application's public key to wrap the FEKEK (via RSA-OAEP) so that it can be passed to the Application by placing the single wrapped FEKEK into the Application's BouncyCastle keystore. The wrapped FEKEKs in each application's BouncyCastle keystore are ephemeral1. The Management Service has a configurable timer in which all BouncyCastle keystores will be wiped once the timer expires.

The TOE utilizes Security First's Secure Parser (SPX Core) for cryptographic services. The TOE uses the SPX Core for generating 128 bit AES keys.

During evaluation testing, Gossamer tested the Trivalent DaR on the Samsung Galaxy Note 3 running Android 4.4.

1.4.1.1 Physical Boundaries

The physical boundary of the TOE is the physical perimeter of the evaluated device (Android 4.4) on which the TOE resides.

1.4.1.2 Logical Boundaries

This section summarizes the security functions provided by Trivalent DaR Service (Inside):

- Cryptographic support
- User data protection
- Identification and authentication
- Security management
- Protection of the TSF
- Trusted path/channels

1.4.1.2.1 Cryptographic support

The evaluated platform runs Android 4.4 operating system. Android's APIs allow generation of keys through KeyGenerator, and random numbers are generated using SecureRandom. Keys are used to protect data belonging to the applications that use the TOE.

The TOE uses Security First's SPX Core (Security First, Secure Parser Library) for cryptographic algorithms. The SPX Core supports encryption via AES and random number generation via an SP 800-90 AES-128 CTR DRBG. The TOE uses the platform's cryptographic API to perform AES key wrapping and keyed hashing via HMAC.

1.4.1.2.2 User data protection

The TOE protects user data by providing encryption services for applications to encrypt their data. The TOE allows encryption of data using AES-128 bit keys.

1.4.1.2.3 Identification and authentication

The TOE authenticates applications by requiring a PIN/passphrase to unlock the application's file encryption key. A wrong password results in the unsuccessful loading of the application's BouncyCastle keystore. Without the correct keystore, the application cannot load the keys necessary for file encryption/decryption.

[1] RSA-OAEP key transport scheme allows the single wrapped FEKEK to be different each time it is encrypted

1.4.1.2.4 Security management

The TOE's services/options are inaccessible until a configuration has been created. The TOE does not allow invocation of its services without configuration of the TOE's settings upon first start up. The TOE allows the changing of passwords for management purposes.

1.4.1.2.5 Protection of the TSF

The TOE relies on the physical boundary of the evaluated platform as well as the Android operating system for the protection of the TOE's application components.

The TOE checks for updates by selecting the check current version option on its menu. If an update is needed, Trivalent shall deliver, via email or other agreed upon method, an updated application. The TOE's software is digitally signed by Trivalent. Each update is accompanied by documentation outlining changes to the overall service, as well as compatible versions of the Trivalent API.

The native Android cryptographic library, which provides the TOE's cryptographic services, have built-in self-tests that are run at power-up to ensure that the algorithms are correct. If any self-tests fail, the TOE will not be able to perform its cryptographic services.

1.4.1.2.6 Trusted path/channels

The TOE does not transmit any data between itself and another product. All of the data managed by the TOE resides on the evaluated platform (Android 4.4).

1.4.2 TOE Documentation

Trivalent offers documents that describe the operation and maintenance for the TOE. The following list of documents was examined as part of the evaluation.

[OM]	Data at Rest (DaR) Data Encryption/Security Solution, Trivalent DaR Service Operations & Maintenance Manual, version 1.0.3, November 23, 2015.

2. Conformance Claims

This TOE is conformant to the following CC specifications:

- Common Criteria for Information Technology Security Evaluation Part 2: Security functional components, Version 3.1, Revision 4, September 2012.
    - Part 2 Extended
- Common Criteria for Information Technology Security Evaluation Part 3: Security assurance components, Version 3.1 Revision 4, September 2012.
    - Part 3 Extended
- Protection Profile for Application Software, Version: 1.1, 5 November 2014 (ASPP11) with Application Software Protection Profile Extended Package: File Encryption. Version 1.0, 11/10/2014 (ASPP11/FEEP10)
- Package Claims:
    - Assurance Level: EAL 1 conformant

2.1 Conformance Rationale

The ST conforms to the ASPP11/FEEP10. As explained previously, the security problem definition, security objectives, and security requirements have been drawn from the PP.

3. Security Objectives

The Security Problem Definition may be found in the ASPP11/FEEP10 and this section reproduces only the corresponding Security Objectives for operational environment for reader convenience. The ASPP11/FEEP10 offers additional information about the identified security objectives, but that has not been reproduced here and the ASPP11/FEEP10 should be consulted if there is interest in that material.

In general, the ASPP11/FEEP10 has defined Security Objectives appropriate for software applications, and as such are applicable to the Data at Rest (DaR) Management TOE.

No objective statements have been provided.

4. Extended Components Definition

All of the extended requirements in this ST have been drawn from the ASPP11/FEEP10. The ASPP11/FEEP10 defines the following extended requirements and since they are not redefined in this ST the ASPP11/FEEP10 should be consulted for more information in regard to those CC extensions.

Extended SFRs:

- FCS_CKM_EXT.1: Cryptographic Key Generation Services
- FCS_CKM_EXT.2: Key Encrypting Key (KEK) Support
- FCS_CKM_EXT.4: Extended: Cryptographic Key Destruction
- FCS_IV_EXT.1: Extended: Initialization Vector Generation
- FCS_KYC_EXT.1: Key Chaining and Key Storage
- FCS_RBG_EXT.1: Random Bit Generation Services
- FCS_RBG_EXT.2: Random Bit Generation from Application
- FCS_STO_EXT.1: Storage of Secrets
- FDP_DAR_EXT.1: Encryption Of Sensitive Application Data
- FDP_DEC_EXT.1: Access to Platform Resources
- FDP_PRT_EXT.1: Extended: Protection of Selected User Data
- FIA_AUT_EXT.1: Authentication and Failure Handling
- FIA_FCT_EXT.1(2): Extended: User Authorization with Password/Passphrase Authorization Factors
- FMT_CFG_EXT.1: Secure by Default Configuration
- FMT_MEC_EXT.1: Supported Configuration Mechanism
- FPT_AEX_EXT.1: AntiExploitation Capabilities
- FPT_API_EXT.1: Use of Supported Services and APIs
- FPT_FEK_EXT.1: File Encryption Key (FEK) Support
- FPT_KYP_EXT.1: Extended: Protection of Key and Key Material
- FPT_LIB_EXT.1: Use of Third Party Libraries
- FPT_TUD_EXT.1: Integrity for Installation and Update
- FTP_DIT_EXT.1: Protection of Data in Transit

5. Security Requirements

This section defines the Security Functional Requirements (SFRs) and Security Assurance Requirements (SARs) that serve to represent the security functional claims for the Target of Evaluation (TOE) and to scope the evaluation effort.

The SFRs have all been drawn from the ASPP11/FEEP10. The refinements and operations already performed in the ASPP11/FEEP10 are not identified (e.g., highlighted) here, rather the requirements have been copied from the ASPP11/FEEP10 and any residual operations have been completed herein. Of particular note, the ASPP11/FEEP10 made a number of refinements and completed some of the SFR operations defined in the Common Criteria (CC) and that PP should be consulted to identify those changes if necessary.

The SARs are also drawn from the ASPP11/FEEP10 which includes all the SARs for EAL 1. However, the SARs are effectively refined since requirement-specific 'Assurance Activities' are defined in the ASPP11/FEEP10 that serve to ensure corresponding evaluations will yield more practical and consistent assurance than the EAL 1 assurance requirements alone. The ASPP11/FEEP10 should be consulted for the assurance activity definitions.

5.1 TOE Security Functional Requirements

The following table identifies the SFRs that are satisfied by Data at Rest (DaR) Management TOE.

Requirement Class	Requirement Component
FCS: Cryptographic support	FCS_CKM_EXT.1(A): Cryptographic key generation (Password/Passphrase conditioning)
	FCS_CKM_EXT.1: Key Encrypting Key (KEK) Support
	FCS_CKM_EXT.2: Cryptographic Key Generation (KEK)
	FCS_CKM_EXT.4: Extended: Cryptographic Key Destruction
	FCS_COP.1(1): Cryptographic Operation Encryption
	FCS_COP.1(4): Cryptographic Operation Encryption (Keyed-Hash Message Authentication)
	FCS_COP.1(5): Cryptographic Operation (Key Wrapping)
	FCS_IV_EXT.1: Extended: Initialization Vector Generation
	FCS_KYC_EXT.1: Key Chaining and Key Storage
	FCS_RBG_EXT.1: Random Bit Generation Services
	FCS_RBG_EXT.2: Random Bit Generation from Application
	FCS_STO_EXT.1: Storage of Secrets
FDP: User data protection	FDP_DAR_EXT.1: Encryption Of Sensitive Application Data
	FDP_DEC_EXT.1: Access to Platform Resources
	FDP_PRT_EXT.1: Extended: Protection of Selected User Data
FIA: Identification and authentication	FIA_AUT_EXT.1: Authentication and Failure Handling
	FIA_FCT_EXT.1(2): Extended: User Authorization with Password/Passphrase Authorization Factors
FMT: Security management	FMT_CFG_EXT.1: Secure by Default Configuration
	FMT_MEC_EXT.1: Supported Configuration Mechanism
	FMT_SMF.1: Specification of Management Functions
FPT: Protection of the TSF	FPT_AEX_EXT.1: AntiExploitation Capabilities
	FPT_API_EXT.1: Use of Supported Services and APIs
	FPT_FEK_EXT.1: File Encryption Key (FEK) Support
	FPT_KYP_EXT.1: Extended: Protection of Key and Key Material
	FPT_LIB_EXT.1: Use of Third Party Libraries
	FPT_TUD_EXT.1: Integrity for Installation and Update
FTP: Trusted path/channels	FTP_DIT_EXT.1: Protection of Data in Transit

Table 1 TOE Security Functional Components

5.1.1 Cryptographic support (FCS)

5.1.1.1 Cryptographic key generation (Password/Passphrase conditioning) (FCS_CKM_EXT.1(A))

FCS_CKM_EXT.1.1(A)
The TSF shall support a password/passphrase of up to [*74*] characters used to generate a password authorization factor.

FCS_CKM_EXT.1.2(A)
The TSF shall allow passwords to be composed of any combination of upper case characters, lower case characters, numbers, and the following special characters: "!", "@", "#", "$", "%", "^", "&", "*", "(", and ")", and [***no other characters***].

FCS_CKM_EXT.1.3(A)
The TSF shall perform Password-based Key Derivation Functions in accordance with a specified cryptographic algorithm [HMAC-[selection: ***SHA- 512***]], with [***4096***] iterations, and output cryptographic key sizes [***128***] that meet the following: [***NIST SP 800-132***].

FCS_CKM_EXT.1.4(A)
The TSF shall not accept passwords less than [***a value settable by the administrator***] and greater than the maximum password length defined in FCS_CKM_EXT.1.1(A).

FCS_CKM_EXT.1.5(A)
The TSF shall generate all salts using a RBG that meets FCS_RBG_EXT.1 (from the AS PP) and with entropy corresponding to the security strength selected for PBKDF in FCS_CKM_EXT.1.3(A).

5.1.1.2 Key Encrypting Key (KEK) Support (FCS_CKM_EXT.1)

FCS_CKM_EXT.1.1
The TSF shall support KEK in the following manner based on the selection chosen in FPT_FEK_EXT.1: [***derive a KEK using a password-based authorization factor conditioned as defined in FCS_CKM_EXT.1(A) and in accordance with FIA_FCT_EXT.1(2), using a Random Bit Generator as specified in FCS_RBG_EXT.1 (from the AS PP) and with entropy corresponding to the security strength of AES key sizes of [128 bit]***].

FCS_CKM_EXT.1.2
All KEKs shall be [***256-bit***] keys corresponding to at least the security strength of the keys encrypted by the KEK.

5.1.1.3 Cryptographic Key Generation (FEK) (FCS_CKM_EXT.2)

FCS_CKM_EXT.2.1
The TSF shall generate FEK cryptographic keys [***using a Random Bit Generator as specified in FCS_RBG_EXT.1 (from the AS PP) and with entropy corresponding to the security strength of AES key sizes of [128 bits]***].

FCS_CKM_EXT.2.2
The TSF shall create a unique FEK for each file (or set of files) using the mechanism on the client as specified in FCS_CKM_EXT.2.1.

FCS_CKM_EXT.2.3
The FEKs must be generated by the TOE.

5.1.1.4 Extended: Cryptographic Key Destruction (FCS_CKM_EXT.4)

FCS_CKM_EXT.4.1

The application shall [***invoke platform-provided key destruction, implement key destruction using***

- ***For volatile memory, the erasure shall be executed by a single direct overwrite [consisting of a pseudo-random pattern using the host platform's RBG, consisting of zeroes] following by a read-verify.***
- ***For non-volatile storage, the erasure shall be executed by: A [single] overwrite of key data storage location consisting of [a static pattern], followed by a [read-verify]. If read-verification of the overwritten data fails, the process shall be repeated again;]***

that meets the following: [***no standard***] for destroying all plaintext keying material and cryptographic security parameters when no longer needed.

5.1.1.5 Cryptographic Operation Encryption (FCS_COP.1(1))

FCS_COP.1.1(1)

Refinement: The application shall [***~~implement~~, implement AES encryption***] shall perform data encryption and decryption in accordance with a specified cryptographic algorithm AES used in [***CBC (as defined in NIST SP 800-38A)***] mode and cryptographic key sizes [***128-bits, 256-bits***].

5.1.1.6 Cryptographic Operation Encryption (Keyed-Hash Message Authentication (FCS_COP.1.1(4))

FCS_COP.1.1(4)

Refinement: The application shall [***invoke platform-provided functionality***] to perform keyed-hash message authentication in accordance with a specified cryptographic algorithm HMAC-[***SHA-384, SHA-512***], key size [***384,512***], and message digest size of [***384, 512***] bits that meet the following: FIPS PUB 198-1, "The Keyed-Hash Message Authentication Code", and FIPS PUB 180-4, "Secure Hash Standard".

5.1.1.7 Cryptographic Operation (Key Wrapping) (FCS_COP.1(5))

FCS_COP.1.1(5)

Refinement: The application shall [***use platform-provided functionality to perform Key Wrapping, implement functionality to perform Key Wrapping***] in accordance with a specified cryptographic algorithm [***AES Key Wrap, RSA using the KTS-OAEP-basic scheme***] and the cryptographic key size [***256 bits (AES), 2048 (RSA)***] that meet the following: [***NIST SP 800-38F' for Key Wrap (section 6.2) and Key Wrap with Padding (section 6.3), NIST SP 800-56B' for RSA using the KTS-OAEP-basic (section 9.2.3) and KTS-OAEP-receiver-confirmation (section9.2.4) scheme***].

5.1.1.8 Extended: Initialization Vector Generation (FCS_IV_EXT.1)

FCS_IV_EXT.1.1

The application shall [***generate IVs***] in accordance with Appendix H: Initialization Vector Requirements for NIST-Approved Cipher Modes.

5.1.1.9 Key Chaining and Key Storage (FCS_KYC_EXT.1)

FCS_KYC_EXT.1.1

The TSF shall maintain a primary key chain of: [***KEKs originating from one or more authorization factors(s) to the FEK(s) using the following method(s): [***

utilization of the platform key storage, and

implement key wrapping as specified in FCS_COP.1(5)
]
while maintaining an effective strength of [[128 bits] for symmetric keys; [112 bits] for asymmetric keys
] commensurate with the strength of the FEK.
] and [***No supplemental key chains***]
]

Note: even though the asymmetric key strength is only 112, the overall strength is still 128

5.1.1.10 Random Bit Generation Services (FCS_RBG_EXT.1)

FCS_RBG_EXT.1.1
The application shall [***invoke platform-provided DRBG functionality***, ***implement DRBG functionality***] for its cryptographic operations.

5.1.1.11 Random Bit Generation from Application (FCS_RBG_EXT.2)

FCS_RBG_EXT.2.1
The application shall perform all deterministic random bit generation (DRBG) services in accordance with [***NIST Special Publication 800-90A using [CTR_DRBG (AES)]***].
FCS_RBG_EXT.2.2
The deterministic RBG shall be seeded by an entropy source that accumulates entropy from a platform-based DRBG and [***a software-based noise source***] with a minimum of [***128 bits***] of entropy at least equal to the greatest security strength (according to NISTSP 80057) of the keys and hashes that it will generate.

5.1.1.12 Storage of Secrets (FCS_STO_EXT.1)

FCS_STO_EXT.1.1
The application shall [***invoke the functionality provided by the platform to securely store [keys]***] to nonvolatile memory.

5.1.2 User data protection (FDP)

5.1.2.1 Encryption Of Sensitive Application Data (FDP_DAR_EXT.1)

FDP_DAR_EXT.1.1
The application shall [***implement functionality to encrypt sensitive data***] in nonvolatile memory.

5.1.2.2 Access to Platform Resources (FDP_DEC_EXT.1)

FDP_DEC_EXT.1.1
The application shall provide user awareness of its intent to access [***SD card, network connectivity***].
FDP_DEC_EXT.1.2
The application shall provide user awareness of its intent to access [***no sensitive information repositories***].
FDP_DEC_EXT.1.3
The application shall only seek access to those resources for which it has provided a justification to access.
FDP_DEC_EXT.1.4
The application shall restrict network communication to [***application checking for updates***].
FDP_DEC_EXT.1.5
The application shall [***not transmit PII over a network***].

5.1.2.3 Extended: Protection of Selected User Data (FDP_PRT_EXT.1)

FDP_PRT_EXT.1.1
The TSF shall perform encryption and decryption of the user-selected file (or set of files) in accordance with FCS_COP.1(1).

FDP_PRT_EXT.1.2
The application shall [***implement functionality***] to ensure that all sensitive data created by the TOE when decrypting/encrypting the user-selected file (or set of files) are destroyed in volatile and non-volatile memory upon completion of the decryption/encryption operation.

5.1.3 Identification and authentication (FIA)

5.1.3.1 Authentication Failure Handling (FIA_AUT_EXT.1)

FIA_AUT_EXT.1.1
The application shall [***provide user authorization***] based on [***password/passphrase authorization factors***].

5.1.3.2 Extended: Single User Authorization with Password/Passphrase Authorization Factors (FIA_FCT_EXT.1(3))

The following reflects the Technical Decision defining a single user use-case.

FIA_FCT_EXT.1.1(3)
The TSF shall provide a mechanism as defined in FCS_CKM_EXT.1 and FCS_COP.1(4) to perform user authorization.

FIA_FCT_EXT.1.2(3)
The TSF shall perform user authorization using the mechanism provided in FIA_FCT_EXT.1.1(2) before allowing decryption of user data.

FIA_FCT_EXT.1.3(3)
The TSF shall verify that the user-entered authorization factors are valid before decrypting the user's encrypted files.

FIA_FCT_EXT.1.4(3)
The TSF shall ensure that the method of validation for each authorization factor does not expose or reduce the effective strength of the KEK, FEK, or CSPs used to derive the KEK or FEK.

FIA_FCT_EXT.1.5(2)
The TSF shall perform user authorization using the mechanism provided in FIA_FCT_EXT.1.1(2) before allowing the user to change the passphrase-based authorization factor as specified in FMT_SMF.1(c).

5.1.4 Security management (FMT)

5.1.4.1 Secure by Default Configuration (FMT_CFG_EXT.1)

FMT_CFG_EXT.1.1
The application shall only provide enough functionality to set new credentials when configured with default credentials or no credentials.

FMT_CFG_EXT.1.2
The application shall be configured by default with file permissions which protect it and its data from unauthorized access.

5.1.4.2 Supported Configuration Mechanism (FMT_MEC_EXT.1)

FMT_MEC_EXT.1.1
The application shall invoke the mechanisms recommended by the platform vendor for storing and setting configuration options.

5.1.4.3 Specification of Management Functions (FMT_SMF.1)

FMT_SMF.1.1
The TSF shall be capable of performing the following management functions: [***change password/passphrase authentication, configure password/passphrase complexity setting***].

5.1.5 Protection of the TSF (FPT)

5.1.5.1 AntiExploitation Capabilities (FPT_AEX_EXT.1)

FPT_AEX_EXT.1.1
The application shall not request to map memory at an explicit address except for [***none***].
FPT_AEX_EXT.1.2
The application shall [***not allocate any memory region with both write and execute permissions***].
FPT_AEX_EXT.1.3
The application shall be compatible with security features provided by the platform vendor.
FPT_AEX_EXT.1.4
The application shall not write user-modifiable files to directories that contain executable files unless explicitly directed by the user to do so.
FPT_AEX_EXT.1.5
The application shall be compiled with stack-based buffer overflow protection enabled.

5.1.5.2 Use of Supported Services and APIs (FPT_API_EXT.1)

FPT_API_EXT.1.1
The application shall only use supported platform APIs.

5.1.5.3 File Encryption Key (FEK) Support (FPT_FEK_EXT.1)

FPT_FEK_EXT.1.1
The TSF shall [- ***Store a FEK in Non-Volatile memory conformant with FPT_KYP_EXT.1***].

5.1.5.4 Extended: Protection of Key and Key Material (FPT_KYP_EXT.1)

FPT_KYP_EXT.1.1
The TSF shall [***only store keys in non-volatile memory when wrapped as specified in FCS_COP.1(5) unless the key meets any one of the following criteria [-The plaintext key is used to wrap a key as specified in FCS_COP.1(5) that is already wrapped as specified in FCS_COP.1(5).]***].

5.1.5.5 Use of Third Party Libraries (FPT_LIB_EXT.1)

FPT_LIB_EXT.1.1
The application shall be packaged with only [***libparser4.so***].

5.1.5.6 Integrity for Installation and Update (FPT_TUD_EXT.1)

FPT_TUD_EXT.1.1
The application shall [***leverage the platform***] to check for updates and patches to the application software.
FPT_TUD_EXT.1.2
The application shall be distributed using the format of the platform-supported package manager.
FPT_TUD_EXT.1.3
The application shall be packaged such that its removal results in the deletion of all traces of the application, with the exception of configuration settings, output files, and audit/log events.
FPT_TUD_EXT.1.4
The application shall not download, modify, replace or update its own binary code.

FPT_TUD_EXT.1.5
The application shall [***provide the ability***] to query the current version of the application software.

FPT_TUD_EXT.1.6
The application installation package and its updates shall be digitally signed such that its platform can cryptographically verify them prior to installation.

5.1.6 Trusted path/channels (FTP)

5.1.6.1 Protection of Data in Transit (FTP_DIT_EXT.1)

FTP_DIT_EXT.1.1
The application shall [***not transmit any sensitive data***] between itself and another trusted IT product.

5.2 TOE Security Assurance Requirements

The SARs for the TOE are the EAL 1 components as specified in Part 3 of the Common Criteria. Note that the SARs have effectively been refined with the assurance activities explicitly defined in association with both the SFRs and SARs.

Requirement Class	Requirement Component
ADV: Development	ADV_FSP.1: Basic functional specification
AGD: Guidance documents	AGD_OPE.1: Operational user guidance
	AGD_PRE.1: Preparative procedures
ALC: Life-cycle support	ALC_CMC.1: Labelling of the TOE
	ALC_CMS.1: TOE CM coverage
ATE: Tests	ATE_IND.1: Independent testing - conformance
AVA: Vulnerability assessment	AVA_VAN.1: Vulnerability survey

Table 2 EAL 1 Assurance Components

5.2.1 Development (ADV)

5.2.1.1 Basic functional specification (ADV_FSP.1)

ADV_FSP.1.1d
The developer shall provide a functional specification.

ADV_FSP.1.2d
The developer shall provide a tracing from the functional specification to the SFRs.

ADV_FSP.1.1c
The functional specification shall describe the purpose and method of use for each SFR-enforcing and SFR-supporting TSFI.

ADV_FSP.1.2c
The functional specification shall identify all parameters associated with each SFR-enforcing and SFR-supporting TSFI.

ADV_FSP.1.3c
The functional specification shall provide rationale for the implicit categorization of interfaces as SFR-non-interfering.

ADV_FSP.1.4c
The tracing shall demonstrate that the SFRs trace to TSFIs in the functional specification.

ADV_FSP.1.1e

The evaluator shall confirm that the information provided meets all requirements for content and presentation of evidence.

ADV_FSP.1.2e

The evaluator shall determine that the functional specification is an accurate and complete instantiation of the SFRs.

5.2.2 Guidance documents (AGD)

5.2.2.1 Operational user guidance (AGD_OPE.1)

AGD_OPE.1.1d

The developer shall provide operational user guidance.

AGD_OPE.1.1c

The operational user guidance shall describe, for each user role, the user-accessible functions and privileges that should be controlled in a secure processing environment, including appropriate warnings.

AGD_OPE.1.2c

The operational user guidance shall describe, for each user role, how to use the available interfaces provided by the TOE in a secure manner.

AGD_OPE.1.3c

The operational user guidance shall describe, for each user role, the available functions and interfaces, in particular all security parameters under the control of the user, indicating secure values as appropriate.

AGD_OPE.1.4c

The operational user guidance shall, for each user role, clearly present each type of security-relevant event relative to the user-accessible functions that need to be performed, including changing the security characteristics of entities under the control of the TSF.

AGD_OPE.1.5c

The operational user guidance shall identify all possible modes of operation of the TOE (including operation following failure or operational error), their consequences and implications for maintaining secure operation.

AGD_OPE.1.6c

The operational user guidance shall, for each user role, describe the security measures to be followed in order to fulfil the security objectives for the operational environment as described in the ST.

AGD_OPE.1.7c

The operational user guidance shall be clear and reasonable.

AGD_OPE.1.1e

The evaluator shall confirm that the information provided meets all requirements for content and presentation of evidence.

5.2.2.2 Preparative procedures (AGD_PRE.1)

AGD_PRE.1.1d

The developer shall provide the TOE including its preparative procedures.

AGD_PRE.1.1c

The preparative procedures shall describe all the steps necessary for secure acceptance of the delivered TOE in accordance with the developer's delivery procedures.

AGD_PRE.1.2c

The preparative procedures shall describe all the steps necessary for secure installation of the TOE and for the secure preparation of the operational environment in accordance with the security objectives for the operational environment as described in the ST.

AGD_PRE.1.1e

The evaluator shall confirm that the information provided meets all requirements for content and presentation of evidence.

AGD_PRE.1.2e

The evaluator shall apply the preparative procedures to confirm that the TOE can be prepared securely for operation.

5.2.3 Life-cycle support (ALC)

5.2.3.1 Labelling of the TOE (ALC_CMC.1)

ALC_CMC.1.1d

The developer shall provide the TOE and a reference for the TOE.

ALC_CMC.1.1c

The TOE shall be labelled with its unique reference.

ALC_CMC.1.1e

The evaluator shall confirm that the information provided meets all requirements for content and presentation of evidence.

5.2.3.2 TOE CM coverage (ALC_CMS.1)

ALC_CMS.1.1d

The developer shall provide a configuration list for the TOE.

ALC_CMS.1.1c

The configuration list shall include the following: the TOE itself; and the evaluation evidence required by the SARs.

ALC_CMS.1.2c

The configuration list shall uniquely identify the configuration items.

ALC_CMS.1.1e

The evaluator shall confirm that the information provided meets all requirements for content and presentation of evidence.

5.2.4 Tests (ATE)

5.2.4.1 Independent testing - conformance (ATE_IND.1)

ATE_IND.1.1d

The developer shall provide the TOE for testing.

ATE_IND.1.1c

The TOE shall be suitable for testing.

ATE_IND.1.1e

The evaluator shall confirm that the information provided meets all requirements for content and presentation of evidence.

ATE_IND.1.2e

The evaluator shall test a subset of the TSF to confirm that the TSF operates as specified.

5.2.5 Vulnerability assessment (AVA)

5.2.5.1 Vulnerability survey (AVA_VAN.1)

AVA_VAN.1.1d

The developer shall provide the TOE for testing.

AVA_VAN.1.1c

The TOE shall be suitable for testing.

AVA_VAN.1.1e
The evaluator shall confirm that the information provided meets all requirements for content and presentation of evidence.

AVA_VAN.1.2e
The evaluator shall perform a search of public domain sources to identify potential vulnerabilities in the TOE.

AVA_VAN.1.3e
The evaluator shall conduct penetration testing, based on the identified potential vulnerabilities, to determine that the TOE is resistant to attacks performed by an attacker possessing Basic attack potential.

6. TOE Summary Specification

This chapter describes the security functions:

- Cryptographic support
- User data protection
- Identification and authentication
- Security management
- Protection of the TSF
- Trusted path/channels

6.1 Cryptographic support

The TOE operates on an evaluated Android 4.4 device and uses the Security First's Secure Parser cryptographic module for encryption services and random number generation. The TOE's platform supports the following algorithms:

Functions	Standards	Certificates
Encryption/Decryption		
• Android: AES CBC (128 and 256) and Key Wrapping (256 bits)	FIPS PUB 197 NIST SP 800-38A NIST SP 800-38F	1884 & 3011 (LG G3 only)
• SPX Core: AES CBC (128 and 256) and Key Wrapping (256 bits)	FIPS PUB 197 NIST SP 800-38A NIST SP 800-38F	AES Cert # 3148, 3748 FIPS 140-2 Cert #1411
Cryptographic hashing		
• SHA-1, SHA-256, SHA-384, and SHA-512 (digest sizes 160, 256, 384, and 512 bits)	FIPS Pub 180-3	1655 & 2519 (LG G3 only)
Keyed-hash message authentication		
• HMAC-SHA-1(digest size 160)	FIPS Pub 198-1 FIPS Pub 180-3	1126 & 1903 (LG G3 only)
Random bit generation		
• Android: AES-256 CTR_DRBG	NIST SP 800-90	157 & 573 (LG G3 only)
• SPX Core AES-128 CTR_DRBG with software based noise sources with a minimum of 128 bits of non-determinism	NIST SP 800-90	1335
Asymmetric algorithms		
• RSA	FIPS Pub 186-4	960 & 1571 (LG G3 only)

Table 3 Algorithm Certificate Table

The TOE generally fulfills all of the NIST SP 800-56B requirements without extensions, the following table specifically identifies the "should", "should not", and "shall not" conditions from the publication along with an indication of how the TOE conforms to those conditions.

NIST SP800-56B Section Reference	"should", "should not", or "shall not"	Implemented?	Rationale for deviation
5.6	Should	Yes	Not applicable

NIST SP800-56B Section Reference	"should", "should not", or "shall not"	Implemented?	Rationale for deviation
5.8	shall not	No	Not applicable
5.9	shall not (first occurrence)	No	Not applicable
5.9	shall not (second occurrence)	No	Not applicable
6.1	should not	No	Not applicable
6.1	should (first occurrence)	Yes	Not applicable
6.1	should (second occurrence)	Yes	Not applicable
6.1	should (third occurrence)	Yes	Not applicable
6.1	should (fourth occurrence)	Yes	Not applicable
6.1	shall not (first occurrence)	No	Not applicable
6.1	shall not (second occurrence)	No	Not applicable
6.2.3	Should	Yes	Not applicable
6.5.1	Should	Yes	Not applicable
6.5.2	Should	Yes	Not applicable
6.5.2.1	Should	Yes	Not applicable
6.6	shall not	No	Not applicable
7.1.2	Should	Yes	Not applicable
7.2.1.3	Should	Yes	Not applicable
7.2.1.3	should not	No	Not applicable
7.2.2.3	should (first occurrence)	Yes	Not applicable
7.2.2.3	should (second occurrence)	Yes	Not applicable
7.2.2.3	should (third occurrence)	Yes	Not applicable
7.2.2.3	should (fourth occurrence)	Yes	Not applicable
7.2.2.3	should not	No	Not applicable
7.2.2.3	shall not	No	Not applicable
7.2.3.3	should (first occurrence)	Yes	Not applicable
7.2.3.3	should (second occurrence)	Yes	Not applicable
7.2.3.3	should (third occurrence)	Yes	Not applicable
7.2.3.3	should (fourth occurrence)	Yes	Not applicable
7.2.3.3	should (fifth occurrence)	Yes	Not applicable
7.2.3.3	should not	No	Not applicable
8	Should	Yes	Not applicable
8.3.2	should not	No	Not applicable

Table 4 NIST SP800-56B Conformance

The Cryptographic support function is designed to satisfy the following security functional requirements:

- FCS_CKM_EXT.1(A): The TOE allows the use of DaR passwords that support all special characters mentioned in FCS_CKM_EXT.1(A). 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 derives a key that serves as a secondary AES wrap on the FEFEK in conjunction with an asymmetric key wrap using 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 74 characters. The minimum password length can be change by an administrator to be any value between 6 and 74 characters.

The TOE performs 4096 iterations of the key derivation function in PBKDF2 to increase the computation needed to derive a key from the DaR password. With a thousand iterations on the evaluated platform, the average derivation time is 249.035 milliseconds.

The salt used in the PBKDF2 operations is generated by the platform's java.security.SecureRandom cryptographic API. First, a byte array is declared. The byte arrays are then filled with a random value from the platform's nextBytes method from SecureRandom. SecureRandom uses /dev/random. The salts are saved in /data/data, which is Android's protected directory. The salt lengths are 64 bytes.

- FCS_CKM_EXT.1: The TOE generates keys using both the Security First's Secure Parser (SPX Core) cryptographic module) and the platform's API (KeyGenerator) as described here. The SPX Core API uses an SP 800-90A AES-128 CTR DRBG and the platform uses an SP 800-90A AES-256 CTR DRBG. Both DRBGs are seeded with sufficient entropy from the platform itself.

  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 BouncyCastle 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 BouncyCastle 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 SPX Core DRBG is used to generate the FEKs. The FEK is wrapped by the FEKEK before being stored.

  FCS_CKM_EXT.2: The TOE generates file encryption keys using the Security First's Secure Parser cryptographic module, which implements an SP 800-90A AES-128 CTR DRBG. The DRBG is seeded with sufficient entropy to generate keys with 128 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's 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.

- FCS_CKM_EXT.4: The TOE relies on the platform and SPX Core 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 SPX Core is a FIPS module, it destroys FEKs by overwriting the targeted keys with zeroes. The SPX Core module_destroy API zeroes non-persistent CSPs from volatile memory.

Key Name	**Cleartext Storage Location**	**Destruction**	**Entity responsible for Destruction**	**When it is destroyed**
FEK	RAM	Java Garbage Collection	SPX Core	After file encryption/decryption
Management service unwrapped FEKEK	RAM	Java Garbage Collection	Platform key destruction API	After storing FEKEK to Application's BouncyCastle keystore
Application unwrapped FEKEK	RAM	Java Garbage Collection	SPX Core	After wrapping/unwrapping the FEK
PBKDF2 derived key	RAM	Java Garbage Collection	Platform key destruction API	After AES unwrapping double wrapped FEKEK
Management Service's private RSA key	TrustZone	TrustZone	TrustZone	After RSA unwrapping of double-wrapped FEKEK
Application's private RSA key	TrustZone	TrustZone	TrustZone	After RSA unwrapping of single-wrapped FEKEK

Table 5 Key Destruction

- FCS_COP.1(1): The TOE invokes the Security First's Secure Parser cryptographic module to perform AES-128-CBC encryption[2]. The TOE invokes the SPX Core's AES implementation to perform AES 128-CBC encryption when encrypting files using the 128-bit FEK[3].
- FCS_COP.1(4): The TOE uses the platform's HMAC-SHA-384 to hash the DaR password with a salt generated by the platform's DRBG. The hash value is used as an authentication factor to load the Management Service's BouncyCastle keystore, which houses an RSA wrapped file encryption key encryption key (FEKEK). If the DaR password is incorrect, the keystore will not load, and the calling application does not gain access to the key used to decrypt its files. The TOE uses the platform's 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 keywrap, after the FEKEK has been unwrapped using RSA-OAEP.
- FCS_COP.1(5): The TOE uses the SPX Core 256 bit AES key wrapping to wrap the FEK with the FEKEK. The TOE uses the evaluated platform's approved cryptographic API (e.g. cipher.init(Cipher.WRAP_MODE, aesWrappingKey), cipher.init(Cipher.UNWRAP_MODE, aesWrappingKey), OAEPEncoding(rsa, digest, null), cipher.wrap and cipher.unwrap) to perform key wrapping functions. The evaluated platform's API performs AES key wrapping in accordance with SP 800-38F and RSA key wrapping in accordance with SP 800-56B. The TOE's Operations and Maintenance manual [OM] contains the full list of APIs used by the TOE.
- FCS_IV_EXT.1: The TOE uses the SPX Core approved cryptographic API to perform Initialization vector generation using the SPX Core AES-128 CTR DRBG. The TOE's Operations and Maintenance manual [OM] contains the full list of APIs used by the TOE. The TOE only uses initialization vectors when performing AES-128 encryption/decryption of user data using the FEK. IVs are not used as part of any key wrap/unwrap process.
- FCS_KYC_EXT.1: 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 BouncyCastle keystore. The DaR password is 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 BouncyCastle 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 by temporarily[4] placing it in the Application's BouncyCastle keystore. 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 BouncyCastle keystore.

[2] The SPX Core can perform AES CBC encryption using either 128-bit or 256-bit keys as demonstrated by the CAVS certificates shown in Table 3 Algorithm Certificate Table.
[3] The FEK is actually a 256-bit key that has been generated with only 128-bits of strength. To avoid the impression that the FEK is stronger than 128-bits, the references to the FEK key size throughout the ST use the FEK strength and not size. Therefore, it is referred to as a 128-bit key. Cryptographic operations using the FEK, are actually using 256-bits of keying material which has a strength of 128-bits.
[4] The Management Service erases the single wrapped FEKEK from the application's keystore when the password timer expires.

When the Application needs to access the FEKEK, it obtains its RSA keypair from its own Android keystore and the single wrapped FEKEK from its own BouncyCastle keystore. The Application performs RSA-2048 unwrapping on the single wrapped FEKEK to obtain a cleartext FEKEK. The FEKEK is used as 256-bit AES key to unwrap the FEK. The FEK is a 128-bit AES key used to decrypt user data.

- FCS_RBG_EXT.1: TOE uses the Security First's Secure Parser cryptographic module for random number generation, which utilizes an SP 800-90A AES-128 CTR DRBG. This DRBG is used to generate the File Encryption keys (FEKs). The TOE generate the FEKEK using the platform's AES-256 AES CTR DRBG.
- FCS_RBG_EXT.2: The TOE uses the Security First's Secure Parser cryptographic module to generate random data. This Module always creates a global AES-256 CTR_DRBG upon load, seeds the global DRBG with 384-bits taken from /dev/urandom, uses that global DRBG elsewhere within the module when it requires random data, and reseeds the global DRBG (again drawing from /dev/urandom) after generating 1,000 blocks of random output.

  When TOE must generate an AES-128 bit FEK, the Security First's Secure Parser cryptographic module creates a separate AES-128 CTR_DRBG context that it uses solely for generation of keys. The module will seed this AES-128 DRBG context using 384-bits of data drawn from the global DRBG.
- FCS_STO_EXT.1: All 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 Management Service's BouncyCastle keystore to store the double wrapped FEKEK
  - The TOE uses the Application's BouncyCastle keystore to store the single wrapped FEKEK.
  - The wrapped FEK and IV are stored in the same hidden directory as the encrypted file, which is all stored on flash.

6.2 User data protection

The TOE protects user data by providing encryption of user selected data. The TOE uses 128-bit AES keys to encrypt the files stored to flash. These keys are derived from DaR passwords through a key derivation and key wrapping process. If an application does not enter the right DaR password, the file encryption key will never be derived correctly, thus preventing the application from decrypting its files. The AES keys used as a File Encryption Key (FEK) are generated using the Security First's Secure Parser cryptographic module. The random data collected to seed the SPX Core DRBG comes from the evaluated device's /dev/urandom.

The User data protection function is designed to satisfy the following security functional requirements:

- FDP_DAR_EXT.1: The TOE implements functionality to encrypt data and store it securely on the evaluated platform. The TOE uses the Security First's Secure Parser cryptographic module to generate AES-128 bit keys for file encryption (a.k.a., FEK).
- FDP_DEC_EXT.1: The TOE only uses the READ_EXTERNAL_STORAGE to write to an external SD Card. The TOE does not access any of the listed sensitive information repositories. The TOE communicates over networks only for the purpose of checking for TOE updates and does not transmit PII data over a network. According to the TOE's Operations and Maintenance manual [OM], the DaR can read/write data to any directory based on the permissions of the integrated application. These directories can reside either in the internal Android SD Card or in expandable memory.

  The TOE requires permissions in order to perform the following:
  - Modify or delete the contents of USB storage
  - Read the contents of USB storage

  - Use Accounts on the device
  - Full network access in order to check for product updates
  - Reorder running apps
  - Retrieve running apps
  - Run at startup
  - Prevent Phone from Sleeping

- FDP_PRT_EXT.1: 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 so that access to specific parts within an encrypted file is easier. Using the SPX Core'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 DaR returns the plaintext data out of our API to the application for processing. The DaR clears all internal buffers of the data. The SPX Core also has routines to destroy keys stored inside the SPX Core. The TOE does not create any temporary resources.

  Each chunk is decrypted separately (using the SPX Core'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 hasn't been chunked, the entire file needs to be decrypted, and the data needs to be inserted in 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 DaR 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 Table 5 Key Destruction (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 SPX Core also has its own zeroization. The module_destroy API zeroes the FEK and FEKEK from volatile memory per Table 5 Key Destruction.

6.3 Identification and authentication

The TOE maintains identification and authentication by using DaR passwords. In order for an application to unlock its files, the application must provide the correct DaR password. The DaR password is used to derive the necessary keys in order to obtain the file encryption key, which is used to decrypt the files.

The Identification and authentication function is designed to satisfy the following security functional requirements:

- FIA_AUT_EXT.1: The TOE provides a DaR password based authorization factor in order to authenticate to the DaR service.
- FIA_FCT_EXT.1(3): The TOE allows a single user to login to the TOE with their own DaR password (a.k.a., authorization factor). The TOE prompts for a DaR password when the DaR service starts. The TOE provides user authorization by requiring a user to provide a DaR password to gain access to the DaR service as well as the user's associated keys for the specific application. Encrypt and Decrypt operations are explicitly requested using the TOE library. The DaR password is used to obtain two unique keys used to recover the file encryption key encryption key (FEKEK). The DaR password is first hashed using HMAC-

SHA384. The resulting hash is used as a DaR password to load the Management Service's BouncyCastle keystore. If the DaR password is incorrect, the keystore will not load, and the wrapped FEKEK cannot be loaded. The DaR password previously provided is then used to derive a key using PBKDF2 with HMAC-SHA-512 as the pseudo-random function. This key is used to further unwrap the FEKEK. If the DaR password is incorrect, the key derived using PBKDF2 will not be able to successfully unwrap the FEKEK retrieved from the Management Service's BouncyCastle keystore. The FEKEK is ultimately used to decrypt the actual file encryption key used to encrypt/decrypt files. In order to change a DaR password, the user must provide the previous DaR password to retrieve the FEKEK. If authentication is successful, the FEKEK is loaded, and the TOE will then use the new DaR password to derive keys to rewrap the FEKEK.

6.4 Security management

The TOE does not allow invocation of its services until a configuration file has been created. The configuration options are stored in the evaluated Android 4.4 OS's defined private area on flash memory. The TOE allows users to change DaR passwords as part of its security management.

The Security management function is designed to satisfy the following security functional requirements:

- FMT_CFG_EXT.1: The TOE restricts access to its services upon first use. The services are grayed out until a configuration file is created. This allows the TOE to force the user to configure the TOE before accessing the TOE's services. There are no default credentials within the TOE.
- FMT_MEC_EXT.1: The TOE's evaluated Android platform automatically uses /data/data/package/shared_prefs/ to store configuration options and settings. The Time-outs, Lock-outs, and SALTs for PBKDFv2 are stored in private files in the file directory.
- FMT_SMF.1: The TOE provides the ability to change DaR passwords/passphrases and configure the DaR password/passphrase complexity setting.

6.5 Protection of the TSF

The TOE is physically protected by the boundary of the evaluated device. The TOE is executed on an evaluated Android 4.4 OS. The TOE utilizes the evaluated platform's APIs only. Android's application management requires application updates to be signed with an Android key, thus allowing the secure updates of its applications. The Android OS Linux kernel is capable of ASLR (address space layout randomization), ensuring that no application uses the same address layout on two different devices. Keys are also stored in memory, which can be wiped by rebooting the device.

The Protection of the TSF function is designed to satisfy the following security functional requirements:

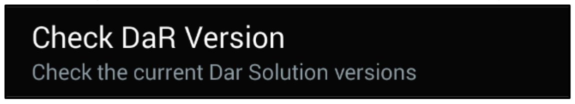
- FPT_AEX_EXT.1: The application is compiled with "-v -DBUILD_JNI -DANDROID -DCyber -O2 -fstack-protector-all –fexceptions" in order to enable ASLR and stack-based buffer overflow protection. The Linux kernel of the TOE's Android operating system also provides address space layout randomization utilizing the get_random_int(void) kernel random function to provide eight unpredictable bits to the base address of any user-space memory mapping. The random function, though not cryptographic, ensures that one cannot predict the value of the bits.
- FPT_API_EXT.1: The TOE uses only platform provided APIs and does not import any third party APIs. The TOE's Operations and Maintenance manual [OM] contains the full list of APIs used by the TOE.
- FPT_FEK_EXT.1: The TOE stores keys in non-volatile memory in conformance with FPT_KYP_EXT.1. When keys are no longer needed, they are destroyed by the platform's mechanism. The TOE destroys keys in volatile memory (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 TOE programmatically destroys these keys in memory after they are no longer needed by the TOE (i.e. after encryption/decryption). The TOE relies on Android's platform application protections to prevent disclosure of application memory, which can lead to recovery of keys. The TOE also stores FEKs on flash storage. The SPX Core has routines to destroy keys and secrets that are stored within the SPX Core. The TOE calls the SPX Core's module_destroy function to destroy FEKs.

- FPT_KYP_EXT.1: The TOE stores keys in nonvolatile memory by relying on the Android and BouncyCastle keystores. The TOE uses the Management Service's BouncyCastle keystore to store the double wrapped FEKEK. The TOE uses the Android keystore to store all RSA public/private keys, which are used to unwrap the double wrapped FEKEKS. The unwrapped FEKEKs are further unwrapped using AES key wrap. 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 keys are stored in non-volatile memory using Android's keystore API.
- FPT_LIB_EXT.1: The TOE only uses the third party library ***libparser4.so***.
- FPT_TUD_EXT.1: If a security vulnerability was found by a user, then the user must report it to Trivalent's email at support@trivalent.co. In the case that the vulnerability impacts the Trivalent DaR API: Trivalent will deliver updated API Code, as well as additional developer's documentation outlining any potential changes in the implementation. 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 the case the vulnerability directly relates to the Trivalent Management Service APK: Trivalent 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 Trivalent 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.

  The TOE's software is digitally signed by Trivalent. Each update is accompanied by documentation outlining changes to the overall service, as well as compatible versions of the Trivalent API.

  Checking for updates can be done in the app by selecting

  A popup will appear indicating whether an update is necessary and instructions on how to retrieve it.

6.6 Trusted path/channels

The TOE does not transmit data to any other products.

The Trusted path/channels function is designed to satisfy the following security functional requirements:

- FTP_DIT_EXT.1: The TOE does not transmit any data between itself and other products.