Samsung Electronics Co., Ltd. Samsung Galaxy Note 4 Android 5 (MDFPP20) Security Target

Version 0.4
2015/03/13

Prepared for:

Samsung Electronics Co., Ltd.

416 Maetan-3dong, Yeongtong-gu, Suwon-si, Gyeonggi-do, 443-742 Korea

Prepared By:

Gossamer Laboratories

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 consists of the Samsung Galaxy Note 4 Android 5 provided by Samsung Electronics Co., Ltd.. The TOE is being evaluated as a Mobile Device.

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 MDFPP 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 – Samsung Electronics Co., Ltd. Samsung Galaxy Note 4 Android 5 (MDFPP20) Security Target

ST Version – Version 0.4

ST Date – 2015/03/13

1.2 TOE Reference

TOE Identification – Samsung Electronics Co., Ltd. Samsung Galaxy Note 4 Android 5.

TOE Developer – Samsung Electronics Co., Ltd.

Evaluation Sponsor – Samsung Electronics Co., Ltd.

1.3 TOE Overview

The Target of Evaluation (TOE) are the Samsung Galaxy Note 4 Android 5.

1.4 TOE Description

The TOE is a mobile operating system based on Android 5.0.1 with modifications made to increase the level of security provided to end users and enterprises. The TOE is intended to be used as part of an enterprise messaging solution providing mobile staff with enterprise connectivity.

The TOE includes a Common Criteria mode (or "CC mode") that an administrator can invoke through the use of an MDM or through a dedicated administrative application (see the Guidance for instructions to obtain the application). The TOE must be configured as follows in order for an administrator to transition the TOE to CC mode.

- Require a screen lock password (swipe, PIN, pattern, or facial recognition screen locks are not allowed).
- The maximum password failure retry policy should be less than or equal to ten.
- Device encryption must be enabled.
- SDCard encryption must be enabled.
- Revocation checking must be enabled.

When CC mode has been enabled, the TOE behaves as follows.

- The TOE sets the system wide Android CC mode property to "Enabled".
- The TOE performs FIPS 140-2 power-on self-tests.
- The TOE performs self-tests for the key management.
- The TOE performs secure boot integrity checking of the kernel and key system executables.
- The TOE prevents loading of custom firmware/kernels and requires all updates occur through FOTA (Samsung's Firmware Over The Air firmware update method)
- The TOE uses CAVP approved cryptographic ciphers when joining and communicating with wireless networks.
- The TOE utilizes CAVP approved cryptographic ciphers for TLS.
- The TOE ensures FOTA updates utilize 2048-bit PKCS #1 RSA-PSS formatted signatures (with SHA-512 hashing).

The TOE includes a containerization capability, KNOX. This container provides a way to segment applications and data into two separate areas on the device, such as a personal area and a work area, each with its own separate apps, data and security policies. For this effort the TOE was evaluated both without and with a KNOX container created (and to create a KNOX container, one must purchase an additional license). Thus, the evaluation includes several KNOX-specific claims that apply to a KNOX container when created.

There are different models of the TOE, the Samsung Galaxy Note 4 Android 5. These models differ only in the amount of internal storeage:

1. Samsung manufactures the tested Galaxy Note 4 hardware with an LTE radio and offers it with 16GB or 32GB of internal Flash storage.

The model number of the mobile device used during the evaluation is as follows:

Device Name	Model Number	Android Version	Kernel Version	Build Number
Galaxy Note 4 (Qualcomm)	SM-N910F	5.0.1	3.10.40	LRX22C

1.4.1 TOE Architecture

The TOE combines with a Mobile Device Management solution that enables the enterprise to watch, control and administer all deployed mobile devices, across multiple mobile service providers as well as facilitate secure communications through a VPN. This partnership provides a secure mobile environment that can be managed and controlled by the environment and reduce the risks that can be introduced through a Bring-Your-Own-Device (BYOD) model.

Data on the TOE is protected through the implementation of Samsung On-Device Encryption (ODE) which utilizes a CAVP certified cryptographic algorithms to encrypt device and SD card storage. This functionality is combined with a number of on-device policies including local wipe, remote wipe, password complexity, automatic lock and privileged access to security configurations to prevent unauthorized access to the device and stored data.

The Samsung Enterprise Software Development Kit (SDK) builds on top of the existing Android security model by expanding the current set of security configuration of options to more than 390 configurable policies and including additional security functionality such as application whitelisting and blacklisting.

KNOX provides the ability to enhance the BYOD model by creating a separate container for the Enterprise. Within this container, the Enterprise can provision separate applications and ensure they are kept separate from anything the user may do outside the KNOX container. The Enterprise can use policy controls to manage the device as a whole or the KNOX container specifically, as needed by the organization.

1.4.1.1 Physical Boundaries

The TOE is a multi-user operating system based on Android (5.0.1) that incorporates the Samsung Enterprise SDK. The TOE does not include the user applications that run on top of the operating system, but does include controls that limit application behavior. The TOE is used as a mobile device within an enterprise environment where the configuration of the device is managed through a compliant device management solution.

The TOE communicates and interacts with 802.11-2012 Access Points and mobile data networks to establish network connectivity, and the through that connectivity interacts with MDM servers that allow administrative control of the TOE.

1.4.1.2 Logical Boundaries

This section summarizes the security functions provided by the Samsung Galaxy Note 4 Android 5:

- Cryptographic support
- User data protection

- Identification and authentication
- Security management
- Protection of the TSF
- TOE access
- Trusted path/channels

1.4.1.2.1 Cryptographic support

The TOE includes a cryptographic module with CAVP certified algorithms for a wide range of cryptographic functions including: asymmetric key generation and establishment, symmetric key generation, encryption/decryption, cryptographic hashing and keyed-hash message authentication. These functions are supported with suitable random bit generation, key derivation, salt generation, initialization vector generation, secure key storage, and key and protected data destruction. These primitive cryptographic functions are used to implement security protocols such as TLS, IPsec, and HTTPS and also to encrypt the media (including the generation and protection of data, right, and key encryption keys) used by the TOE. Many of these cryptographic functions are also accessible as services to applications running on the TOE.

1.4.1.2.2 User data protection

The TOE is designed to control access to system services by hosted applications, including protection of the Trust Anchor Database. Additionally, the TOE is design to protect user and other sensitive data using encryption so that even if a device is physically lost, the data remains protected. The functionality provided by a KNOX container enhances the security of user data by providing an additional layer of separation between apps and data while the device is in use.

1.4.1.2.3 Identification and authentication

The TOE supports a number of features related to identification and authentication. From a user perspective, except for making phone calls to an emergency number, a password (i.e., Password Authentication Factor) must be correctly entered to unlock the TOE. Also, even when the TOE is unlocked the password must be re-entered to change the password. Passwords are obscured when entered so they cannot be read from the TOE's display and the frequency of entering passwords is limited and when a configured number of failures occurs, the TOE will be wiped to protect its contents. Passwords can be constructed using upper and lower cases characters, numbers, and special characters and passwords between 4 and 16 characters are supported.

The TOE can also serve as an 802.1X supplicant and can use X509v3 and validate certificates for EAP-TLS, TLS and IPsec exchanges.

1.4.1.2.4 Security management

The TOE provides all the interfaces necessary to manage the security functions identified throughout this Security Target as well as other functions commonly found in mobile devices. Many of the available functions are available to users of the TOE while many are restricted to administrators operating through a Mobile Device Management solution once the TOE has been enrolled. Once the TOE has been enrolled and then un-enrolled, it removes all MDM policies and disables CC mode.

1.4.1.2.5 Protection of the TSF

The TOE implements a number of features designed to protect itself to ensure the reliability and integrity of its security features. It protects particularly sensitive data such as cryptographic keys so that they are not accessible or exportable. It also provides its own timing mechanism to ensure that reliable time information is available (e.g., for log accountability). It enforces read, write, and execute memory page protections, uses address space layout randomization, and stack-based buffer overflow protections to minimize the potential to exploit application flaws. It is also designed to protect itself from modification by applications as well as to isolate the address spaces of applications from one another to protect those applications.

The TOE includes functions to perform self-tests and software/firmware integrity checking so that it might detect when it is failing or may be corrupt. If any of the self-tests fail, the TOE will not go into an operational mode. It also includes mechanisms (i.e., verification of the digital signature of each new image) so that the TOE itself can be updated while ensuring that the updates will not introduce malicious or other unexpected changes in the TOE. Digital signature checking also extends to verifying applications prior to their installation.

1.4.1.2.6 TOE access

The TOE can be locked, obscuring its display, by the user or after a configured interval of inactivity. The TOE also has the capability to display an advisory message (banner) when users unlock the TOE for use.

The TOE is also able to attempt to connect to wireless networks as configured.

1.4.1.2.7 Trusted path/channels

The TOE supports the use of 802.11-2012, 802.1X, EAP-TLS, TLS and IPsec to secure communications channels between itself and other trusted network devices.

1.4.2 TOE Documentation

Samsung Android 5 on Galaxy Devices Guidance Documentation, version 2.0a, March 13, 2015 [Admin Guide]

Samsung Android 5 on Galaxy Devices User Guidance Documentation, version 2.0a, March 16, 2015 [User-Guide].

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 Mobile Device Fundamentals, Version 2, 17 September 2014 (MDFPP20)
- Package Claims:
    - Assurance Level: EAL 1 augmented with ALC_TSU_EXT.1

2.1 Conformance Rationale

The ST conforms to the MDFPP20. As explained previously, the security problem definition, security objectives, and security requirements are defined the PP.

3. Security Objectives

The Security Problem Definition may be found in the MDFPP20 and this section reproduces only the corresponding Security Objectives for operational environment for reader convenience. The MDFPP20 offers additional information about the identified security objectives, but that has not been reproduced here and the MDFPP20 should be consulted if there is interest in that material.

In general, the MDFPP20 has defined Security Objectives appropriate for Mobile Device and as such are applicable to the Samsung Galaxy Note 4 Android 5 TOE.

3.1 Security Objectives for the Operational Environment

- **OE.CONFIG** TOE administrators will configure the Mobile Device security functions correctly to create the intended security policy.
- **OE.NOTIFY** The Mobile User will immediately notify the administrator if the Mobile Device is lost or stolen.
- **OE.PRECAUTION** The Mobile User exercises precautions to reduce the risk of loss or theft of the Mobile Device.

4. Extended Components Definition

All of the extended requirements in this ST have been drawn from the MDFPP20. The MDFPP20 defines the following extended SFRs and SARs and since they are not redefined in this ST the MDFPP20 should be consulted for more information in regard to those CC extensions.

Extended SFRs:

- FCS_CKM_EXT.1: Extended: Cryptographic Key Support
- FCS_CKM_EXT.2: Extended: Cryptographic Key Random Generation
- FCS_CKM_EXT.3: Extended: Cryptographic Key Generation
- FCS_CKM_EXT.4: Extended: Key Destruction
- FCS_CKM_EXT.5: Extended: TSF Wipe
- FCS_CKM_EXT.6: Extended: Salt Generation
- FCS_HTTPS_EXT.1: Extended: HTTPS Protocol
- FCS_IV_EXT.1: Extended: Initialization Vector Generation
- FCS_RBG_EXT.1: Extended: Cryptographic Operation (Random Bit Generation)
- FCS_SRV_EXT.1: Extended: Cryptographic Algorithm Services
- FCS_STG_EXT.1: Extended: Cryptographic Key Storage
- FCS_STG_EXT.2: Extended: Encrypted Cryptographic Key Storage
- FCS_STG_EXT.3: Extended: Integrity of encrypted key storage
- FCS_TLSC_EXT.1: Extended: EAP TLS Protocol
- FCS_TLSC_EXT.2: Extended: TLS Protocol
- FDP_ACF_EXT.1: Extended: Security access control
- FDP_DAR_EXT.1: Extended: Protected Data Encryption
- FDP_DAR_EXT.2: Extended: Sensitive Data Encryption
- FDP_IFC_EXT.1: Extended: Subset information flow control
- FDP_STG_EXT.1: Extended: User Data Storage
- FDP_UPC_EXT.1: Extended: Inter-TSF user data transfer protection
- FIA_AFL_EXT.1: Authentication failure handling
- FIA_BLT_EXT.1: Extended: Bluetooth User Authorization
- FIA_BLT_EXT.2: Extended: Bluetooth Authentication
- FIA_PAE_EXT.1: Extended: PAE Authentication
- FIA_PMG_EXT.1: Extended: Password Management
- FIA_TRT_EXT.1: Extended: Authentication Throttling
- FIA_UAU_EXT.1: Extended: Authentication for Cryptographic Operation
- FIA_UAU_EXT.2: Extended: Timing of Authentication
- FIA_UAU_EXT.3: Extended: Re-Authentication

- FIA_X509_EXT.1: Extended: Validation of certificates
- FIA_X509_EXT.2: Extended: X509 certificate authentication
- FIA_X509_EXT.3: Extended: Request Validation of certificates
- FMT_MOF_EXT.1: Extended: Management of security functions behavior
- FMT_SMF_EXT.1: Extended: Specification of Management Functions
- FMT_SMF_EXT.2: Extended: Specification of Remediation Actions
- FPT_AEX_EXT.1: Extended: Anti-Exploitation Services (ASLR)
- FPT_AEX_EXT.2: Extended: Anti-Exploitation Services (Memory Page Permissions)
- FPT_AEX_EXT.3: Extended: Anti-Exploitation Services (Overflow Protection)
- FPT_AEX_EXT.4: Extended: Domain Isolation
- FPT_BBD_EXT.1: Application Processor Mediation
- FPT_KST_EXT.1: Extended: Key Storage
- FPT_KST_EXT.2: Extended: No Key Transmission
- FPT_KST_EXT.3: Extended: No Plaintext Key Export
- FPT_NOT_EXT.1: Extended: Self-Test Notification
- FPT_TST_EXT.1: Extended: TSF Cryptographic Functionality Testing
- FPT_TST_EXT.2: Extended: TSF Integrity Testing
- FPT_TUD_EXT.1: Extended: Trusted Update: TSF version query
- FPT_TUD_EXT.2: Extended: Trusted Update Verification
- FTA_SSL_EXT.1: Extended: TSF- and User-initiated locked state
- FTA_WSE_EXT.1: Extended: Wireless Network Access
- FTP_ITC_EXT.1: Extended: Trusted channel Communication

Extended SARs:

- ALC_TSU_EXT.1: Timely Security Updates

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 MDFPP20. The refinements and operations already performed in the MDFPP20 are not identified (e.g., highlighted) here, rather the requirements have been copied from the MDFPP20 and any residual operations have been completed herein. Of particular note, the MDFPP20 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 MDFPP20 which includes all the SARs for EAL 1 augmented with ALC_TSU_EXT.1. However, the SARs are effectively refined since requirement-specific 'Assurance Activities' are defined in the MDFPP20 that serve to ensure corresponding evaluations will yield more practical and consistent assurance than the assurance requirements alone. The MDFPP20 should be consulted for the assurance activity definitions.

5.1 TOE Security Functional Requirements

The following table identifies the SFRs that are satisfied by the Samsung Galaxy Note 4 Android 5 TOE.

Requirement Class	Requirement Component
FCS: Cryptographic support	FCS_CKM.1(1): Cryptographic key generation
	FCS_CKM.1(2): Cryptographic key generation
	FCS_CKM.2(1): Cryptographic key establishment
	FCS_CKM.2(2): Cryptographic key distribution
	FCS_CKM_EXT.1: Extended: Cryptographic Key Support
	FCS_CKM_EXT.2: Extended: Cryptographic Key Random Generation
	FCS_CKM_EXT.3: Extended: Cryptographic Key Generation
	FCS_CKM_EXT.4: Extended: Key Destruction
	FCS_CKM_EXT.5: Extended: TSF Wipe
	FCS_CKM_EXT.6: Extended: Salt Generation
	FCS_COP.1(1): Cryptographic operation
	FCS_COP.1(2): Cryptographic operation
	FCS_COP.1(3): Cryptographic operation
	FCS_COP.1(4): Cryptographic operation
	FCS_COP.1(5): Cryptographic operation
	FCS_HTTPS_EXT.1: Extended: HTTPS Protocol
	FCS_IV_EXT.1: Extended: Initialization Vector Generation
	FCS_RBG_EXT.1(*): Extended: Cryptographic Operation (Random Bit Generation)
	FCS_SRV_EXT.1: Extended: Cryptographic Algorithm Services
	FCS_STG_EXT.1: Extended: Cryptographic Key Storage
	FCS_STG_EXT.2: Extended: Encrypted Cryptographic Key Storage
	FCS_STG_EXT.3: Extended: Integrity of encrypted key storage
	FCS_TLSC_EXT.1: Extended: EAP TLS Protocol
	FCS_TLSC_EXT.2: Extended: TLS Protocol
FDP: User data protection	FDP_ACF_EXT.1(*): Extended: Security access control
	FDP_DAR_EXT.1: Extended: Protected Data Encryption
	FDP_DAR_EXT.2: Extended: Sensitive Data Encryption

	FDP_IFC_EXT.1: Extended: Subset information flow control
	FDP_STG_EXT.1: Extended: User Data Storage
	FDP_UPC_EXT.1: Extended: Inter-TSF user data transfer protection
FIA: Identification and authentication	FIA_AFL_EXT.1(*): Authentication failure handling
	FIA_BLT_EXT.1: Extended: Bluetooth User Authorization
	FIA_BLT_EXT.2: Extended: Bluetooth Authentication
	FIA_PAE_EXT.1: Extended: PAE Authentication
	FIA_PMG_EXT.1: Extended: Password Management
	FIA_TRT_EXT.1: Extended: Authentication Throttling
	FIA_UAU.7: Protected authentication feedback
	FIA_UAU_EXT.1: Extended: Authentication for Cryptographic Operation
	FIA_UAU_EXT.2: Extended: Timing of Authentication
	FIA_UAU_EXT.3: Extended: Re-Authentication
	FIA_X509_EXT.1: Extended: Validation of certificates
	FIA_X509_EXT.2: Extended: X509 certificate authentication
	FIA_X509_EXT.3: Extended: Request Validation of certificates
FMT: Security management	FMT_MOF_EXT.1: Extended: Management of security functions behavior
	FMT_SMF_EXT.1: Extended: Specification of Management Functions
	FMT_SMF_EXT.2(*): Extended: Specification of Remediation Actions
FPT: Protection of the TSF	FPT_AEX_EXT.1: Extended: Anti-Exploitation Services (ASLR)
	FPT_AEX_EXT.2: Extended: Anti-Exploitation Services (Memory Page Permissions)
	FPT_AEX_EXT.3: Extended: Anti-Exploitation Services (Overflow Protection)
	FPT_AEX_EXT.4: Extended: Domain Isolation
	FPT_BBD_EXT.1: Application Processor Mediation
	FPT_KST_EXT.1: Extended: Key Storage
	FPT_KST_EXT.2: Extended: No Key Transmission
	FPT_KST_EXT.3: Extended: No Plaintext Key Export
	FPT_NOT_EXT.1: Extended: Self-Test Notification
	FPT_STM.1: Reliable time stamps
	FPT_TST_EXT.1: Extended: TSF Cryptographic Functionality Testing
	FPT_TST_EXT.2: Extended: TSF Integrity Testing
	FPT_TUD_EXT.1: Extended: Trusted Update: TSF version query
	FPT_TUD_EXT.2: Extended: Trusted Update Verification
FTA: TOE access	FTA_SSL_EXT.1: Extended: TSF- and User-initiated locked state
	FTA_TAB.1: Default TOE Access Banners
	FTA_WSE_EXT.1: Extended: Wireless Network Access
FTP: Trusted path/channels	FTP_ITC_EXT.1: Extended: Trusted channel Communication

Table 1 TOE Security Functional Components

5.1.1 Cryptographic support (FCS)

5.1.1.1 Cryptographic key generation (FCS_CKM.1(1))

FCS_CKM.1(1).1

The TSF shall generate asymmetric cryptographic keys in accordance with a specified cryptographic key generation algorithm [***- [RSA schemes] using cryptographic key sizes of [2048-bit or greater] that meet the following: [o ANSI X9.31-1998, Section 4.1;],***
- [ECC schemes] using ['NIST curves' P-256, P-384 and [P-521]] that meet the following: [FIPS PUB 186-4, 'Digital Signature Standard (DSS)', Appendix B.4];
- [FFC schemes] using cryptographic key sizes of [2048-bit or greater] that meet the following: [FIPS PUB 186-4, 'Digital Signature Standard (DSS)', Appendix B.1]].

5.1.1.2 Cryptographic key generation (FCS_CKM.1(2))

FCS_CKM.1(2).1

The TSF shall generate symmetric cryptographic keys in accordance with a specified cryptographic key generation algorithm [PRF-384] and specified cryptographic key sizes [128 bits] using a Random Bit Generator as specified in FCS_RBG_EXT.1 that meet the following: [IEEE 802.11-2012].

5.1.1.3 Cryptographic key establishment (FCS_CKM.2(1))

FCS_CKM.2(1).1

The TSF shall perform cryptographic key establishment in accordance with a specified cryptographic key establishment method:
- [RSA-based key establishment schemes] that meets the following: [NIST Special Publication 800-56B, 'Recommendation for Pair-Wise Key Establishment Schemes Using Integer Factorization Cryptography'];
and [- ***[Elliptic curve-based key establishment schemes] that meets the following: [NIST Special Publication 800-56A, 'Recommendation for Pair-Wise Key Establishment Schemes Using Discrete Logarithm Cryptography'];***
- [Finite field-based key establishment schemes] that meets the following: [NIST Special Publication 800-56A, 'Recommendation for Pair-Wise Key Establishment Schemes Using Discrete Logarithm Cryptography'];].

5.1.1.4 Cryptographic key distribution (FCS_CKM.2(2))

FCS_CKM.2(2).1

The TSF shall decrypt Group Temporal Key (GTK) in accordance with a specified cryptographic key distribution method [AES Key Wrap in an EAPOL-Key frame] that meets the following: [NIST SP 800-38F, IEEE 802.11-2012 for the packet format and timing considerations] and does not expose the cryptographic keys.

5.1.1.5 Extended: Cryptographic Key Support (FCS_CKM_EXT.1)

FCS_CKM_EXT.1.1

The TSF shall support a [***hardware-protected***] REK with a key of size [***256 bits***].

FCS_CKM_EXT.1.2

System software on the TSF shall be able only to request [***AES encryption/decryption, NIST SP 800-108 key derivation***] by the key and shall not be able to read, import, or export a REK.

FCS_CKM_EXT.1.3

A REK shall be generated by a RBG in accordance with FCS_RBG_EXT.1.

5.1.1.6 Extended: Cryptographic Key Random Generation (FCS_CKM_EXT.2)

FCS_CKM_EXT.2.1
All DEKs shall be randomly generated with entropy corresponding to the security strength of AES key sizes of [***128, 256***] bits.

5.1.1.7 Extended: Cryptographic Key Generation (FCS_CKM_EXT.3)

FCS_CKM_EXT.3.1
All KEKs shall be [***256-bit***] keys corresponding to at least the security strength of the keys encrypted by the KEK.

FCS_CKM_EXT.3.2
The TSF shall generate all KEKs using one of the following methods:
a) derive the KEK from a Password Authentication Factor using PBKDF and
[***b) generate the KEK using an RBG that meets this profile (as specified in FCS_RBG_EXT.1),***].

5.1.1.8 Extended: Key Destruction (FCS_CKM_EXT.4)

FCS_CKM_EXT.4.1
The TSF shall destroy cryptographic keys in accordance with the specified cryptographic key destruction methods:
- by clearing the KEK encrypting the target key,
- in accordance with the following rules:

o For volatile memory, the destruction shall be executed by a single direct overwrite [***consisting of zeroes***] following by a read-verify.
o For non-volatile EEPROM, the destruction shall be executed by a single direct overwrite consisting of a pseudo random pattern using the TSF's RBG (as specified in FCS_RBG_EXT.1), followed a read-verify.
o For non-volatile flash memory, the destruction shall be executed [***by a single direct overwrite consisting of zeros followed by a read-verify***].
o For non-volatile memory other than EEPROM and flash, the destruction shall be executed by overwriting three or more times with a random pattern that is changed before each write.

FCS_CKM_EXT.4.2
The TSF shall destroy all plaintext keying material and critical security parameters when no longer needed.

5.1.1.9 Extended: TSF Wipe (FCS_CKM_EXT.5)

FCS_CKM_EXT.5.1
The TSF shall wipe all protected data by [***Cryptographically erasing the encrypted DEKs and/or the KEKs in non-volatile memory by following the requirements in FCS_CKM_EXT.4.1;***]

FCS_CKM_EXT.5.2
The TSF shall perform a power cycle on conclusion of the wipe procedure.

5.1.1.10 Extended: Salt Generation (FCS_CKM_EXT.6)

FCS_CKM_EXT.6.1
The TSF shall generate all salts using a RBG that meets FCS_RBG_EXT.1.

5.1.1.11 Cryptographic operation (FCS_COP.1(1))

FCS_COP.1(1).1
The TSF shall perform [encryption/decryption] in accordance with a specified cryptographic algorithm
- AES-CBC (as defined in FIPS PUB 197, and NIST SP 800-38A) mode,
- AES-CCMP (as defined in FIPS PUB 197, NIST SP 800-38C and IEEE 802.11-2012), and

[- ***AES Key Wrap (KW) (as defined in NIST SP 800-38F),***
- ***AES-GCM (as defined in NIST SP 800-38D)***]
and cryptographic key sizes 128-bit key sizes and [***256-bit key sizes***].

5.1.1.12 Cryptographic operation (FCS_COP.1(2))

FCS_COP.1(2).1
The TSF shall perform [cryptographic hashing] in accordance with a specified cryptographic algorithm SHA-1 and [***SHA-256, SHA-384, SHA-512***] and message digest sizes 160 and [***256, 384, 512***] that meet the following: [FIPS Pub 180-4].

5.1.1.13 Cryptographic operation (FCS_COP.1(3))

FCS_COP.1(3).1
The TSF shall perform [cryptographic signature services (generation and verification)] in accordance with a specified cryptographic algorithm
- [RSA schemes] using cryptographic key sizes [of 2048-bit or greater] that meet the following: [FIPS PUB 186-4, 'Digital Signature Standard (DSS)', Section 4]
and [***- [ECDSA schemes] using ['NIST curves' P-256, P-384 and [P-521]] that meet the following: [FIPS PUB 186-4, 'Digital Signature Standard (DSS)', Section 5];***].

5.1.1.14 Cryptographic operation (FCS_COP.1(4))

FCS_COP.1(4).1
The TSF shall perform [keyed-hash message authentication] in accordance with a specified cryptographic algorithm HMAC-SHA-1 and [***HMAC-SHA-256, HMAC-SHA-384, HMAC-SHA-512***] and cryptographic key sizes [**160, 256, 384, 512**] and message digest sizes 160 and [***256, 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'].

5.1.1.15 Cryptographic operation (FCS_COP.1(5))

FCS_COP.1(5).1
The TSF shall perform [Password-based Key Derivation Functions] in accordance with a specified cryptographic algorithm [HMAC-[***SHA-1, SHA-256, SHA-384, SHA-512***]], with [**16384 for SKMM and 32768 for Secure Storage**] iterations, and output cryptographic key sizes [***256***] that meet the following: [NIST SP 800-132].

5.1.1.16 Extended: HTTPS Protocol (FCS_HTTPS_EXT.1)

FCS_HTTPS_EXT.1.1
The TSF shall implement the HTTPS protocol that complies with RFC 2818.
FCS_HTTPS_EXT.1.2
The TSF shall implement HTTPS using TLS (FCS_TLSC_EXT.2).
FCS_HTTPS_EXT.1.3
The TSF shall notify the application and [***no other action***] if the peer certificate is deemed invalid.

5.1.1.17 Extended: Initialization Vector Generation (FCS_IV_EXT.1)

FCS_IV_EXT.1.1
The TSF shall generate IVs in accordance with Table 14: References and IV Requirements for NIST-approved Cipher Modes.

5.1.1.18 Extended: Cryptographic Operation (Random Bit Generation) (FCS_RBG_EXT.1(1))

FCS_RBG_EXT.1(1).1
The TSF shall perform all deterministic random bit generation services in accordance with [***NIST Special Publication 800-90A using [Hash_DRBG (any), HMAC_DRBG (any), CTR_DRBG (AES)]***].

FCS_RBG_EXT.1(1).2
The deterministic RBG shall be seeded by an entropy source that accumulates entropy from [***TSF-hardware-based noise source***] with a minimum of [***256 bits***] of entropy at least equal to the greatest security strength (according to NIST SP 800-57) of the keys and hashes that it will generate.

FCS_RBG_EXT.1(1).3
The TSF shall be capable of providing output of the RBG to applications running on the TSF that request random bits.

5.1.1.19 Extended: Cryptographic Operation (Random Bit Generation) (FCS_RBG_EXT.1(2))

FCS_RBG_EXT.1(2).1
The TSF shall perform all deterministic random bit generation services in accordance with [***FIPS Pub 140-2 Annex C: X9.31 Appendix 2.4 using AES***].

FCS_RBG_EXT.1(2).2
The deterministic RBG shall be seeded by an entropy source that accumulates entropy from [***TSF-hardware-based noise source***] with a minimum of [***128 bits***] of entropy at least equal to the greatest security strength (according to NIST SP 800-57) of the keys and hashes that it will generate.

FCS_RBG_EXT.1(2).3
The TSF shall be capable of providing output of the RBG to applications running on the TSF that request random bits.

5.1.1.20 Extended: Cryptographic Algorithm Services (FCS_SRV_EXT.1)

FCS_SRV_EXT.1.1
The TSF shall provide a mechanism for applications to request the TSF to perform the following cryptographic operations:
- All mandatory and selected algorithms in FCS_CKM.2(1)
- The following algorithms in FCS_COP.1(1): AES-CBC, [***AES Key Wrap, AES-GCM***]
- All mandatory and selected algorithms in FCS_COP.1(3)
- All mandatory and selected algorithms in FCS_COP.1(2)
- All mandatory and selected algorithms in FCS_COP.1(4)

[***- All mandatory and selected algorithms in FCS_CKM.1(1),***
- The selected algorithms in FCS_COP.1(5)].

5.1.1.21 Extended: Cryptographic Key Storage (FCS_STG_EXT.1)

FCS_STG_EXT.1.1
The TSF shall provide [***hardware-isolated***] secure key storage for asymmetric private keys and [***no other keys***].

FCS_STG_EXT.1.2
The TSF shall be capable of importing keys/secrets into the secure key storage upon request of [***the user, the administrator***] and [***applications running on the TSF***].

FCS_STG_EXT.1.3
The TSF shall be capable of destroying keys/secrets in the secure key storage upon request of [***the user***].

FCS_STG_EXT.1.4

The TSF shall have the capability to allow only the application that imported the key/secret the use of the key/secret. Exceptions may only be explicitly authorized by [***a common application developer***].

FCS_STG_EXT.1.5

The TSF shall allow only the application that imported the key/secret to request that the key/secret be destroyed. Exceptions may only be explicitly authorized by [***a common application developer***].

5.1.1.22 Extended: Encrypted Cryptographic Key Storage (FCS_STG_EXT.2)

FCS_STG_EXT.2.1

The TSF shall encrypt all DEKs and KEKs and [***no other keys***] by KEKs that are [***1) Protected by the REK with [a. encryption by a REK,***
b. encryption by a KEK chaining to a REK] ,
2) Protected by the REK and the password with [a. encryption by a REK and the password-derived KEK,
b. encryption by a KEK chaining to a REK and the password-derived KEK]].

FCS_STG_EXT.2.2

DEKs and KEKs and [***no other keys***] shall be encrypted using AES in the [***GCM, CBC mode***].

5.1.1.23 Extended: Integrity of encrypted key storage (FCS_STG_EXT.3)

FCS_STG_EXT.3.1

The TSF shall protect the integrity of any encrypted DEKs and KEKs and [***no other keys***] by [***- [GCM] cipher mode for encryption according to FCS_STG_EXT.2;***].

FCS_STG_EXT.3.2

The TSF shall verify the integrity of the [***hash***] of the stored key prior to use of the key.

5.1.1.24 Extended: EAP TLS Protocol (FCS_TLSC_EXT.1)

FCS_TLSC_EXT.1.1

The TSF shall implement TLS 1.0 and [***no other TLS version***] supporting the following ciphersuites: [- Mandatory Ciphersuites: o TLS_RSA_WITH_AES_128_CBC_SHA as defined in RFC 5246 - [***o TLS_RSA_WITH_AES_256_CBC_SHA as defined in RFC 5246,***
o TLS_DHE_RSA_WITH_AES_128_CBC_SHA as defined in RFC 5246,
o TLS_DHE_RSA_WITH_AES_256_CBC_SHA as defined in RFC 5246,
o TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA as defined in RFC 4492,
o TLS_ECDHE_RSA_WITH_AES_256_CBC_SHA as defined in RFC 4492,
o TLS_ECDHE_ECDSA_WITH_AES_128_CBC_SHA as defined in RFC 4492,
o TLS_ECDHE_ECDSA_WITH_AES_256_CBC_SHA as defined in RFC 4492]].

FCS_TLSC_EXT.1.2

The TSF shall verify that the server certificate presented for EAP-TLS [***chains to one of the specified CAs***].

FCS_TLSC_EXT.1.3

The TSF shall not establish a trusted channel if the peer certificate is invalid.

FCS_TLSC_EXT.1.4

The TSF shall support mutual authentication using X.509v3 certificates.

FCS_TLSC_EXT.1.5

The TSF shall present the Supported Elliptic Curves Extension in the Client Hello with the following NIST curves: [***secp256r1, secp384r1, secp521r1***] and no other curves.

5.1.1.25 Extended: TLS Protocol (FCS_TLSC_EXT.2)

FCS_TLSC_EXT.2.1

The TSF shall implement TLS 1.2 (RFC 5246) supporting the following ciphersuites: [- Mandatory Ciphersuites: o TLS_RSA_WITH_AES_128_CBC_SHA as defined in RFC 5246 - [***o TLS_RSA_WITH_AES_256_CBC_SHA as defined in RFC 5246,***
o TLS_DHE_RSA_WITH_AES_128_CBC_SHA as defined in RFC 5246,
o TLS_DHE_RSA_WITH_AES_256_CBC_SHA as defined in RFC 5246,
o TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA as defined in RFC 4492,
o TLS_ECDHE_RSA_WITH_AES_256_CBC_SHA as defined in RFC 4492,
o TLS_ECDHE_ECDSA_WITH_AES_128_CBC_SHA as defined in RFC 4492,
o TLS_ECDHE_ECDSA_WITH_AES_256_CBC_SHA as defined in RFC 4492,
o TLS_RSA_WITH_AES_128_CBC_SHA256 as defined in RFC 5246,
o TLS_RSA_WITH_AES_256_CBC_ SHA256 as defined in RFC 5246,
o TLS_DHE_RSA_WITH_AES_128_CBC_ SHA256 as defined in RFC 5246,
o TLS_DHE_RSA_WITH_AES_256_CBC_ SHA256 as defined in RFC 5246,
o TLS_ECDHE_ECDSA_WITH_AES_128_CBC_SHA256 as defined in RFC 5289,
o TLS_ECDHE_ECDSA_WITH_AES_256_CBC_SHA384 as defined in RFC 5289]].

FCS_TLSC_EXT.2.2

The TSF shall verify that the presented identifier matches the reference identifier according to RFC 6125.

FCS_TLSC_EXT.2.3

The TSF shall not establish a trusted channel if the peer certificate is invalid.

FCS_TLSC_EXT.2.4

The TSF shall support mutual authentication using X.509v3 certificates.

FCS_TLSC_EXT.2.5

The TSF shall present the Supported Elliptic Curves Extension in the Client Hello with the following NIST curves: [***secp256r1, secp384r1, secp521r1***] and no other curves.

5.1.2 User data protection (FDP)

5.1.2.1 Extended: Security access control (FDP_ACF_EXT.1(1))

FDP_ACF_EXT.1(1).1

The TSF shall provide a mechanism to restrict the system services that are accessible to an application.

FDP_ACF_EXT.1(1).2

The TSF shall provide an access control policy that prevents [***application processes***] from accessing [***all***] data stored by other [***application processes***]. Exceptions may only be explicitly authorized for such sharing by [***a common application developer***].

5.1.2.2 Extended: Security access control (FDP_ACF_EXT.1(2))

FDP_ACF_EXT.1(2).1

The TSF shall provide a mechanism to restrict the system services that are accessible to an application.

FDP_ACF_EXT.1(2).2

The TSF shall provide an access control policy that prevents [***groups of application processes***] from accessing [***all***] data stored by other [***groups of application processes***]. Exceptions may only be explicitly authorized for such sharing by [***the administrator***].

5.1.2.3 Extended: Protected Data Encryption (FDP_DAR_EXT.1)

FDP_DAR_EXT.1.1
Encryption shall cover all protected data.

FDP_DAR_EXT.1.2
Encryption shall be performed using DEKs with AES in the [***CBC, GCM***] mode with key size [***128, 256***] bits.

5.1.2.4 Extended: Sensitive Data Encryption (FDP_DAR_EXT.2)

FDP_DAR_EXT.2.1
The TSF shall provide a mechanism for applications to mark data and keys as sensitive.

FDP_DAR_EXT.2.2
The TSF shall use an asymmetric key scheme to encrypt and store sensitive data received while the product is locked.

FDP_DAR_EXT.2.3
The TSF shall encrypt any stored symmetric key and any stored private key of the asymmetric key(s) used for the protection of sensitive data according to FCS_STG_EXT.2.1 selection 2.

FDP_DAR_EXT.2.4
The TSF shall decrypt the sensitive data that was received while in the locked state upon transitioning to the unlocked state using the asymmetric key scheme and shall re-encrypt that sensitive data using the symmetric key scheme.

5.1.2.5 Extended: Subset information flow control (FDP_IFC_EXT.1)

FDP_IFC_EXT.1.1
The TSF shall [***enable all IP traffic (other than IP traffic required to establish the VPN connection) to flow through the IPsec VPN client***].

5.1.2.6 Extended: User Data Storage (FDP_STG_EXT.1)

FDP_STG_EXT.1.1
The TSF shall provide protected storage for the Trust Anchor Database.

5.1.2.7 Extended: Inter-TSF user data transfer protection (FDP_UPC_EXT.1)

FDP_UPC_EXT.1.1
The TSF provide a means for non-TSF applications executing on the TOE to use TLS, HTTPS, Bluetooth BR/EDR, and [***Bluetooth LE***] to provide a protected communication channel between the non-TSF application and another IT product that is logically distinct from other communication channels, provides assured identification of its end points, protects channel data from disclosure, and detects modification of the channel data.

FDP_UPC_EXT.1.2
The TSF shall permit the non-TSF applications to initiate communication via the trusted channel.

5.1.3 Identification and authentication (FIA)

5.1.3.1 Authentication failure handling (FIA_AFL_EXT.1(1))

FIA_AFL_EXT.1(1).1
The TSF shall detect when a configurable positive integer within [**1-100**] of unsuccessful authentication attempts occur related to last successful **non container** authentication by that user.

FIA_AFL_EXT.1(1).2
When the defined number of unsuccessful **non container** authentication attempts has been surpassed, the TSF shall perform wipe of all protected data.

FIA_AFL_EXT.1(1).3
The TSF shall maintain the number of unsuccessful **non container** authentication attempts that have occurred upon power off.

5.1.3.2 Authentication failure handling (FIA_AFL_EXT.1(2))

FIA_AFL_EXT.1(2).1
The TSF shall detect when a configurable positive integer within [**1-99**] of unsuccessful authentication attempts occur related to last successful **container** authentication by that user.

FIA_AFL_EXT.1(2).2
When the defined number of unsuccessful **container** authentication attempts has been surpassed, the TSF shall perform **a lock of the container or a** wipe of all protected data.

FIA_AFL_EXT.1(2).3
The TSF shall maintain the number of unsuccessful **container** authentication attempts that have occurred upon power off.

5.1.3.3 Extended: Bluetooth User Authorization (FIA_BLT_EXT.1)

FIA_BLT_EXT.1.1
The TSF shall require explicit user authorization before pairing with a remote Bluetooth device.

5.1.3.4 Extended: Bluetooth Authentication (FIA_BLT_EXT.2)

FIA_BLT_EXT.2.1
The TSF shall require Bluetooth mutual authentication between devices prior to any data transfer over the Bluetooth link.

FIA_BLT_EXT.2.2
The TSF shall discard connection attempts from a Bluetooth device address (BD_ADDR) to which a current connection already exists.

5.1.3.5 Extended: PAE Authentication (FIA_PAE_EXT.1)

FIA_PAE_EXT.1.1
The TSF shall conform to IEEE Standard 802.1X for a Port Access Entity (PAE) in the 'Supplicant' role.

5.1.3.6 Extended: Password Management (FIA_PMG_EXT.1)

FIA_PMG_EXT.1.1
The TSF shall support the following for the Password Authentication Factor:
1. Passwords shall be able to be composed of any combination of [***upper and lower case letters***], numbers, and special characters: [***! @ # $ % ^ & * ( ) + = _ / - ' " : ; , ? ` ~ \ | < > { } [ ]***];
2. Password length up to [**16**] characters shall be supported.

5.1.3.7 Extended: Authentication Throttling (FIA_TRT_EXT.1)

FIA_TRT_EXT.1.1
The TSF shall limit automated user authentication attempts by [***enforcing a delay between incorrect authentication attempts***]. The minimum delay shall be such that no more than 10 attempts can be attempted per 500 milliseconds.

5.1.3.8 Protected authentication feedback (FIA_UAU.7)

FIA_UAU.7.1
The TSF shall provide only [obscured feedback to the device's display] to the user while the authentication is in progress.

5.1.3.9 Extended: Authentication for Cryptographic Operation (FIA_UAU_EXT.1)

FIA_UAU_EXT.1.1
The TSF shall require the user to present the Password Authentication Factor prior to decryption of protected data and encrypted DEKs, KEKs and [***no other keys***] at startup.

5.1.3.10 Extended: Timing of Authentication (FIA_UAU_EXT.2)

FIA_UAU_EXT.2.1
The TSF shall allow [***[enter password to unlock, make emergency calls, receive calls, take pictures and screen shots (automatically named stored internally by the TOE), turn the TOE off, restart the TOE, enable emergency mode, enable and disable airplane mode, see notifications, configure sound/vibrate/mute, set the volume (up and down) for various sound categories, and see the configured banner]***] on behalf of the user to be performed before the user is authenticated.

FIA_UAU_EXT.2.2
The TSF shall require each user to be successfully authenticated before allowing any other TSF-mediated actions on behalf of that user.

5.1.3.11 Extended: Re-Authentication (FIA_UAU_EXT.3)

FIA_UAU_EXT.3.1
The TSF shall require the user to enter the correct Password Authentication Factor when the user changes the Password Authentication Factor, and following TSF- and user-initiated locking in order to transition to the unlocked state, and [***[when changing the lock screen, when changing one's password, and when initializing the keystore]***].

5.1.3.12 Extended: Validation of certificates (FIA_X509_EXT.1)

FIA_X509_EXT.1.1
The TSF shall validate certificates in accordance with the following rules:
- RFC 5280 certificate validation and certificate path validation.
- The certificate path must terminate with a certificate in the Trust Anchor Database.
- The TSF shall validate a certificate path by ensuring the presence of the basicConstraints extension and that the CA flag is set to TRUE for all CA certificates.
- The TSF shall validate the revocation status of the certificate using [***a Certificate Revocation List (CRL) as specified in RFC 5759***].
- The TSF shall validate the extendedKeyUsage field according to the following rules:

o Certificates used for trusted updates and executable code integrity verification shall have the Code Signing purpose (id-kp 3 with OID 1.3.6.1.5.5.7.3.3) in the extendedKeyUsage field.
o Server certificates presented for TLS shall have the Server Authentication purpose (id-kp 1 with OID 1.3.6.1.5.5.7.3.1) in the extendedKeyUsage field.
o (Conditional) Server certificates presented for EST shall have the CMC Registration Authority (RA) purpose (id-kp-cmcRA with OID 1.3.6.1.5.5.7.3.28) in the extendedKeyUsage field.

FIA_X509_EXT.1.2
The TSF shall only treat a certificate as a CA certificate if the basicConstraints extension is present and the CA flag is set to TRUE.

5.1.3.13 Extended: X509 certificate authentication (FIA_X509_EXT.2)

FIA_X509_EXT.2.1
The TSF shall use X.509v3 certificates as defined by RFC 5280 to support authentication for EAP-TLS exchanges, and [***IPsec, TLS, HTTPS***], and [***no additional uses***].

FIA_X509_EXT.2.2
When the TSF cannot establish a connection to determine the validity of a certificate, the TSF shall [***not accept the certificate***].

5.1.3.14 Extended: Request Validation of certificates (FIA_X509_EXT.3)

FIA_X509_EXT.3.1
The TSF shall provide a certificate validation service to applications.

FIA_X509_EXT.3.2
The TSF shall respond to the requesting application with the success or failure of the validation.

5.1.4 Security management (FMT)

5.1.4.1 Extended: Management of security functions behavior (FMT_MOF_EXT.1)

FMT_MOF_EXT.1.1
The TSF shall restrict the ability to perform the functions in column 3 of ~~Table 1~~ to the user.

FMT_MOF_EXT.1.2
The TSF shall restrict the ability to perform the **non container** functions in column 5 of ~~Table 1~~ to the administrator when the device is enrolled and according to the administrator-configured policy.

FMT_MOF_EXT.1.3
The TSF shall restrict the ability to perform the container functions in column 7 of ~~Table 1~~ to the administrator when the device is enrolled and according to the administrator-configured policy.

5.1.4.2 Extended: Specification of Management Functions (FMT_SMF_EXT.1)

FMT_SMF_EXT.1.1
The TSF shall be capable of performing the ~~following management~~ functions~~:~~ **in column 2 of Table 2 Security Management Functions**.

FMT_SMF_EXT.1.2
The TSF shall be capable of allowing the administrator to perform the non container functions in column 4 of Table 2 Security Management Functions.

FMT_SMF_EXT.1.3
The TSF shall be capable of allowing the administrator to perform the container functions in column 6 of Table 2 Security Management Functions.

Management Function **Status Markers:** M – Mandatory I – Implemented	FMT_SMF_EXT.1.1	FMT_MOF_EXT.1.1	FMT_SMF_EXT.1.2	FMT_MOF_EXT.1.2	FMT_SMF_EXT.1.3	FMT_MOF_EXT.1.3
1. configure password policy: a. minimum password length b. minimum password complexity c. maximum password lifetime The administrator can configure the required password characteristics (minimum length, complexity, and lifetime) using the Android MDM APIs. There are distinct settings for the passwords used to unlock the base device and Knox container.	M		M	M	I	I
2. configure session locking policy: a. screen-lock enabled/disabled b. screen lock timeout c. number of authentication failures The administrator can configure the session locking policy using the Android MDM APIs. There are distinct settings for base device and Knox container inactivity. The user can also adjust each of the session locking policiesfor the base device and Knox container; however, if set by the administrator, the user can only set a more strict policy (e.g., setting the device to allow fewer authentication failures than configured by the administrator).	M		M	M	I	I
3. enable/disable the VPN protection: a. across device [**c. *no other method***] The user can configure and then enable the TOE's VPN to protect traffic. The administrator (through an MDM Agent that utilizes the TOE's MDM APIs) can restrict the TOE's ability to connect to a VPN.	M		I	I		
4. enable/disable [**nfc, Bluetooth, Wi-Fi, and cellular radios**] The administrator can disable the radios using the TOE's MDM APIs. Once disabled, a user cannot enable the radio. The TOE's radio's operate at frequencies of 2.4 GHz (NFC/Bluetooth), 2.4/5 GHz (Wi-Fi), and 850 MHz (4G/LTE).	M		I	I		
5. enable/disable [**camera, microphone**]: a. across device	M		M	M	I	I

[**c.** ***no other method***] An administrator may configure the TOE (through an MDM agent utilizing the TOE's MDM APIs) to turn off the camera and or microphones. If the administrator has disabled either the camera or the microphones, then the user cannot use those capture devices. The administrator can also disable the use of the camera or microphone inside a Knox container without affecting access to those devices when outside the container.						
6. specify wireless networks (SSIDs) to which the TSF may connect An administrator can specify a list of wireless network SSIDs to which the TOE may connect and can restrict the TOE to only allow a connection to the specified SSIDs.	M		M	I		
7. configure security policy for each wireless network: a. [***specify the CA(s) from which the TSF will accept WLAN authentication server certificate(s)***] b. security type c. authentication protocol d. client credentials to be used for authentication Both users and administrators (using the TOE's MDM APIs) can define wireless connection policies for specific SSIDs.	M		M			
8. transition to the locked state Both users and administrators (using the TOE's MDM APIs) can transition the TOE or KNOX container into a locked state.	M		M	-	I	
9. full wipe of protected data Both users and administrators (using the TOE's MDM APIs) can force the TOE to perform a full wipe (factory reset) of data. Administrators can also cause the contents of a Knox container to be wiped by deleting the container.	M		M		I	
10. configure application installation policy by [selection: ***a. specifying authorized application repository(s),*** ***c. denying installation of applications***] The administrator using the TOE's MDM APIs can configure the TOE so that applications cannot be installed and can also block the use of the Google Market Place. There are distinct settings for disabling the installation of applications for the base device and Knox container.	M		M	M	I	I
11. import keys/secrets into the secure key storage Both users and administrators (using the TOE's MDM APIs) can import secret keys into the secure key storage.	M		I			
12. destroy imported keys/secrets and [***no other keys/secrets***] in the secure	M		I			

key storage Both users and administrators (using the TOE's MDM APIs) can destroy secret keys in the secure key storage.						
13. import X.509v3 certificates into the Trust Anchor Database Both users and administrators (using the TOE's MDM APIs) can import X.509v3 certificates into the Trust Anchor Database.	M		M			
14. remove imported X.509v3 certificates and [***no other X.509v3 certificates***] in the Trust Anchor Database Both users and administrators (using the TOE's MDM APIs) can remove imported X.509v3 certificates from the Trust Anchor Database.	M		I			
15. enroll the TOE in management TOE users can enroll the TOE in management according to the instructions specific to a given MDM. Presumably any enrollment would involve at least some user functions (e.g., install an MDM agent application) on the TOE prior to enrollment.	M	M				
16. remove applications Both users and administrators (using the TOE's MDM APIs) can uninstall applications on the TOE (non-container) and applications inside a KNOX container.	M		M		I	
17. update system software Users can check for updates and cause the device to update if an update is available. A wide range of possibilities exist for the administrator since and using the MDM APIs the administrator can query the version of the TOE and installed applications and an MDM agent on the TOE could issues pop-ups, initiate updates, block communication, etc. until any necessary updates are completed.	M		M			
18. install applications Both users and administrators (using the TOE's MDM APIs) can install applications on the TOE (non-container) and applications inside a KNOX container.	M		M		I	
19. remove Enterprise applications Both users and administrators (using the TOE's MDM APIs) can uninstall applications on the TOE (non-container) and applications inside a KNOX container. Note that there is no distinction between Enterprise and other applications.	M		M		I	
20. configure the Bluetooth trusted channel: a. disable/enable the Discoverable mode (for BR/EDR) b. change the Bluetooth device name [**c.** ***allow/disallow additional wireless technologies to be used with***	M					

Bluetooth] TOE users can enable Bluetooth discoverable mode for a short period of time and can also change the device name which is used for the Bluetooth name. Additional wireless technologies include Android Beam and S-Beam which are related to NFC and can be enabled and disabled by the TOE user.						
21. enable/disable display notification in the locked state of: [**f.** ***all notifications***] TOE users can configure the TOE to allow or disallow notifications while in a locked state. The TOE user can also configure a Knox container to allow or disallow contact and calendar notifications.	M					
22. enable/disable all data signaling over [**USB**] The administrator (using the TOE's MDM APIs) can disable USB communication features (tethering, debugging, mass storage access, etc.) to prevent data signaling via USB.	I		I	I		
23. enable/disable [**Wi-Fi tethering, USB tethering, and Bluetooth tethering**] The administrator (using the TOE's MDM APIs) can individually disable each tethering method. Unless disabled by the administrator, TOE users can individually enable and disable tethering via a Wi-Fi hotspot, USB connection, and Bluetooth pairing. The TOE acts as a server (acting as an access point, a USB Ethernet adapter, and as a Bluetooth Ethernet adapter respectively) in order to share its network connection with another device.	I		I	I		
24. enable/disable developer modes The administrator (using the TOE's MDM APIs) can disable Developer Mode. Unless disabled by the administrator, TOE users can enable and disable Developer Mode.	I		I	I		
25. enable data-at rest protection The administrator (using the TOE's MDM APIs) can configure a device encryption policy. Once enabled the TOE internal flash will be encrypted. Users can also encrypt the TOE internal flash even if not enforced by administrator policy. However, if the policy is set the user cannot decrypt the TOE.	I		I	I		
26. enable removable media's data-at-rest protection The administrator (using the TOE's MDM APIs) can configure a SD card encryption policy. Once enabled, SD cards can be used only after they are	I		I	I		

encrypted. Users can also encrypt the SD cards even if not enforced by administrator policy. However, if the policy is set the user cannot use an SD card unless it is encrypted first and cannot decrypt an SD card.						
27. enable/disable bypass of local user authentication						
28. wipe Enterprise data						
29. approve [selection: *import, removal*] by applications of X.509v3 certificates in the Trust Anchor Database						
30. configure whether to establish a trusted channel or disallow establishment if the TSF cannot establish a connection to determine the validity of a certificate						
31. enable/disable the cellular protocols used to connect to cellular network base stations						
32. read audit logs kept by the TSF						
33. configure [selection: *certificate, public-key*] used to validate digital signature on applications						
34. approve exceptions for shared use of keys/secrets by multiple applications						
35. approve exceptions for destruction of keys/secrets by applications that did not import the key/secret						
36. configure the unlock banner The administrator (using the TOE's MDM APIs) can defined a banner of a maximum of 256 characters to be displayed while the TOE is locked. There is no method for the user to change the banner.	I		I	I		
37. configure the auditable items						
38. retrieve TSF-software integrity verification values						
39. enable/disable [a. ***USB mass storage mode***] The administrator (using the TOE's MDM APIs) can disable USB mass storage mode.	I		I	I		
40. enable/disable backup to [***remote system***] The TOE user can configure and enable/disable a Google backup feature to backup application data.	I					
41. enable/disable [a. ***Hotspot functionality authenticated by [pre-shared key],*** b. ***USB tethering authenticated by [no authentication]***] The administrator (using the TOE's MDM APIs) can disable the Wi-Fi hotspot and USB tethering. Unless disabled by the administrator, TOE users can configure the Wi-Fi hotspot with a pre-shared key and can configure USB tethering (with no authentication).	I		I	I		

42. approve exceptions for sharing data between [selection: *application processes, groups of application processes*]						
43. place applications into application process groups based on [assignment: *application characteristics*]						
44. enable/disable location services: a. across device [**c.** ***no other method***] The administrator (using the TOE's MDM APIs) can disable location services. Unless disabled by the administrator, TOE users can enable and disable location services.	M		I	I		
45. [**enable/disable voice command control of device functions**] The administrator (using the TOE's MDM APIs) can disable S-Voice. Unless disabled by the administrator, TOE users can start the S-Voice application in order to issue voice commands. They can also disable that function by terminating the S-Voice application.	I		I	I		

Table 2 Security Management Functions

5.1.4.3 Extended: Specification of Remediation Actions (FMT_SMF_EXT.2(1))

FMT_SMF_EXT.2(1).1
The TSF shall offer [***wipe of protected data, wipe of sensitive data, remove Enterprise applications, [remove MDM policies and disable CC mode]***] upon **non container** unenrollment and [***no other triggers***].

5.1.4.4 Extended: Specification of Remediation Actions (FMT_SMF_EXT.2(2))

FMT_SMF_EXT.2(2).1
The TSF shall offer [***[remove Enterprise applications, full wiper of container data]***] upon **container** unenrollment and [***no other triggers***].

5.1.5 Protection of the TSF (FPT)

5.1.5.1 Extended: Anti-Exploitation Services (ASLR) (FPT_AEX_EXT.1)

FPT_AEX_EXT.1.1
The TSF shall provide address space layout randomization (ASLR) to applications.

FPT_AEX_EXT.1.2
The base address of any user-space memory mapping will consist of at least 8 unpredictable bits.

5.1.5.2 Extended: Anti-Exploitation Services (Memory Page Permissions) (FPT_AEX_EXT.2)

FPT_AEX_EXT.2.1
The TSF shall be able to enforce read, write, and execute permissions on every page of physical memory.

5.1.5.3 Extended: Anti-Exploitation Services (Overflow Protection) (FPT_AEX_EXT.3)

FPT_AEX_EXT.3.1

TSF processes that execute in a non-privileged execution domain on the application processor shall implement stack-based buffer overflow protection.

5.1.5.4 Extended: Domain Isolation (FPT_AEX_EXT.4)

FPT_AEX_EXT.4.1

The TSF shall protect itself from modification by untrusted subjects.

FPT_AEX_EXT.4.2

The TSF shall enforce isolation of address space between applications.

5.1.5.5 Application Processor Mediation (FPT_BBD_EXT.1)

FPT_BBD_EXT.1.1

The TSF shall prevent code executing on any baseband processor (BP) from accessing application processor (AP) resources except when mediated by the AP.

5.1.5.6 Extended: Key Storage (FPT_KST_EXT.1)

FPT_KST_EXT.1.1

The TSF shall not store any plaintext key material in readable non-volatile memory.

5.1.5.7 Extended: No Key Transmission (FPT_KST_EXT.2)

FPT_KST_EXT.2.1

The TSF shall not transmit any plaintext key material outside the security boundary of the TOE.

5.1.5.8 Extended: No Plaintext Key Export (FPT_KST_EXT.3)

FPT_KST_EXT.3.1

The TSF shall ensure it is not possible for the TOE user(s) to export plaintext keys.

5.1.5.9 Extended: Self-Test Notification (FPT_NOT_EXT.1)

FPT_NOT_EXT.1.1

The TSF shall transition to non-operational mode and [***no other actions***] when the following types of failures occur:
- failures of the self-test(s)
- TSF software integrity verification failures
- [***no other failures***].

5.1.5.10 Reliable time stamps (FPT_STM.1)

FPT_STM.1.1

The TSF shall be able to provide reliable time stamps for its own use.

5.1.5.11 Extended: TSF Cryptographic Functionality Testing (FPT_TST_EXT.1)

FPT_TST_EXT.1.1

The TSF shall run a suite of self-tests during initial start-up (on power on) to demonstrate the correct operation of all cryptographic functionality.

5.1.5.12 Extended: TSF Integrity Testing (FPT_TST_EXT.2)

FPT_TST_EXT.2.1

The TSF shall verify the integrity of the bootchain up through the Application Processor OS kernel, and [

Name	Path
charon	/system/bin
dalvikvm	/system/bin
icd	/system/bin
keystore	/system/bin
odekeymgr	/system/bin
qrngd	/system/bin
qseecomd	/system/bin
sfotahelper	/system/bin
time_daemon	/system/bin
vold	/system/bin
wpa_supplicant	/system/bin
libSecurityManagerNative.so	/system/lib
libcharon.so	/system/lib
libcrypto.so	/system/lib
libdirencryption.so	/system/lib
libhydra.so	/system/lib
libjavacrypto.so	/system/lib
libkeystore_binder.so	/system/lib
libkeyutils.so	/system/lib
libsec_devenc.so	/system/lib
libsec_ecryptfs.so	/system/lib
libsec_ode_km.so	/system/lib
libskmm.so	/system/lib
libskmm_helper.so	/system/lib
libsoftkeymaster.so	/system/lib
libssl.so	/system/lib
libstrongswan.so	/system/lib
libvstr.so	/system/lib

], stored in mutable media prior to its execution through the use of [***a digital signature using a hardware-protected asymmetric key, a hardware-protected hash***].

5.1.5.13 Extended: Trusted Update: TSF version query (FPT_TUD_EXT.1)

FPT_TUD_EXT.1.1

The TSF shall provide authorized users the ability to query the current version of the TOE firmware/software.

FPT_TUD_EXT.1.2

The TSF shall provide authorized users the ability to query the current version of the hardware model of the device.

FPT_TUD_EXT.1.3

The TSF shall provide authorized users the ability to query the current version of installed mobile applications.

5.1.5.14 Extended: Trusted Update Verification (FPT_TUD_EXT.2)

FPT_TUD_EXT.2.1

The TSF shall verify software updates to the Application Processor system software and [***[communications processor software, bootloader software, carrier specific configuration]***] using a digital signature by the manufacturer prior to installing those updates.

FPT_TUD_EXT.2.2

The TSF shall [***update only by verified software***] the TSF boot integrity [***key, hash***].

FPT_TUD_EXT.2.3

The TSF shall verify that the digital signature verification key used for TSF updates [***matches a hardware-protected public key***].

FPT_TUD_EXT.2.4

The TSF shall verify mobile application software using a digital signature mechanism prior to installation.

5.1.6 TOE access (FTA)

5.1.6.1 Extended: TSF- and User-initiated locked state (FTA_SSL_EXT.1)

FTA_SSL_EXT.1.1

The TSF shall transition to a locked state after a time interval of inactivity.

FTA_SSL_EXT.1.2

The TSF shall transition to a locked state after initiation by either the user or the administrator.

FTA_SSL_EXT.1.3

The TSF shall, upon transitioning to the locked state, perform the following operations:
a) clearing or overwriting display devices, obscuring the previous contents;
b) [**no other actions**].

5.1.6.2 Default TOE Access Banners (FTA_TAB.1)

FTA_TAB.1.1

Before establishing a user session, the TSF shall display an advisory warning message regarding unauthorized use of the TOE.

5.1.6.3 Extended: Wireless Network Access (FTA_WSE_EXT.1)

FTA_WSE_EXT.1.1

The TSF shall be able to attempt connections to wireless networks specified as acceptable networks as configured by the administrator in FMT_SMF_EXT.1.

5.1.7 Trusted path/channels (FTP)

5.1.7.1 Extended: Trusted channel Communication (FTP_ITC_EXT.1)

FTP_ITC_EXT.1.1

The TSF shall use 802.11-2012, 802.1X, and EAP-TLS and [***IPsec and TLS***] to provide a communication channel between itself and another trusted IT product that is logically distinct from other communication channels, provides assured identification of its end points, protects channel data from disclosure, and detects modification of the channel data.

FTP_ITC_EXT.1.2

The TSF shall permit the TSF to initiate communication via the trusted channel.

FTP_ITC_EXT.1.3

The TSF shall initiate communication via the trusted channel for wireless access point connections, administrative communication, configured enterprise connections, and [***no other connections***].

5.2 TOE Security Assurance Requirements

The SARs for the TOE are the EAL 1 augmented with ALC_TSU_EXT.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
	ALC_TSU_EXT.1: Timely Security Updates
ATE: Tests	ATE_IND.1: Independent testing - conformance
AVA: Vulnerability assessment	AVA_VAN.1: Vulnerability survey

Table 3 EAL 1 augmented with ALC_TSU_EXT.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 categorisation 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.2.1d

The developer shall provide a configuration list for the TOE.

ALC_CMS.2.1c

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

ALC_CMS.2.2c

The configuration list shall uniquely identify the configuration items.

ALC_CMS.2.1e

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

5.2.3.3 Timely Security Updates (ALC_TSU_EXT.1)

ALC_TSU_EXT.1.1d

The developer shall provide a description in the TSS of how timely security updates are made to the TOE.

ALC_TSU_EXT.1.1c

The description shall include the process for creating and deploying security updates for the TOE software/firmware.

ALC_TSU_EXT.1.2c

The description shall express the time window as the length of time, in days, between public disclosure of a vulnerability and the public availability of security updates to the TOE.

ALC_TSU_EXT.1.3c

The description shall include the mechanisms publicly available for reporting security issues pertaining to the TOE.

ALC_TSU_EXT.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
- TOE access
- Trusted path/channels

6.1 Cryptographic support

FCS_CKM.1(1): The TOE supports asymmetric key generation for all types. The TOE generates RSA and DH/ECDH/ECDSA (including P-521) keys in its OpenSSL library (which generates the keys in accordance with ANSI X9.31 and FIPS 186-4 respectively). The TOE supports generating keys with a security strength of 112-bits and larger, thus supports 2048-bit RSA and DH keys, and 256-bit ECDH/ECDSA keys. The TOE has RSA certificate #960 and ECDSA certificate # 264 for its RSA and ECDSA key generation implementations.

FCS_CKM.1(2): The TOE adheres to 802.11-2012 for key generation. The TOE's wpa_supplicant provides the PRF384 for WPA2 derivation of 128-bit AES Temporal Key and also employs its OpenSSL AES-256 DRBG when generating random values use in the EAP-TLS and 802.11 4-way handshake. The TOE has been designed for compliance, and the Device has successfully completed certification (including WPA2 Enterprise) and received a WiFi CERTIFIED Interoperability Certificate from the WiFi Alliance (Certification ID: WFA55732). The WiFi Alliance maintains a website providing further information about the testing program: http://www.wi-fi.org/certification

FCS_CKM.2(1): The TOE supports RSA (800-56B), DHE (FFC 800-56A), and ECDHE (ECC 800-56A) methods in TLS key establishment/exchange. The TOE has CVL algorithm certificate #10 and ECDSA algorithm certificate #264 for Elliptic Curve key establishment and key generation respectively. The TOE has RSA algorithm certificate #960 for RSA key generation and for RSA key establishment. The user and administrator need take no special configuration of the TOE as the TOE automatically generates the keys needed for negotiated TLS ciphersuite. Because the TOE only acts as a TLS client, the TOE only performs 800-56B encryption (specifically the encryption of the Pre-Master Secret using the Server's RSA public key) when participating in TLS_RSA_* based TLS handshakes. Thus, the TOE does not perform 800-56B decryption. However, the TOE's TLS client correctly handles other cryptographic errors (for example, invalid checksums, incorrect certificate types, corrupted certificates) by sending a TLS fatal alert.

FCS_CKM.2(2): The TOE adheres to RFC 3394, SP 800-38F, and 802.11-2012 standards and unwraps the GTK (sent encrypted with the WPA2 KEK using AES Key Wrap in an EAPOL-Key frame). The TOE, upon receiving an EAPOL frame, will subject the frame to a number of checks (frame length, EAPOL version, frame payload size, EAPOL-Key type, key data length, EAPOL-Key CCMP descriptor version, and replay counter) to ensure a proper EAPOL message and then decrypt the GTK using the KEK, thus ensuring that it does not expose the Group Temporal Key (GTK).

FCS_CKM_EXT.1: The TOE supports a Root Encryption Key (REK) within the main (application) processor. Requests for encryption or decryption chaining to the REK are only accessible through the Trusted Execution Environment, or TEE (TrustZone). The REK lies in a series of 256-bit fuses, which the TOE programs during manufacturing. The TEE does not allow direct access to the REK but provides access to a derived HEK (Hardware Encryption Key) which is derived from the REK through a KDF function for encryption and decryption.

The REK value is generated during manufacturing either by the TOE (if it detects that the REK fuses have not been set) using it hardware DRBG or is generated during fabrication using an external RBG that meets the requirements of this PP in that the process utilizes a SHA-256 Hash_DRBG seeded by a hardware entropy source identical in architecture to that within the TOE. This fabrication process includes strict controls to ensure that fabricator cannot access any REK values between generation and programming.

FCS_CKM_EXT.2: The TOE supports Data Encryption Key (DEK) generation using its approved RBGs. The TOE generates AES 256-bit DEKs using its OpenSSL CTR_DRBG in response to an application through either the Android Java API or through the native C API. This is used, for example, by the ODE (Onboard Device Encryption, which refers to the TOE's encryption of the user data partition) system using the native C API as well as by the media encryption system application to generate a master key for protection of File Encryption Keys (FEKs) for individual files.

The TOE also generates DEKs by using its ANSI X9.31 A.2.4 AES-128 RNG (contained in the kernel crypto module). The removable media encryption system of the TOE utilizes this function to generate FEKs to encrypt individual files on the removable media.

FCS_CKM_EXT.3: The TOE generates KEKs (which are always AES 256-bit keys generated by the CTR_DRBG) through a combination of methods. First, the module generates a KEK (denoted as a Domain KEK, or DKEK) for each user of the phone, for each Domain (where a Domain provides separation of application data, e.g., "Corporate" or "Home"). Note that the current TOE only supports a single Domain for each user; however future versions of the TOE may support multiple Domains.

To protect the user's DKEK, the TOE first combines a key derived from the user's password with PBKDF2 with a random SALT as well as a unique Domain string. The resulting ephemeral PKEK (password key encryption key) is then used to protect the randomly generated DKEK, the second factor being the REK-derived HEK. The DKEK is encrypted with AES256 GCM inside the TEE, to provide both security and integrity for storage. This process is reversed to access the DKEK.

The TOE generates a number of different KEKs. In addition to the DKEK, applications may request key generation (either through the Java or native C API), and the TOE utilizes the same RBG to satisfy those requests. The requesting application ultimately chooses whether to use that key as a DEK or a KEK (and can choose whether it wishes that key to be 128 or 256-bits), but it is worth mentioning here, as an application can utilize such a key as a KEK, should it choose.

In addition to those KEKs, the TOE also derives several other ephemeral KEKs. The HEK is derived directly from the REK using an AES_CMAC based KDF to derive the 256-bit HEK. The PKEK2 is derived from the user's password combined with Salt (again, taken from the AES-256 CTR_DRBG) forming PKEK1, which is then put through an SP 800-108 KDF using a specific Domain value to derive PKEK2. Finally, the TOE can derive a KEK for an application by using an SP 800-108 KDF to combine the DKEK with a string provided by the application to form KEK1. In each of these operations, the key management scheme ensures preservation of effective entropy.

FCS_CKM_EXT.4: The TOE destroys cryptographic keys when they are no longer in use by the system. The exceptions to this are public keys (that protect the boot chain and software updates) and the REK, which are never cleared. Keys stored in RAM for immediate use are destroyed by a zero overwrite. Keys stored in Flash (i.e. eMMC) are destroyed by cryptographic erasure through a BLKSECDISCARD command (as defined in the JEDEC eMMC 5.0 specification, JESD84-B50) to the flash controller for the location where the ODE and removable media keys are stored. Once these are erased, the keys stored within the encrypted data partition of the TOE are considered cryptographically erased.

FCS_CKM_EXT.5: The TOE provides a TOE Wipe function that first erases the ODE DEK used to encrypt the entire data partition using a Secure Trim command to ensure that TOE writes zeros to the eMMC blocks containing the ODE DEK (the data partition footer contains the ODE DKEK and ODE DEK as well as the SDCard DKEK and SDCard MK). After overwriting that, the TOE will reformat the partition (though that does not matter as one can no longer retrieve the data at that point). Upon completion of reformatting the Flash partition holding user data, the TOE will perform a power-cycle.

FCS_CKM_EXT.6: The TOE creates salt and nonces (which are just salt values used in WPA2) using its AES-256 CTR_DRBG.

FCS_COP.1: The TOE performs cryptographic algorithms in accordance with the following NIST standards and has received the following CAVP algorithm certificates.

The OpenSSL FIPS Object Module provides the following algorithms

Algorithm	NIST Standard	SFR Refereence	Cert#
AES 128/256 CBC, CCM, GCM, KW	FIPS 197, SP 800-38A/C/D/F	FCS_COP.1(1)	3313
CVL ECC CDH P-224/256/384/521	SP 800-56A	FCS_CKM.1(1)	476
CVL RSADP	SP 800-56B	FCS_CKM.1(1)	477
DRBG Hash/HMAC/CTR/Dual_EC	SP 800-90A	FCS_RBG_EXT.1(1)	760

ECDSA PKG/PKV/SigGen/SigVer	FIPS 186-4	FCS_CKM.1(1)
		FCS_CKM.1(2)
		FCS_COP.1(3)
			644

HMAC SHA-1/256/384/512	FIPS 198-1 & 180-4	FCS_COP.1(4)	2105

RSA SIG(gen)/SIG(ver)/Key(gen)	FIPS 186-2	FCS_CKM.1(1)
		FCS_CKM.1(2)
		FCS_COP.1(3)
			1695

SHS SHA-1/256/384/512	FIPS 180-4	FCS_COP.1(2)	2746

Table 4 OpenSSL Cryptographic Algorithms

The Samsung Kernel Cryptographic (“Kernel Crypto”) Module provides the following algorithms

Algorithm	NIST Stadnard	SFR Refereence	Cert#
AES 128/256 CBC	FIPS 197, SP 800-38A	FCS_COP.1(1)	3311
HMAC SHA-1/256/384/512	FIPS 198-1 & 180-4	FCS_COP.1(4)	2104
RNG ANSI X9.31 AES-128	ANSI X9.31 & 931rngext.pdf	FCS_RBG_EXT.1(2)	1357
SHS SHA-1/256/384/512	FIPS 180-4	FCS_COP.1(2)	2745

Table 5 Samsung Kernel Cryptographic Algorithms

The TOE’s application processors include hardware entropy implementations that supply random data within the TEE and to the Linux kernel RNG (/dev/random).

Note that kernel-space system applications utilize the cryptographic algorithm implementations in the Samsung Kernel Cryptographic Module (Kernel Crypto), while user-space system applications and mobile applications utilize the OpenSSL module (either through a Java API or through the native C API). In the case of each cryptographic module, the module itself includes any algorithms required (for example, OpenSSL provides hash functions for use by HMAC and digital signature algorithms).

For its HMAC implementations, the TOE accepts all key sizes of 160, 256, 384, & 512; supports all SHA sizes save 224 (e.g., SHA-1, 256, 384, & 512), utilizes the specified block size (512 for SHA-1 and 256, and 1024 for SHA-384 & 512), and output MAC lengths of 160, 256, 384, and 512.

The TOE conditions the user's password exactly as per SP 800-132 (and thus as per SP 800-197-1) with no deviations by using PBKDF2 with 16,384 HMAC-SHA-512 iterations to combine a 128-bit salt with the user’s password. The time needed to derive keying material does not matter as the combination of the password derived KEK with REK value entirely prevents offline attacks and the TOE’s maximum incorrect login attempts prevents exhaustive online attacks.

FCS_HTTPS_EXT.1: The TOE includes the ability to support the HTTPS protocol (compliant with RFC 2818) so that (mobile and system client) applications executing on the TOE can securely connect to external servers using HTTPS. Administrators have no credentials and cannot use HTTPS or TLS to establish administrative sessions with the TOE as the TOE does not provide any such capabilities.

FCS_IV_EXT.1: The TOE generates IVs for data storage encryption and for key storage encryption. The TOE uses AES-CBC mode for data encryption and AES-GCM for key storage.

FCS_RBG_EXT.1(*): The TOE provides a number of different RBGs including

1. A SHA-256 Hash_DRBG provided in the hardware of the Qualcomm Application Processor.
2. RBGs provided by OpenSSL: Hash_DRBG (using any size SHA), HMAC_DRBG (again using size SHA), and CTR_DRBG (using AES). Note that while the TOE includes implementations of three DRBG variants (and supports all options within each variant), the TOE (and its current system level applications) make use of only an AES-256 CTR_DRBG. Furthermore The TOE provides mobile applications access (through an Android Java API) to random data drawn from its AES-256 CTR_DRBG. However, future (system-level) applications could utilize other DRBG variants supported by the TOE’s OpenSSL module.
3. And an ANSI X9.31 AES-128 RBG provided by Kernel Crypto

The TOE ensures that it initializes each RBG with sufficient entropy ultimately accumulated from a TOE-hardware-based noise source. The TOE uses its hardware-based noise source to continuously fill /dev/random with random data with full entropy, and in turn, the TOE draws from /dev/random to seed both its AES-256 CTR_DRBG and ANSI X9.31 AES-128 RBG. The TOE seeds its AES-256 CTR_DRBG using 384-bits of data from /dev/random, thus ensuring at least 256-bits of entropy. Finally the TOE seeds its X9.31 AES-128 with 256-bits of data from /dev/random, also ensuring that it contains at least 128-bits of entropy.

FCS_SRV_EXT.1: The TOE provides applications access to the cryptographic operations including encryption (AES), hashing (SHA), signing and verification (RSA & ECDSA), key hashing (HMAC), password-based key-derivation functions (PKBDFv2 HMAC-SHA-512), generate asymmetric keys for key establishment (RSA, DH, and ECDH), and generate asymmetric keys for signature generation and verification (RSA, ECDSA). The TOE provides access through the Android operating system’s Java API, through the native OpenSSL API, and through the kernel. The vendor also developed testing applications to enable execution of the NIST algorithm testing suite in order to verify the correctness of the algorithm implementations.

FCS_STG_EXT.1: The TOE provides for secure key storage for asymmetric keys. For these asymmetric keys, the TOE secures a series of public keys used for Secure Boot through a chain of trust that operates as follows. The Application Processor (AP) contains the SHA-256 hash of the Secure Boot Public Key (an RSA 2048-bit key embedded in end of the signed bootloader image), and upon verifying the SBPK attached to the bootloader produces the expected hash, the AP uses this public key to verify the PKCS 1.5 RSA 2048 w/ SHA-256 signature of the bootloader image, to ensure its integrity and authenticity before transitioning execution to the bootloader. The bootloader, in turn, contains the Image Signing Public Key (ISPK), which the bootloader will use to verify the PKCS 1.5 RSA 2048 w/ SHA-1 signature on either kernel image (primary kernel image or recovery kernel image). Note that when configured for Common Criteria mode, the TOE only accepts updates to the TOE firmware Over The Air; however, when not configured for CC mode, the TOE allows updates through the bootloader’s ODIN mode. The primary kernel includes an embedded FOTA Public Key, which the TOE uses to verify the authenticity and integrity of Firmware Over-The-Air update signatures (which contain a PKCS 2.1 PSS RSA 2048 w/ SHA-512 signature).

In addition to the internal public keys protected by the chain-of-trust with a hardware root/anchor, the TOE also provides users and applications running on the TOE the ability to import keys. The TOE allows a user to import a certificate (in PKCS#12 [PFX] format) and provides applications running on the TOE an API to import a certificate. In either case, the TOE will place the certificate into the user’s key store, where the TOE will remove the PKCS#12 password-based protection from the certificate and then encrypt the certificate with a DEK, which in turn is encrypted with the a KEK derived from the user's DKEK. All user and application items placed into the user’s keystore are secured in this fashion. Note that while operating in CC mode, the TOE also encrypts (using ODE) the Flash filesystem in which keystore items reside, providing a second layer of encryption protection to keystore objects.

The user of the TOE can elect to remove values from the keystore, as well as to securely wipe the entire device.

The TOE affords applications control (control over use and destruction) of keys that they create or import, and only the user or a common application developer can explicitly authorize access, use, or destruction of one application's key by any other application.

Entity	Can import?	Can destroy?	Allow other app to use?	Allow other app to destroy
User	Yes	Yes	Yes	
Administrator	Yes			
Mobile application	Yes			
Common application developer			Yes	Yes

FCS_STG_EXT.2: The TOE provides protection for all stored keys (i.e. those written to storage media for persistent storage) chained to both the user's password and the REK. FEKs (which are DEKs for individual encrypted files on removable media), which are 128-bit, and stored as part of the metadata for each encrypted file, are encrypted with AES-CBC. The FEK is protected by the SDCard MK, a 256-bit key. All other keys (including the SDCard MK) are encrypted with AES-GCM. All KEKs are 256-bit, ensuring that that the TOE encrypts every key with another key of equal or greater strength/size.

In the case of WiFi, the TOE utilizes the 802.11-2012 KCK and KEK keys to unwrap (decrypt) the WPA2 Group Temporal Key received from the access point. Additionally, the TOE protects persistent Wifi keys (user certificates and Pre-Shared Key values) in the same way as other DEKs, namely by encrypting them with a KEK chained to both the user's password and the REK.

FCS_STG_EXT.3: The key hierarchy shows AES-256 GCM is used to encrypt all KEKs and the GCM encryption mode itself ensures integrity as authenticated decryption operations fail if the encrypted KEK becomes corrupted.

FCS_TLSC_EXT.1: The TSF supports TLS version 1.0 with client (mutual) authentication and supports the eight selected ciphersuites utilizing SHA (see the selections in section 5.1.1.24) for use with EAP-TLS as part of WPA2. The TOE does not support any other ciphersuites with EAP-TLS, as the remainder of selectable cipher suites are based on SHA256, which requires TLS v1.2. The TOE, by design, supports only evaluated elliptic curves (P-256, P-384, & P-521 and no others) and requires/allows no configuration of the supported curves.

FCS_TLS_EXT.2: The TOE provides mobile applications (through its Android API) the use of TLS versions 1.0, 1.1, and 1.2 including support for all sixteen selectable ciphersuites (see the selections in section 5.1.1.25). The TOE supports Common Name (CN) and Subject Alternative Name (SAN) (DNS and IP address) as reference identifiers. The TOE supports client (mutual) authentication. The TOE, by design, supports only evaluated elliptic curves (P-256, P-384, & P-521 and no others) and requires/allows no configuration of the supported curves. While the TOE supports the use of wildcards in X.509 reference identifiers (CN and SAN), the TOE does not support certificate pinning.

6.2 User data protection

FDP_ACF_EXT.1: The TOE provides the following categories of system services to applications.

1. Normal – these are lower-risk services that the system automatically grants to a requesting application at installation, without asking for the user's explicit approval (though the user always has the option to review these permissions before installing)
2. Dangerous – these are higher-risk services that would give a requesting application access to private user data or control over the device that can negatively impact the user.
3. Signature – these are privileged services that the system grants only if the requesting application is signed with the same certificate as the application that declared the permission. If the certificates match, the system automatically grants the permission without notifying the user or asking for the user's explicit approval.
4. Signature or System – these are privileged services that the system grants only to applications that are in the Android system image or that are signed with the same certificate as the application that declared the permission. Like the Signature permission, the system automatically grants the permission without notifying the user.

An example of a Normal permission is the ability to vibrate the device: `android.permission.VIBRATE`. This permission allows an application to make the device vibrate, and an application that does not declare this permission would have its vibration requests ignored.

An example of a Dangerous privilege would be access to location services to determine the location of the mobile device: `android.permission.ACCESS_FINE_LOCATION`. The TOE controls access to Dangerous permissions during the installation of the application. The TOE prompts the user to review the application's requested permissions (by displaying a description of each permission group, into which individual permissions map, that an application requested access to). If the user approves, then the mobile device continues with the installation of the application. Thereafter, the mobile device grants that application during execution access to the set of permissions declared in its Manifest file.

An example of a Signature permission is the `android.permission.BIND_VPN_SERVICE`, that an application must declare in order to utilize the VpnService APIs of the device. Because the permission is a Signature permission, the mobile device only grants this permission to an application that requests this permission *and* that has been signed with the same developer key used to sign the application declaring the permission (in the case of the example, the Android Framework itself).

An example of a System or Signature permission is the `android.permission.LOCATION_HARDWARE`, which allows an application to use location features in hardware (such as the geofencing API). The device grants this permission to requesting applications that either have been signed with the same developer key used to sign the android application declaring the permissions or that reside in the "system" directory within Android, which for Android 4.4 and above, are applications residing in the /system/priv-app/ directory on the read-only system partition). Put another way, the device grants System or Signature permissions by Signature or by virtue of the requesting application being part of the "system image."

FDP_DAR_EXT.1: The TOE provides AES-256 CBC encryption of all data (which includes both user data and TSF data) stored on the TOE through On Device Encryption (ODE) of the data partition. The TOE also has TSF data for its ODE keystore, which the TOE stores outside the ODE encrypted data partition, and the TOE also includes a read-only filesystem in which the TOE's system executables, libraries, and their configuration data reside. For its ODE encryption of the data partition (where the TOE stores all user data and all application data), the TOE uses an AES-256 bit DEK with CBC feedback mode to encrypt the entire partition. The TOE also provides AES-128 CBC encryption of protected data stored on the external SDCard using FEKs. The TOE encrypts each individual file stored on the SDCard, generating a unique FEK for each file.

FDP_DAR_EXT.2: The TOE provides mobile applications a JAVA API allowing them mark data as sensitive data and have the TOE encrypt it accordingly. Any mobile application may use of the API to encrypt its data. The TOE stores the sensitive data in a segregated database not directly accessible by any mobile application, and furthermore, the TOE encrypts the data contents within the database.

FDP_IFC_EXT.1: The TOE includes an IPsec VPN Client that ensures that all traffic other than traffic necessary to establish the VPN connection (802.11-2012 traffic and IKEv2) flows through the VPN.

FDP_STG_EXT.1: The TOE's Trusted Anchor Database consists of the built-in certs (individually stored in /system/etc/security/cacerts) which the user can disable using the TOE's Android user interface [Settings->Security-> Trusted Credentials]) and consists of any additional user or admin/MDM loaded certificates. Disabled default certificates and user added certificates reside in the /data/misc/keychain/cacerts-removed and /data/misc/keychain/cacerts-added directories respectively. The built-in ones are protected as they are part of the TSF's read only system partition, while the TOE protects user-loaded certificates by storing them with appropriate permissions to prevent modification by mobile applications. The TOE also stores the user-loaded certificates in the user's keystore.

FDP_UPC_EXT.1: The TOE provides APIs allowing non-TSF applications (mobile applications) the ability to establish a secure channel using TLS, HTTPS, and Bluetooth DR/EDR. Mobile applications can use the following Android APIs for TLS, HTTPS, and Bluetooth respectively:

`javax.net.ssl.SSLContext:`

http://developer.android.com/reference/javax/net/ssl/SSLContext.html

`javax.net.ssl.HttpsURLConnection`:

http://developer.android.com/reference/javax/net/ssl/HttpsURLConnection.html

`android.bluetooth`:

http://developer.android.com/reference/android/bluetooth/package-summary.html

6.3 Identification and authentication

FIA_AFL_EXT.1(1): The TOE maintains, for each user, the number of failed logins since the last successful login, and upon reaching the maximum number of incorrect logins, the TOE performs a full wipe of all protected data (and in fact, wipes all users' data). An administrator can adjust the number of failed logins from the default of ten failed logins to a value between one and one hundred through an MDM. The TOE maintains the number of failed logins across power-cycles (so for example, assuming a configured maximum retry of ten incorrect attempts, if one were to enter five incorrect passwords and power cycle the phone, the phone would only all five more incorrect login attempts before wiping.

FIA_BLT_EXT.1: The TOE requires explicit user authorization before it will pair with a remote Bluetooth device. When pairing with another device, the TOE requires that the user either confirm that a displayed numeric passcode matches between the two devices or that the user enter (or choose) a numeric passcode that the peer device generates (or must enter).

FIA_PAE_EXT.1: The TOE can join WPA2-802.1X (802.11i) wireless networks requiring EAP-TLS authentication, acting as a client/supplicant (and in that role connect to the 802.11 access point and communicate with the 802.1X authentication server).

FIA_PMG_EXT.1: The TOE authenticates user through a password consisting of basic Latin characters (upper and lower case, numbers, and the special characters noted in the selection (see section 5.1.3.6). The TOE defaults to requiring passwords to have a minimum of six characters and can support no more than sixteen characters, contain at least one letter and one number. An MDM application can change these defaults and impose password restrictions (like quality, specify another minimum length, the minimum number of letters, numeric characters, lower case letters, upper case letters, symbols, a non-letters).

FIA_TRT_EXT.1: The TOE allows users to authenticate through external ports (either a USB keyboard or a Bluetooth keyboard paired in advance of the login attempt). If not using an external keyboard, a user must authenticate through the standard User Interface (using the TOE touchscreen). The TOE limits the number of authentication attempts through the UI to no more than five attempts within 30 seconds (irrespective of what keyboard the operator uses). Thus if the current [the n^{th}] and prior four authentication attempts have failed, and the $n\text{-}4^{th}$ attempt was less than 30 second ago, the TOE will prevent any further authentication attempts until 30 seconds has elapsed. Note as well that the TOE will wipe itself when it reaches the maximum number of unsuccessful authentication attempts (as described in FIA_AFL_EXT.1 above).

FIA_UAU.7: The TOE allows the user to enter the user's password from the lock screen. The TOE will, by default, display the most recently entered character of the password briefly or until the user enters the next character in the password, at which point the TOE obscures the character by replacing the character with a dot symbol. The user can configure the TOE's behavior for the normal lockscreen so that it does not briefly display the last typed character; however, the TOE always briefly displays the last entered character for the crypt-lock screen.

FIA_UAU_EXT.1: As described before, the TOE's Key Hierarchy requires the user's password in order to derive the PKEK1 & PKEK2 keys in order to decrypt other KEKs and DEKs. Thus, until it has the user's password, the TOE cannot decrypt the DEK utilized by ODE to decrypt protected data.

FIA_UAU_EXT.2: The TOE, when configured to require a user password (as is the case in CC mode), allows a user to make an emergency call, take screen shots (stored internally), turn the TOE off, restart the TOE, enable or disable airplane mode, configure sound/vibrate/mute, set volume for sound categories, or to receive an incoming phone call without first successfully authenticating. Beyond those actions, a user cannot perform any other actions other than observing notifications displayed on the lock screen until after successfully authenticating.

FIA_UAU_EXT.3: The TOE requires the user to enter their password in order to unlock the TOE. Additionally the TOE requires the user to confirm their current password when accessing the “Settings->Screen Security” menu in the TOE’s user interface. Only after entering their current user password can the user then elect to change their password.

FIA_X509_EXT.1: The TOE checks the validity of all imported CA certificates by checking for the presence of the basicConstraints extension and that the CA flag is set to TRUE as the TOE imports the certificate into the TOE’s Trust Anchor Database. If the TOE detects the absence of either the extension or flag, the TOE will import the certificate as a user public key and place add it to the keystore (not the Trust Anchor Database). The TOE also checks for the presence of the basicConstraints extension and CA flag in each CA certificate in a server’s certificate chain. Similarly, the TOE verifies the extendedKeyUsage Server Authentication purpose during certificate validation. The TOE’s certificate validation algorithm examines each certificate in the path (starting with the peer’s certificate) and first checks for validity of that certificate (e.g., has the certificate expired or it not yet valid, whether the certificate contains the appropriate X.509 extensions [e.g., the CA flag in the basic constraints extension for a CA certificate, or that a server certificate contains the Server Authentication purpose in the ExtendedKeyUsagefield], then verifies each certificate in the chain (applying the same rules as above, but also ensuring that the Issuer of each certificate matches the Subject in the next rung “up” in the chain and that the chain ends in a self-signed certificate present in either the TOE’s trusted anchor database or matches a specified Root CA), and finally the TOE performs revocation checking for all certificates in the chain..

FIA_X509_EXT.2: The TOE uses X.509v3 certificates as part of EAP-TLS, TLS and IPsec authentication. The TOE comes with a built-in set of default of Trusted Credentials (Android's set of trusted CA certificates). And while the user cannot remove any of the built-in default CA certificates, the user can disable any of those certificates through the user interface so that certificates issued by disabled CA’s cannot validate successfully. In addition, a user can import a new trusted CA certificate into the Key Store or an administrator can install a new certificate through an MDM.

The TOE does not establish TLS connections itself (beyond EAP-TLS used for WPA2 Wifi connections), but provides a series of APIs that mobile applications can use to check the validity of a peer certificate. The mobile application, after correctly using the specified APIs can be assured as to the validity of the peer certificate and will not establish the trusted connection if the peer certificate cannot be verified (including validity, certification path, and revocation [through CRL]). If, during the process of certificate verification, the TOE cannot establish a connection with the server acting as the CRL Distribution Point (CDP), the TOE will deem the server’s certificate as invalid and not establish a TLS connection with the server.

FIA_X509_EXT.3: The TOE’s Android operating system provides applications the `java.security.cert` Java API Class of methods validating certificates and certification paths (certificate chains establishing a trust chain from a certificate to a trust anchor). The available APIs may be found here:

http://developer.android.com/reference/java/security/cert/package-summary.html

6.4 Security management

FMT_MOF.1/FMT_SMF_EXT.1: The TOE provides the management functions described in the table in section 5.1.4.2. The table includes annotations describing the roles that have access to each service and how to access the service.

FMT_SMF_EXT.2(1): The TOE when unenrolled from an MDM removes all MDM policies and disables CC mode. The TOE also provides the capability to the MDM Agent to wipe all protected data and to remove any installed Enterprise applications.

6.5 Protection of the TSF

FPT_AEX_EXT.1: The Linux kernel of the TOE's Android operating system provides address space layout randomization utilizing a non-cryptographic kernel random function to provide 8 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_AEX_EXT.2: The TOE's Android 5.0.1 operating system utilizes a 3.10 Linux kernel, whose memory management unit (MMU) enforces read, write, and execute permissions on all pages of virtual memory and ensures that write and execute permissions are not simultaneously granted on all memory (exceptions are only made for Dalvik JIT compilation). The Android operating system (as of Android 2.3) sets the ARM No eXecute (XN) bit on memory pages and the TOE's ARMv7 Application Processor's MMU circuitry enforces the XN bits. From Android's documentation (https://source.android.com/devices/tech/security/index.html), Android 2.3 forward supports "Hardware-based No eXecute (NX) to prevent code execution on the stack and heap". Section B4.2 of the ARM v7 Architecture Reference Manual contains additional details about the MMU of ARM-based processors: http://www.club.cc.cmu.edu/~mjrosenb/ARM%20v7%20Architecture%20Reference%20Manual.pdf

FPT_AEX_EXT.3: The TOE's Android operating system provides explicit mechanisms to prevent stack buffer overruns (enabling -fstack-protector) in addition to taking advantage of hardware-based No eXecute to prevent code execution on the stack and heap. Samsung requires and applies these protections to all TSF executable binaries and libraries.

FPT_AEX_EXT.4: The TOE protects itself from modification by untrusted subjects using a variety of methods. The first protection employed by the TOE is a Secure Boot process that uses cryptographic signatures to ensure the authenticity and integrity of the bootloader and kernels using data fused into the device processor.

The TOE protects access to the REK and derived HEK to only trusted applications within the TEE (TrustZone). The TOE key manager includes a TEE module which utilizes the HEK to protect all other keys in the key hierarchy. All TEE applications are cryptographically signed, and when invoked at runtime (at the behest of an untrusted application), the TEE will only load the trusted application after successfully verifying its cryptographic signature. Furthermore, SKMM_TA checks the integrity of the system by checking the result from both Secure Boot/SecurityManager and from the Integrity Check Daemon before servicing any requests. Without this TEE application, no keys within the TOE (including keys for ScreenLock, the key store, and user data) can be successfully decrypted, and thus are useless.

The third protection is the TOE's internal SecurityManager watchdog service. The SecurityManager manages the CC mode of the TOE by looking for unsigned kernels or failures from other, non-cryptographic checks on system integrity, and upon detecting of a failure in either, disables the CC mode and notifies the TEE application. The TEE application then locks itself, again rendering all TOE keys useless.

Finally, the TOE's Android OS provides "sandboxing" that ensures that each non-system mobile application executes with the file permissions of a unique Linux user ID, in a different virtual memory space. This ensures that applications cannot access each other's memory space (it is possible for two processes to utilize shared memory, but not directly access the memory of another application) or files and cannot access the memory space or files of system-level applications.

FPT_BBD_EXT.1: The TOE's hardware and software architecture ensures separation the application processor (AP) from the baseband or communications processor (CP). From a software perspective, the AP and CP communicate logically through the Android Radio Interface Layer (RIL) daemon. This daemon, which executes on the AP, coordinates all communication between the AP and CP. It makes requests of the CP and accepts the response from the CP; however, the RIL daemon does not provide any reciprocal mechanism for the CP to make requests of the AP. Because the mobile architecture provides only the RIL daemon interface, the CP has no method to access the resources of the software executing on the AP.

FPT_KST_EXT.1: The TOE does not store any plaintext key material in its internal Flash or on external SDCards; instead, the TOE encrypts all keys before storing them. This ensures that irrespective of how the TOE powers down (e.g., a user commands the TOE to power down, the TOE reboots itself, or battery is removed), all keys in internal Flash or on an external SDCard are wrapped with a KEK. Please refer to section 6.1of the TSS for further information (including the KEK used) regarding the encryption of keys stored in the internal Flash and on external SDCards. As the TOE encrypts all keys stored in Flash, upon boot-up, the TOE must first decrypt and utilize keys.

FPT_KST_EXT.2: The TOE utilizes a cryptographic module consisting of an implementation of OpenSSL, the kernel crypto module, the key management module, and the following system-level executables that utilize KEKs: dm-crypt, eCryptfs, wpa_supplicant, and the keystore.

The TOE ensures that plaintext key material does not leave this cryptographic module first by not allowing the REK to leave and by ensuring that only authenticated entities can request utilization of the REK. Furthermore, the TOE only allows the system-level executables access to plaintext DEK values needed for their operation. The TSF software (the system-level executables) protects those plaintext DEK values in memory both by not providing any access to these values and by clearing them when no longer needed (in compliance with FCS_CKM.4).

FPT_KST_EXT.3: The TOE does not provide any way to export plaintext DEKs or KEKs (including all keys stored in the keystore) as the TOE chains all KEKs to the HEK/REK.

FPT_NOT_EXT.1: When the TOE encounters a self-test failure or when the TOE software integrity verification fails, the TOE transitions to a non-operational mode. The user may attempt to power-cycle the TOE to see if the failure condition persists, and if it does persist, the user may attempt to boot to the recovery mode/kernel to wipe data and perform a factory reset in order to recover the device.

FPT_STM.1: The TOE requires time for the Package Manager, FOTA image verifier, TLS certificate validation, wpa_supplicant, and key store applications. These TOE components obtain time from the TOE using system API calls [e.g., time() or gettimeofday()]. An application cannot modify the system time as mobile applications need the android "SET_TIME" permission to do so. Likewise, only a process with root privileges can directly modify the system time using system-level APIs. The TOE uses the Cellular Carrier time (obtained through the Carrier's network time server) as a trusted source; however, the user can also manually set the time through the TOE's user interface.

FPT_TST_EXT.1: The TOE performs known answer power on self-tests (POST) on its cryptographic algorithms to ensure that they are functioning correctly. The kernel itself performs known answer tests on its cryptographic algorithms to ensure they are working correctly and the SecurityManager service invokes the self-tests of OpenSSL at start-up to ensure that those cryptographic algorithms are working correctly. Should any of the tests fail, the TOE will reboot to see if that will clear the error.

Algorithm	Implemented in	Descripton
AES encryption/decryption	OpenSSL	Comparison of known answer to calculated valued
ECDH key agreement	OpenSSL	Comparison of known answer to calculated valued
DRBG random bit generation	OpenSSL	Comparison of known answer to calculated valued
ECDSA sign/verify	OpenSSL	Comparison of known answer to calculated valued
HMAC-SHA	OpenSSL	Comparison of known answer to calculated valued
RSA sign/verify	OpenSSL	Comparison of known answer to calculated valued
SHA hashing	OpenSSL	Comparison of known answer to calculated valued
AES encryption/decryption	Kernel Crypto	Comparison of known answer to calculated valued
HMAC-SHA	Kernel Crypto	Comparison of known answer to calculated valued
SHRNG random bit generation	Kernel Crypto	Comparison of known answer to calculated valued
SHA hashing	Kernel Crypto	Comparison of known answer to calculated valued

Table 6 Power-up Cryptographic Algorithm Known Answer Tests

FPT_TST_EXT.2: The TOE ensures a secure boot process in which the TOE verifies the digital signature of the bootloader software for the Application Processor (using a public key whose hash resides in the processor's internal fuses) before transferring control. The bootloader, in turn, verifies the signature of the Linux kernel (either the primary or the recovery kernel) it loads.

FPT_TUD_EXT.1: The TOE's user interface provides a method to query the current version of the TOE software/firmware (Android version, baseband version, kernel version, build number, and security software verison) and hardware (model and version). Additionally, the TOE provides users the ability to review the currently installed apps (including 3rd party "built-in" applications) and their version.

FPT_TUD_EXT.2: When in CC mode, the TOE verifies all updates to the TOE software using a public key (FOTA public key) chaining ultimately to the Secure Boot Public Key (SBPK), a hardware protected key whose SHA-256 hash resides inside the application processor (note that when not in CC mode, the TOE allows updates to the TOE software through ODIN mode of the bootloader). After verifying an update's FOTA signature, the TOE will then install those updates to the TOE.

The application processing verifies the bootloader's authenticity and integrity (thus tying the bootloader and subsequent stages to a hardware root of trust: the SHA-256 hash of the SBPK, which cannot be reprogrammed after the "write-enable" fuse has been blown).

The Android OS on the TOE requires that all applications bear a valid signature before Android will install the application.

ALC_TSU_EXT: Samsung utilizes industry best practices to ensure their devices are patched to mitigate security flaws. Samsung provides customers with a developer web portal (http://developer.samsung.com/notice/How-to-Use-the-Forum) in which one can ask questions as well as inform Samsung of potential issues with their software (as Samsung moderators monitor the forums), and as an Android OEM, also works with Google on reported Android issues (http://source.android.com/source/report-bugs.html) to ensure customer devices are secure.

Samsung will create updates and patches to resolve reported issues as quickly as possible, at which point the update is provided to the wireless carriers. The delivery time for resolving an issue depends on the severity, and can be as rapid as a few days before the carrier handoff for high priority cases. The wireless carriers perform additional tests to ensure the updates will not adversely impact their networks and then plan device rollouts once that testing is complete. Carrier updates usually take at least two weeks to as much as two months (depending on the type and severity of the update) to be rolled out to customers.

Samsung communicates with the reporting party to inform them of the status of the reported issue. Further information about updates is handled through the carrier release notes. Issues reported to Google directly are handled through Google's notification processes.

6.6 TOE access

FTA_SSL_EXT.1: The TOE transitions to its locked state either immediately after a User initiates a lock by pressing the power button or after a configurable period of inactivity, and as part of that transition, the TOE will display a lock screen to obscure the previous contents; however, the TOE's lock screen still displays email notifications, calendar appointments, text message notifications, the time, date, call notifications, battery life, signal strength, and carrier network. But without authenticating first, a user cannot perform any related actions based upon these notifications (they cannot respond to emails, calendar appointments, or text messages) other than responding to an incoming phone call.

Note that during power up, the TOE presents the user with an initial power-up ODE screen, where the user can only make an emergency call or enter the ODE password in order to allow the TOE to decrypt the OBE key so as to be able to access the data partition. After successfully authenticating at the power-up login screen, the TOE (when subsequently locked) presents the user with the lock screen.

FTA_TAB.1: The TOE can be configured to display a user-specified message (maximum of 23 characters) on the Lock screen, and additionally an administrator can also set a Lock screen message using an MDM.

FTA_WSE_EXT.1: The TOE allows an administrator to specify (through the use of an MDM) a list of wireless networks (SSIDs) to which the user may direct the TOE to connect to. When not enrolled with an MDM, the TOE allows the user to control to which wireless networks the TOE should connect, but does not provide an explicit list of such networks, rather the use may scan for available wireless network (or directly enter a specific wireless network), and then connect. Once a user has connected to a wireless network, the TOE will automatically reconnect to that network when in range and the user has enabled the TOE's WiFi radio.

6.7 Trusted path/channels

FTP_ITC_EXT.1: The TOE provides secured (encrypted and mutually authenticated) communication channels between itself and other trusted IT products through the use of 802.11-2012, 802.1X, and EAP-TLS, TLS and IPsec. The TOE permits itself and applications to initiate communicate via the trusted channel, and the TOE initiates communicate via the trusted channel for connection to a wireless access point. The TOE provides access to TLS via published APIs which are accessible to any application that needs an encrypted end-to-end trusted channel. The TOE has also been validated against the Protection Profile for IPsec Virtual Private Network (VPN) Clients.

6.8 KNOX Container Functionality

To differentiate the functionality provided by a KNOX container, this section enumerates the functionality provided by the container.

FDP_ACF_EXT.1.2: The TOE, through a combination of Android's multi-user capabilities and Security Enhancements (SE) for Android, provides the ability to create isolated containers within the device. Within a container a group of applications can be installed, and access to those applications is then restricted to usage solely within the container. The container boundary restricts the ability of sharing data such that applications outside the container cannot see, share or even copy data to those inside the container and vice versa. Exceptions to the boundary (such as allowing a copy operation) must be authorized by the administrator via policy. Furthermore, the container boundary policy can control access to hardware features, such as the camera or microphone, and restrict the ability of applications within the container to access those services.

FIA_AFL_EXT.1(2): The KNOX container maintains, for each user, the number of failed logins since the last successful login, and upon reaching the maximum number of incorrect logins, the KNOX container can either perform a full wipe of data protected by KNOX (i.e. data inside the container) or it can perform a remediation action set by the administrator (i.e. not a wipe of data, but lock the container until unlocked by the administrator). An administrator can adjust the number of failed logins from the default of ten failed logins to a value between one and one hundred through an MDM.

FIA_UAU.7: The KNOX container allows the user to enter the user's password from the KNOX lock screen. The KNOX container will, by default, display the most recently entered character of the password briefly or until the user enters the next character in the password, at which point the KNOX container obscures the character by replacing the character with a dot symbol.

FIA_UAU_EXT.3: The KNOX container requires the user to enter their password in order to unlock the KNOX container. Additionally the KNOX container requires the user to confirm their current password when accessing the "KNOX Settings -> KNOX unlock method" menu in the KNOX container's user interface. Only after entering their current user password can the user then elect to change their password.

FMT_MOF.1/FMT_SMF_EXT.1: The KNOX container grants the user the exclusive functions specified in the SFR (see section 5.1.4).

The control over the camera and microphone within the KNOX container only affects access to those resources inside the container, not outside the container. If either of these is disabled outside the container then they will not be available within the container, even if they are enabled.

FMT_SMF_EXT.2(2): The KNOX container, when a container is unenrolled (or removed), wipes all data inside the KNOX container.

FTA_SSL_EXT.1: The KNOX container transitions to its locked state either immediately after a User initiates a lock by pressing the container lock button from the notification bar or after a configurable period of inactivity, and as part of that transition, the KNOX container will display a lock screen to obscure the previous contents. When the KNOX container is locked, it can still display calendar appointments and other notifications allowed by the administrator to be shown on outside the container (in the notification area). But without authenticating first to the KNOX container, a user cannot perform any related actions based upon these container notifications (they cannot respond to emails, calendar appointments, or text messages).

The KNOX container timeout is independent from the TOE timeout and as such can be set to different values.

7. TSF Inventory

Below is a list of user-mode TSF binaries and libraries. All are built with the -fstack-protector option set. For each binary/library, the name, path and security function is provided.

Name	Path	Securuty Function
charon	/system/bin	VPN
dalvikvm	/system/bin	VM
icd	/system/bin	Integrity Check
keystore	/system/bin	Key Store
odekeymgr	/system/bin	DAR
qrngd	/system/bin	RNG
qseecomd	/system/bin	Trustzone Daemon
sfotahelper	/system/bin	FOTA
time_daemon	/system/bin	Time
vold	/system/bin	DAR
wpa_supplicant	/system/bin	DAR
libSecurityManagerNative.so	/system/lib	Self-test
libcharon.so	/system/lib	VPN
libcrypto.so	/system/lib	Crypto
libdirencryption.so	/system/lib	DAR
libhydra.so	/system/lib	VPN
libjavacrypto.so	/system/lib	Crypto JNI
libkeystore_binder.so	/system/lib	KeyStore
libkeyutils.so	/system/lib	DAR
libsec_devenc.so	/system/lib	DAR
libsec_ecryptfs.so	/system/lib	DAR
libsec_ode_km.so	/system/lib	DAR
libskmm.so	/system/lib	Key Mgmt
libskmm_helper.so	/system/lib	Key Mgmt
libsoftkeymaster.so	/system/lib	Key Store
libssl.so	/system/lib	SSL/TLS
libstrongswan.so	/system/lib	VPN
libvstr.so	/system/lib	VPN