# Zebra SM6375 Devices on Android 11 Security Target

Version 1.4
04/26/23

***Prepared for:***

**Zebra Technologies Corporation**

3 Overlook Point
Lincolnshire, IL 60069-4302

***Prepared By:***


www.gossamersec.com

**LIST OF TABLES**

# 1. Security Target Introduction

This section identifies the Security Target (ST) and Target of Evaluation (TOE) identification, ST conventions, ST conformance claims, and the ST organization. The TOE is Zebra Technologies Corporation Zebra SM6375 Devices on Android 11. 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 parenthetical number placed at the end of the component. For example FDP_ACC.1(1) and FDP_ACC.1(2) indicate that the ST includes two iterations of the FDP_ACC.1 requirement.
    - 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 …").
- 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 –** Zebra SM6375 Devices on Android 11 Security Target

**ST Version** – Version 1.4

**ST Date** – 04/26/23

## 1.2 TOE Reference

**TOE Identification** – Zebra Technologies Corporation Zebra SM6375 Devices on Android 11

**TOE Developer** – Zebra Technologies Corporation

**Evaluation Sponsor** – Zebra Technologies Corporation

## 1.3 TOE Overview

The Target of Evaluation (TOE) is Zebra Technologies Corporation Zebra SM6375 Devices on Android 11.

The Zebra Devices are rugged handheld utilizing the SM6375 chipset, angled rear-facing barcode reader, optional stylus pen, and battery that is warm-swappable. The Devices use the Android operating system, providing access to applications from the Google Play store or Zebra's partners. The Devices feature built-in multi-carrier 4G LTE and FirstNet Ready with Band 14, voice capabilities, and dual SIM cards. The TOE supports using client certificates to connect to access points offering WPA2/WPA3 networks with 802.1x/EAP-TLS, or alternatively connecting to cellular base stations when utilizing mobile data.

The TOE offers mobile applications an Application Programming Interface (API) including that provided by the Android framework and supports API calls to the Android Management APIs.

## 1.4 TOE Description

The TOE encompasses mobile devices that support enterprises and individual users alike and this evaluation includes the following models and versions.

| Product | Model # | CPU | Kernel | Android OS version | Security Patch Level |
|---|---|---|---|---|---|
| 6375 Mobile Handhelds | ET40 | Qualcomm SM6375 | 5.4.147 | Android 11.0 | February 2023 |
| | ET45 | Qualcomm SM6375 | 5.4.147 | Android 11.0 | February 2023 |
| | ET40HC | Qualcomm SM6375 | 5.4.147 | Android 11.0 | February 2023 |
| | ET45HC | Qualcomm SM6375 | 5.4.147 | Android 11.0 | February 2023 |

Some features and settings must be enabled for the TOE to operate in its evaluated configuration. The following features and settings must be enabled:

1. Convert to File-Based Encryption (FBE) TOE using the FDE to FBE package and the StageNow, fastboot, or recovery mode method to apply the update
2. Require a lockscreen password
3. Disable Smart Lock
4. Enable Encryption of Wi-Fi and Bluetooth secrets by enabling ‘niap_mode’
5. Disable Debugging Features (Developer options)
6. Disable installation of applications from unknown sources
7. Enable Audit Logging

Doing this ensures that the device complies with the MDFPP requirements. Please refer to the Admin Guide on how to configure these settings and features.

### 1.4.1 TOE Architecture

The TOE provides a rich API to mobile applications and provides users installing an application the option to either approve or reject an application based upon the API access that the application requires (or to grant applications access at runtime).

The TOE also provides users with the ability to protect Data-At-Rest with AES encryption, including all user and mobile application data stored in the user’s data partition. The TOE uses a key hierarchy that combines a REK with the user’s password to provide protection to all user and application cryptographic keys stored in the TOE.

Finally, the TOE can interact with a Mobile Device Management (MDM) system (not part of this evaluation) to allow enterprise control of the configuration and operation of the device so as to ensure adherence to enterprise-wide policies (for example, restricting use of a corporate provided device’s camera, forced configuration of maximum login attempts, pulling of audit logs off the TOE, etc.) as well as policies governing enterprise applications and data. An MDM is made up of two parts: the MDM agent and MDM server. The MDM Agent is installed on the phone/mobile

computer as an administrator with elevated permissions (allowing it to change the relevant settings on the phone/device) while the MDM Server is used to issue the commands to the MDM Agent. Neither portion of the MDM process is considered part of the TOE, and therefore not being directly evaluated.

The TOE includes several different levels of execution including (from lowest to highest): hardware, a Trusted Execution Environment, Android's Linux kernel, and Android's user space, which provides APIs allowing applications to leverage the cryptographic functionality of the device.

#### 1.4.1.1 Physical Boundaries

The TOE's physical boundary is the physical perimeter of its enclosure. The TOE runs Android as its software/OS, executing on a Qualcomm Snapdragon processor. The TOE does not include the user applications that run on top of the operating system but does include controls that limit application behavior. Further, the device provides support for downloadable MDM agents to be installed to limit or permit different functionality of the device. There is no built-in MDM agent pre-installed on the device.

The TOE communicates and interacts with 802.11-2012 Access Points and mobile data networks to establish network connectivity, and 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 TOE:

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

##### 1.4.1.2.1 Security audit

The TOE implements a security log and logcat that are each stored in a circular memory buffer. An MDM agent can read/fetch the security logs, can retrieve logcat logs, and then handle appropriately (potentially storing the log to Flash or transmitting its contents to the MDM server). These log methods meet the logging requirements outlined by FAU_GEN.1 in MDFPPv3.2. Please see the Security audit section for further information and specifics.

##### 1.4.1.2.2 Cryptographic support

The TOE includes multiple cryptographic libraries with CAVP certified algorithms for a wide range of cryptographic functions including the following: 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 may be used to implement security protocols such as TLS, EAP-TLS, IPsec, and HTTPS and to encrypt the media (including the generation and protection of data and key encryption keys) used by the TOE. Many of these cryptographic functions are also accessible as services to applications running on the TOE allowing application developers to ensure their application meets the required criteria to remain compliant to MDFPP standards.

##### 1.4.1.2.3 User data protection

The TOE controls access to system services by hosted applications, including protection of the Trust Anchor Database. Additionally, the TOE protects user and other sensitive data using File-Based Encryption (FBE) so that even if a device is physically lost, the data remains protected. The TOE's evaluated configuration supports Android Enterprise

profiles to provide additional separation between application and application data belonging to the Enterprise profile. Please see the Admin Guide for additional details regarding how to set up and use Enterprise profiles.

#### 1.4.1.2.4 Identification and authentication

The TOE supports a number of features related to identification and authentication. From a user perspective, except for FCC mandated (making phone calls to an emergency number) or non-sensitive functions (e.g., choosing the keyboard input method or taking screen shots), a password (i.e., Password Authentication Factor) must be correctly entered to unlock the TOE. Also, even when unlocked, the TOE requires the user re-enter the password 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 up to 16 characters are supported.

The TOE can also serve as an 802.1X supplicant and can both use X.509v3 and validate certificates for EAP-TLS, TLS, and HTTPS exchanges.

#### 1.4.1.2.5 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.

#### 1.4.1.2.6 Protection of the TSF

The TOE implements a number of features 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 through the use of the application processor's hardware. The TOE disallows all read access to the Root Encryption Key and retains all keys derived from the REK within its Trusted Execution Environment (TEE). Application software can only use keys derived from the REK by reference and receive the result.

The TOE 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 also protects 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 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 as all applications must have signatures (even if self-signed).

#### 1.4.1.2.7 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 administrator specified (using the TOE's MDM API) advisory message (banner) when the user unlocks the TOE for the first use after reboot.

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

#### 1.4.1.2.8 Trusted path/channels

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

### 1.4.2 TOE Documentation

Administrator Guidance for Zebra Devices (ET4X), Version 0.4, 2023/04/04 **[Admin 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 5, April 2017.
    - Part 2 Extended
- Common Criteria for Information Technology Security Evaluation Part 3: Security assurance components, Version 3.1, Revision 5, April 2017.
    - Part 3 Extended
- Package Claims:
    - PP-Configuration for Mobile Device Fundamentals and Bluetooth, 2021-04-15 (CFG_MDF-BT_V1.0)
        - The PP-Configuration includes the following components:
            - Base-PP: Protection Profile for Mobile Device Fundamentals, Version 3.2, (PP_MD_V3.2)
            - PP-Module: PP-Module for Bluetooth, Version 1.0, (MOD_BT_V1.0)
    - Package Claims:
        - General Purpose Operating Systems Protection Profile/Mobile Device Fundamentals Protection Profile Extended Package (EP) Wireless Local Area Network (WLAN) Clients, Version 1.0, 08 February 2016 (PP_WLAN_CLI_EP_V1.0)
        - Functional Package: Functional Package for Transport Layer Security (TLS), Version 1.1, (PKG_TLS_V1.1)

| Package | Technical Decision | Applied | Notes |
|---|---|---|---|
| MOD_BT_V1.0 | TD0671 – Bluetooth PP-Module updated to allow for new PP and PP-Module Versions | Yes | |
| MOD_BT_V1.0 | TD0640 – Handling BT devices that do not support encryption | Yes | |
| MOD_BT_V1.0 | TD0685 - BT missing multiple SFR-to-Obj mappings | Yes | |
| MOD_BT_V1.0 | TD0707 – Formatting corrections for MOD_BT_V1.0 | Yes | |
| MOD_BT_V1.0 | TD0600 - Conformance claim sections updated to allow for MOD_VPNC_V2.3 | No | VPNC not claimed |
| MOD_BT_V1.0 | TD0650 - Conformance claim sections updated to allow for MOD_VPNC_V2.3 and 2.4 | No | VPNC not claimed |
| MOD_BT_V1.0 | TD0645 - Bluetooth audit details | Yes | |
| PKG_TLS_V1.1 | TD0588 - Session Resumption Support in TLS package | No | (D)TLSS not claimed |
| PKG_TLS_V1.1 | TD0513 - CA Certificate loading | Yes | Manageable Trust store |
| PKG_TLS_V1.1 | TD0499 - Testing with pinned certificates | Yes | Pinned certs not supported |
| PKG_TLS_V1.1 | TD0469 - Modification of test activity for FCS_TLSS_EXT.1.1 test 4.1 | No~~Yes~~ | |
| PKG_TLS_V1.1 | TD0442 - Updated TLS Ciphersuites for TLS Package | Yes | |
| PKG_TLS_V1.1 | TD0726 – Corrections to (D) TLSS SFRs in TLS 1.1 FP | No | (D)TLSS not claimed |
| PP_MDF_V3.2 | TD0677 - Correction to Symbol in FCS_RBG_EXT.1 Test EA for MDF 3.3 | No~~Yes~~ | |

| | | | |
|---|---|---|---|
| PP_MDF_V3.2 | TD0676 - Correction to Symbol in FCS_RBG_EXT.1 Test EA for MDF 3.2 | Yes | |
| PP_MDF_V3.2 | TD0663 - Audit Listing for MDF Moved to Guidance | Yes | |
| PP_MDF_V3.2 | TD0658 - Updates to Table 7 Column References in MDF v3.2 | Yes | Applied |
| PP_MDF_V3.2 | TD0653 - MDF v3.2 ASE References | Yes | Applied |
| PP_MDF_V3.2 | TD0650 - Conformance claim sections updated to allow for MOD_VPNC_V2.3 and 2.4 | No | VPNC module not claimed |
| PP_MDF_V3.2 | TD0646 - Function 23 Allow Invalid Certs | Yes | |
| PP_MDF_V3.2 | TD0643 – Data Signaling, Mgmt Function #24 | No | Not claimed |
| PP_MDF_V3.2 | TD0623 - FIA_X509_EXT.2.1 Protocol Selection | Yes | |
| PP_MDF_V3.2 | TD0600 - Conformance claim sections updated to allow for MOD_VPNC_V2.3 | No | VPNC not claimed |
| PP_MDF_V3.2 | TD0596 - VPN Traffic Permitted in FDP_IFC_EXT.1 | Yes | |
| PP_MDF_V3.2 | TD0603 - RFC Update in FIA_X509_EXT.1 for MDF PP v3.2 | YesNo | |
| PP_MDF_V3.2 | TD0705 – ECD for MDF 3.2 | Yes | |
| PP_MDF_V3.2 | TD0706 – FIA_UAU.6 Iterations | Yes | |
| PP_WLAN_CLI_EP_V1.0 | TD0517 - WLAN Client Corrections for X509 and TLSC | Yes | |
| PP_WLAN_CLI_EP_V1.0 | TD0492 - TLS-EAP Ciphers and TLS versions for WLAN Client | Yes | |
| PP_WLAN_CLI_EP_V1.0 | TD0470 - Wireless Network Restrictions | Yes | |
| PP_WLAN_CLI_EP_V1.0 | TD0439 - EAP-TLS Revocation Checking | Yes | |
| PP_WLAN_CLI_EP_V1.0 | TD0244 - FCS_TLSC_EXT - TLS Client Curves Allowed | Yes | |
| PP_WLAN_CLI_EP_V1.0 | TD0194 - Update to Audit of FTP_ITC_EXT.1/WLAN | Yes | |

***Acronyms and Terminology***

| | |
|---|---|
| MDFPP32 | PP_MDF_V3.2 |
| PKGTLS11 | PKG_TLS_V1.1 |
| WLANCEP10 | PP_WLAN_CLI_EP_V1.0 |
| BT10 | MOD_BT_V1.0 |

## 2.1 Conformance Rationale

The ST conforms to the MDFPP32/PKGTLS11/WLANCEP10/BT10. As explained previously, the security problem definition, security objectives, and security requirements have been drawn from the PP.

# 3. Security Objectives

The Security Problem Definition may be found in the MDFPP32/PKGTLS11/WLANCEP10/BT10 and this section reproduces only the corresponding Security Objectives for operational environment for reader convenience. The MDFPP32/PKGTLS11/WLANCEP10/BT10 offers additional information about the identified security objectives, but that has not been reproduced here and the MDFPP32/PKGTLS11/WLANCEP10/BT10 should be consulted if there is interest in that material.

In general, the MDFPP32/PKGTLS11/WLANCEP10/BT10 has defined Security Objectives appropriate for mobile device and as such are applicable to the Zebra Technologies Corporation Zebra SM6375 Devices on Android 11 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.DATA_PROPER_USER** Administrators take measures to ensure that mobile device users are adequately vetted against malicious intent and are made aware of the expectations for appropriate use of the device.

**OE.NO_TOE_BYPASS** Information cannot flow between external and internal networks located in different enclaves without passing through the TOE.

**OE.NOTIFY** The Mobile User will immediately notify the administrator if the Mobile Device is lost or stolen.

**OE.PRECAUTION** The mobile device user exercises precautions to reduce the risk of loss or theft of the Mobile Device.

**OE.TRUSTED_ADMIN** TOE Administrators are trusted to follow and apply all administrator guidance in a trusted manner.

# 4. Extended Components Definition

All of the extended requirements in this ST have been drawn from the MDFPP32/PKGTLS11/WLANCEP10/BT10. The MDFPP32/PKGTLS11/WLANCEP10/BT10 defines the following extended requirements and since they are not redefined in this ST the MDFPP32/PKGTLS11/WLANCEP10/BT10 should be consulted for more information in regard to those CC extensions.

**Extended SFRs:**

- MDFPP32:FCS_CKM_EXT.1: Cryptographic Key Support
- MDFPP32:FCS_CKM_EXT.2: Cryptographic Key Random Generation
- MDFPP32:FCS_CKM_EXT.3: Cryptographic Key Generation
- MDFPP32:FCS_CKM_EXT.4: Key Destruction
- MDFPP32:FCS_CKM_EXT.5: TSF Wipe
- MDFPP32:FCS_CKM_EXT.6: Salt Generation
- BT10:FCS_CKM_EXT.8: Bluetooth Key Generation
- MDFPP32:FCS_HTTPS_EXT.1: HTTPS Protocol
- MDFPP32:FCS_IV_EXT.1: Initialization Vector Generation
- MDFPP32:FCS_RBG_EXT.1: Random Bit Generation
- MDFPP32:FCS_SRV_EXT.1: Cryptographic Algorithm Services
- MDFPP32:FCS_SRV_EXT.2: Cryptographic Algorithm Services
- MDFPP32:FCS_STG_EXT.1: Cryptographic Key Storage
- MDFPP32:FCS_STG_EXT.2: Encrypted Cryptographic Key Storage
- MDFPP32:FCS_STG_EXT.3: Integrity of Encrypted Key Storage
- PKGTLS11:FCS_TLS_EXT.1: TLS Protocol
- PKGTLS11:FCS_TLSC_EXT.1: TLS Client Protocol
- WLANCEP10:FCS_TLSC_EXT.1/WLAN: Extensible Authentication Protocol-Transport Layer Security
- PKGTLS11:FCS_TLSC_EXT.2: TLS Client Support for Mutual Authentication
- WLANCEP10:FCS_TLSC_EXT.2/WLAN: TLS Client Protocol
- PKGTLS11:FCS_TLSC_EXT.3: TLS Client Support for Signature Algorithms Extension
- PKGTLS11:FCS_TLSC_EXT.4: TLS Client Support for Renegotiation
- PKGTLS11:FCS_TLSC_EXT.5: TLS Client Support for Supported Groups Extension
- MDFPP32:FDP_ACF_EXT.1: Access Control for System Services
- MDFPP32:FDP_ACF_EXT.2: Access Control for System Resources
- MDFPP32:FDP_DAR_EXT.1: Protected Data Encryption
- MDFPP32:FDP_DAR_EXT.2: Sensitive Data Encryption
- MDFPP32:FDP_IFC_EXT.1: Subset Information Flow Control - per TD0596
- MDFPP32:FDP_STG_EXT.1: User Data Storage
- MDFPP32:FDP_UPC_EXT.1/APPS: Inter-TSF User Data Transfer Protection (Applications)
- MDFPP32:FDP_UPC_EXT.1/BLUETOOTH: Inter-TSF User Data Transfer Protection (Bluetooth)

- MDFPP32:FIA_AFL_EXT.1: Authentication Failure Handling
- BT10:FIA_BLT_EXT.1: Bluetooth User Authorization
- BT10:FIA_BLT_EXT.2: Bluetooth Mutual Authentication
- BT10:FIA_BLT_EXT.3: Rejection of Duplicate Bluetooth Connections
- BT10:FIA_BLT_EXT.4: Secure Simple Pairing
- BT10:FIA_BLT_EXT.6: Trusted Bluetooth Device User Authorization
- BT10:FIA_BLT_EXT.7: Untrusted Bluetooth Device User Authorization
- WLANCEP10:FIA_PAE_EXT.1: Port Access Entity Authentication
- MDFPP32:FIA_PMG_EXT.1: Password Management
- MDFPP32:FIA_TRT_EXT.1: Authentication Throttling
- MDFPP32:FIA_UAU.6/CREDENTIAL: Re-authenticating (Credential Change) - per TD0706
- MDFPP32:FIA_UAU.6/LOCKED: Re-authenticating (TSF Lock) - per TD0706
- MDFPP32:FIA_UAU_EXT.1: Authentication for Cryptographic Operation
- MDFPP32:FIA_UAU_EXT.2: Timing of Authentication
- MDFPP32:FIA_X509_EXT.1: X.509 Validation of Certificates
- WLANCEP10:FIA_X509_EXT.1/WLAN: X.509 Certificate Validation
- MDFPP32:FIA_X509_EXT.2: X.509 Certificate Authentication - per TD0623
- WLANCEP10:FIA_X509_EXT.2/WLAN: X.509 Certificate Authentication (EAP-TLS)
- MDFPP32:FIA_X509_EXT.3: Request Validation of Certificates
- MDFPP32:FMT_MOF_EXT.1: Management of Security Functions Behavior
- MDFPP32:FMT_SMF_EXT.1: Specification of Management Functions
- BT10:FMT_SMF_EXT.1/BT: Specification of Management Functions
- WLANCEP10:FMT_SMF_EXT.1/WLAN: Specification of Management Functions (Wireless LAN)
- MDFPP32:FMT_SMF_EXT.2: Specification of Remediation Actions
- MDFPP32:FMT_SMF_EXT.3: Current Administrator
- MDFPP32:FPT_AEX_EXT.1: Application Address Space Layout Randomization
- MDFPP32:FPT_AEX_EXT.2: Memory Page Permissions
- MDFPP32:FPT_AEX_EXT.3: Stack Overflow Protection
- MDFPP32:FPT_AEX_EXT.4: Domain Isolation
- MDFPP32:FPT_AEX_EXT.5: Kernel Address Space Layout Randomization
- MDFPP32:FPT_BBD_EXT.1: Application Processor Mediation
- MDFPP32:FPT_JTA_EXT.1: JTAG Disablement
- MDFPP32:FPT_KST_EXT.1: Key Storage
- MDFPP32:FPT_KST_EXT.2: No Key Transmission
- MDFPP32:FPT_KST_EXT.3: No Plaintext Key Export
- MDFPP32:FPT_NOT_EXT.1: Self-Test Notification

- MDFPP32:FPT_TST_EXT.1: TSF Cryptographic Functionality Testing
- WLANCEP10:FPT_TST_EXT.1/WLAN: TSF Cryptographic Functionality Testing (Wireless LAN)
- MDFPP32:FPT_TST_EXT.2/POSTKERNEL: TSF Integrity Checking (Post-Kernel)
- MDFPP32:FPT_TST_EXT.2/PREKERNEL: TSF Integrity Checking (Pre-Kernel)
- MDFPP32:FPT_TUD_EXT.1: Trusted Update: TSF Version Query
- MDFPP32:FPT_TUD_EXT.2: TSF Update Verification
- MDFPP32:FPT_TUD_EXT.3: Application Signing
- MDFPP32:FTA_SSL_EXT.1: TSF- and User-initiated Locked State
- WLANCEP10:FTA_WSE_EXT.1: Wireless Network Access
- BT10:FTP_BLT_EXT.1: Bluetooth Encryption
- BT10:FTP_BLT_EXT.2: Persistence of Bluetooth Encryption
- BT10:FTP_BLT_EXT.3/BR: Bluetooth Encryption Parameters (BR/EDR)
- BT10:FTP_BLT_EXT.3/LE: Bluetooth Encryption Parameters (LE)
- MDFPP32:FTP_ITC_EXT.1: Trusted Channel Communication
- WLANCEP10:FTP_ITC_EXT.1/WLAN: Trusted Channel Communication (Wireless LAN)

**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 MDFPP32/PKGTLS11/WLANCEP10/BT10. The refinements and operations already performed in the MDFPP32/PKGTLS11/WLANCEP10/BT10 are not identified (e.g., highlighted) here, rather the requirements have been copied from the MDFPP32/PKGTLS11/WLANCEP10/BT10 and any residual operations have been completed herein. Of particular note, the MDFPP32/PKGTLS11/WLANCEP10/BT10 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 MDFPP32/PKGTLS11/WLANCEP10/BT10. However, the SARs are effectively refined since requirement-specific 'Assurance Activities' are defined in the MDFPP32/PKGTLS11/WLANCEP10/BT10 that serve to ensure corresponding evaluations. The MDFPP32/PKGTLS11/WLANCEP10/BT10 should be consulted for the assurance activity definitions.

## 5.1 TOE Security Functional Requirements

The following table identifies the SFRs that are satisfied by Zebra Technologies Corporation Zebra SM6375 Devices on Android 11 TOE.

| Requirement Class | Requirement Component |
|---|---|
| **FAU: Security audit** | MDFPP32/BT10/WLANEP10:FAU_GEN.1: Audit Data Generation |
| | MDFPP32:FAU_SAR.1: Audit Review |
| | MDFPP32:FAU_STG.1: Audit Storage Protection |
| | MDFPP32:FAU_STG.4: Prevention of Audit Data Loss |
| **FCS: Cryptographic support** | MDFPP32:FCS_CKM.1: Cryptographic Key Generation |
| | WLANCEP10:FCS_CKM.1/WLAN: Cryptographic Key Generation (Symmetric Keys for WPA2 Connections) |
| | MDFPP32:FCS_CKM.2/LOCKED: Cryptographic Key Establishment |
| | MDFPP32:FCS_CKM.2/UNLOCKED: Cryptographic Key Establishment |
| | WLANCEP10:FCS_CKM.2/WLAN: Cryptographic Key Distribution (GTK) |
| | MDFPP32:FCS_CKM_EXT.1: Cryptographic Key Support |
| | MDFPP32:FCS_CKM_EXT.2: Cryptographic Key Random Generation |
| | MDFPP32:FCS_CKM_EXT.3: Cryptographic Key Generation |
| | MDFPP32:FCS_CKM_EXT.4: Key Destruction |
| | MDFPP32:FCS_CKM_EXT.5: TSF Wipe |
| | MDFPP32:FCS_CKM_EXT.6: Salt Generation |
| | BT10:FCS_CKM_EXT.8: Bluetooth Key Generation |
| | MDFPP32:FCS_COP.1/CONDITION: Cryptographic Operation |
| | MDFPP32:FCS_COP.1/ENCRYPT: Cryptographic Operation |
| | MDFPP32:FCS_COP.1/HASH: Cryptographic Operation |
| | MDFPP32:FCS_COP.1/KEYHMAC: Cryptographic Operation |
| | MDFPP32:FCS_COP.1/SIGN: Cryptographic Operation |
| | MDFPP32:FCS_HTTPS_EXT.1: HTTPS Protocol |
| | MDFPP32:FCS_IV_EXT.1: Initialization Vector Generation |
| | MDFPP32:FCS_RBG_EXT.1: Random Bit Generation |
| | MDFPP32:FCS_SRV_EXT.1: Cryptographic Algorithm Services |
| | MDFPP32:FCS_SRV_EXT.2: Cryptographic Algorithm Services |

| | |
|---|---|
| | MDFPP32:FCS_STG_EXT.1: Cryptographic Key Storage |
| | MDFPP32:FCS_STG_EXT.2: Encrypted Cryptographic Key Storage |
| | MDFPP32:FCS_STG_EXT.3: Integrity of Encrypted Key Storage |
| | PKGTLS11:FCS_TLS_EXT.1: TLS Protocol |
| | PKGTLS11:FCS_TLSC_EXT.1: TLS Client Protocol |
| | WLANCEP10:FCS_TLSC_EXT.1/WLAN: Extensible Authentication Protocol-Transport Layer Security |
| | PKGTLS11:FCS_TLSC_EXT.2: TLS Client Support for Mutual Authentication |
| | WLANCEP10:FCS_TLSC_EXT.2/WLAN: TLS Client Protocol |
| | PKGTLS11:FCS_TLSC_EXT.3: TLS Client Support for Signature Algorithms Extension |
| | PKGTLS11:FCS_TLSC_EXT.4: TLS Client Support for Renegotiation |
| | PKGTLS11:FCS_TLSC_EXT.5: TLS Client Support for Supported Groups Extension |
| **FDP: User data protection** | MDFPP32:FDP_ACF_EXT.1: Access Control for System Services |
| | MDFPP32:FDP_ACF_EXT.2: Access Control for System Resources |
| | MDFPP32:FDP_DAR_EXT.1: Protected Data Encryption |
| | MDFPP32:FDP_DAR_EXT.2: Sensitive Data Encryption |
| | MDFPP32:FDP_IFC_EXT.1: Subset Information Flow Control - per TD0596 |
| | MDFPP32:FDP_STG_EXT.1: User Data Storage |
| | MDFPP32:FDP_UPC_EXT.1/APPS: Inter-TSF User Data Transfer Protection (Applications) |
| | MDFPP32:FDP_UPC_EXT.1/BLUETOOTH: Inter-TSF User Data Transfer Protection (Bluetooth) |
| **FIA: Identification and authentication** | MDFPP32:FIA_AFL_EXT.1: Authentication Failure Handling |
| | BT10:FIA_BLT_EXT.1: Bluetooth User Authorization |
| | BT10:FIA_BLT_EXT.2: Bluetooth Mutual Authentication |
| | BT10:FIA_BLT_EXT.3: Rejection of Duplicate Bluetooth Connections |
| | BT10:FIA_BLT_EXT.4: Secure Simple Pairing |
| | BT10:FIA_BLT_EXT.6: Trusted Bluetooth Device User Authorization |
| | BT10:FIA_BLT_EXT.7: Untrusted Bluetooth Device User Authorization |
| | WLANCEP10:FIA_PAE_EXT.1: Port Access Entity Authentication |
| | MDFPP32:FIA_PMG_EXT.1: Password Management |
| | MDFPP32:FIA_TRT_EXT.1: Authentication Throttling |
| | MDFPP32:FIA_UAU.5: Multiple Authentication Mechanisms |
| | MDFPP32:FIA_UAU.6/CREDENTIAL: Re-authenticating (Credential Change) - per TD0706 |
| | MDFPP32:FIA_UAU.6/LOCKED: Re-authenticating (TSF Lock) - per TD0706 |
| | MDFPP32:FIA_UAU.7: Protected Authentication Feedback |
| | MDFPP32:FIA_UAU_EXT.1: Authentication for Cryptographic Operation |
| | MDFPP32:FIA_UAU_EXT.2: Timing of Authentication |
| | MDFPP32:FIA_X509_EXT.1: X.509 Validation of Certificates |
| | WLANCEP10:FIA_X509_EXT.1/WLAN: X.509 Certificate Validation |

| | |
|---|---|
| | MDFPP32:FIA_X509_EXT.2: X.509 Certificate Authentication - per TD0623 |
| | WLANCEP10:FIA_X509_EXT.2/WLAN: X.509 Certificate Authentication (EAP-TLS) |
| | MDFPP32:FIA_X509_EXT.3: Request Validation of Certificates |
| **FMT: Security management** | MDFPP32:FMT_MOF_EXT.1: Management of Security Functions Behavior |
| | MDFPP32:FMT_SMF_EXT.1: Specification of Management Functions |
| | BT10:FMT_SMF_EXT.1/BT: Specification of Management Functions |
| | WLANCEP10:FMT_SMF_EXT.1/WLAN: Specification of Management Functions (Wireless LAN) |
| | MDFPP32:FMT_SMF_EXT.2: Specification of Remediation Actions |
| | MDFPP32:FMT_SMF_EXT.3: Current Administrator |
| **FPT: Protection of the TSF** | MDFPP32:FPT_AEX_EXT.1: Application Address Space Layout Randomization |
| | MDFPP32:FPT_AEX_EXT.2: Memory Page Permissions |
| | MDFPP32:FPT_AEX_EXT.3: Stack Overflow Protection |
| | MDFPP32:FPT_AEX_EXT.4: Domain Isolation |
| | MDFPP32:FPT_AEX_EXT.5: Kernel Address Space Layout Randomization |
| | MDFPP32:FPT_BBD_EXT.1: Application Processor Mediation |
| | MDFPP32:FPT_JTA_EXT.1: JTAG Disablement |
| | MDFPP32:FPT_KST_EXT.1: Key Storage |
| | MDFPP32:FPT_KST_EXT.2: No Key Transmission |
| | MDFPP32:FPT_KST_EXT.3: No Plaintext Key Export |
| | MDFPP32:FPT_NOT_EXT.1: Self-Test Notification |
| | MDFPP32:FPT_STM.1: Reliable time stamps |
| | MDFPP32:FPT_TST_EXT.1: TSF Cryptographic Functionality Testing |
| | WLANCEP10:FPT_TST_EXT.1/WLAN: TSF Cryptographic Functionality Testing (Wireless LAN) |
| | MDFPP32:FPT_TST_EXT.2/POSTKERNEL: TSF Integrity Checking (Post-Kernel) |
| | MDFPP32:FPT_TST_EXT.2/PREKERNEL: TSF Integrity Checking (Pre-Kernel) |
| | MDFPP32:FPT_TUD_EXT.1: Trusted Update: TSF Version Query |
| | MDFPP32:FPT_TUD_EXT.2: TSF Update Verification |
| | MDFPP32:FPT_TUD_EXT.3: Application Signing |
| **FTA: TOE access** | MDFPP32:FTA_SSL_EXT.1: TSF- and User-initiated Locked State |
| | MDFPP32:FTA_TAB.1: Default TOE Access Banners |
| | WLANCEP10:FTA_WSE_EXT.1: Wireless Network Access |
| **FTP: Trusted path/channels** | BT10:FTP_BLT_EXT.1: Bluetooth Encryption |
| | BT10:FTP_BLT_EXT.2: Persistence of Bluetooth Encryption |
| | BT10:FTP_BLT_EXT.3/BR: Bluetooth Encryption Parameters (BR/EDR) |
| | BT10:FTP_BLT_EXT.3/LE: Bluetooth Encryption Parameters (LE) |
| | MDFPP32:FTP_ITC_EXT.1: Trusted Channel Communication |
| | WLANCEP10:FTP_ITC_EXT.1/WLAN: Trusted Channel Communication (Wireless LAN) |

**Table 1 TOE Security Functional Components**

## 5.1.1 Security audit (FAU)

### 5.1.1.1 Audit Data Generation (MDFPP32/BT10/WLANEP10:FAU_GEN.1)

**MDFPP32/BT10/WLANEP10:FAU_GEN.1.1**

The TSF shall be able to generate an audit record of the following auditable events:
1. Start-up and shutdown of the audit functions;
2. All auditable events for the not specified level of audit;
3. All administrative actions;
4. Start-up and shutdown of the OS;
5. Insertion or removal of removable media;
6. Specifically defined auditable events in Table 2;
7. [***no additional auditable events***].

**MDFPP32/BT10/WLANEP10:FAU_GEN.1.2**

The TSF shall record within each audit record at least the following information:
1. Date and time of the event;
2. Type of event;
3. Subject identity;
4. The outcome (success or failure) of the event;
5. Additional information in Table 2;
6. For each audit event type, based on the auditable event definitions of the functional components included in the PP/ST, [***no additional information***].

| Requirement | Audit Event | Additional Contents |
|---|---|---|
| **FAU_GEN.1** | | |
| **FAU_SAR.1** | | |
| **FAU_STG.1** | None. | |
| **FAU_STG.4** | None. | |
| **FCS_CKM.1** | [***None***]. | No additional information. |
| **FCS_CKM.1/WLAN** | | |
| **FCS_CKM.2/LOCKED** | None. | |
| **FCS_CKM.2/UNLOCKED** | None. | |
| **FCS_CKM.2/WLAN** | | |
| **FCS_CKM_EXT.1** | [***None***]. | |
| **FCS_CKM_EXT.2** | None. | |
| **FCS_CKM_EXT.3** | None. | |
| **FCS_CKM_EXT.4** | None. | |
| **FCS_CKM_EXT.5** | [***None***]. | |
| **FCS_CKM_EXT.6** | | |
| **FCS_CKM_EXT.7** | | |
| **FCS_CKM_EXT.8** | | |
| **FCS_COP.1/CONDITION** | None. | |
| **FCS_COP.1/ENCRYPT** | None. | |
| **FCS_COP.1/HASH** | None. | |
| **FCS_COP.1/KEYHMAC** | None. | |
| **FCS_COP.1/SIGN** | None. | |
| **FCS_HTTPS_EXT.1** | Failure of the certificate validity check. | Issuer Name and Subject Name of certificate. [***No additional information***]. |
| **FCS_IV_EXT.1** | None. | |
| **FCS_RBG_EXT.1** | | |

| | | |
|---|---|---|
| **FCS_SRV_EXT.1** | None. | |
| **FCS_STG_EXT.1** | Import or destruction of key. [***No other events***]. | Identity of key. Role and identity of requestor. |
| **FCS_STG_EXT.2** | None. | |
| **FCS_STG_EXT.3** | Failure to verify integrity of stored key. | Failure to verify integrity of stored key. |
| **FCS_TLS_EXT.1** | | |
| **FCS_TLSC_EXT.1** | | |
| **FCS_TLSC_EXT.1/WLAN** | Failure to establish an EAP-TLS session.<br>Establishment/termination of an EAP-TLS session. | Reason for failure. Non-TOE endpoint of connection. |
| **FCS_TLSC_EXT.2** | | |
| **FCS_TLSC_EXT.2/WLAN** | | |
| **FCS_TLSC_EXT.3** | | |
| **FCS_TLSC_EXT.4** | | |
| **FCS_TLSC_EXT.5** | | |
| **FDP_ACF_EXT.1** | | |
| **FDP_ACF_EXT.2** | | |
| **FDP_ACF_EXT.3** | | |
| **FDP_BCK_EXT.1** | | |
| **FDP_BLT_EXT.1** | | |
| **FDP_DAR_EXT.1** | [***None***]. | No additional information |
| **FDP_DAR_EXT.2** | Failure to encrypt/decrypt data. | No additional information |
| **FDP_IFC_EXT.1** | None. | |
| **FDP_STG_EXT.1** | Addition or removal of certificate from Trust Anchor Database. | Subject name of certificate. |
| **FDP_UPC_EXT.1/APPS** | | |
| **FDP_UPC_EXT.1/BLUETOOTH** | | |
| **FIA_AFL_EXT.1** | | |
| **FIA_BLT_EXT.1** | User authorization of Bluetooth device. User authorization for local Bluetooth service. | User authorization decision. ***[complete]*** BD_ADDR and ***[no other information].***<br><br>Bluetooth profile. Identity of local service with ***[profile name].*** |
| **FIA_BLT_EXT.2** | Initiation of Bluetooth connection. Failure of Bluetooth connection. | ***[complete]*** BD_ADDR and ***[no other information]***.<br><br>Reason for failure. |
| **FIA_BLT_EXT.3** | Duplicate connection attempt. | ***[complete]*** BD_ADDR of connection attempt. |
| **FIA_BLT_EXT.4** | None. | |
| **FIA_BLT_EXT.6** | None. | |
| **FIA_BLT_EXT.7** | None. | |
| **FIA_PAE_EXT.1** | | |
| **FIA_PMG_EXT.1** | None. | |
| **FIA_TRT_EXT.1** | None. | |
| **FIA_UAU.5** | None. | |
| **FIA_UAU.6/CREDENTIAL** | | |
| **FIA_UAU.6/LOCKED** | | |
| **FIA_UAU.7** | None. | |

| | | |
|---|---|---|
| **FIA_UAU_EXT.1** | None. | |
| **FIA_UAU_EXT.2** | | |
| **FIA_X509_EXT.1** | Failure to validate X.509v3 certificate. | Reason for failure of validation. |
| **FIA_X509_EXT.1/WLAN** | Failure to validate X.509v3 certificate. | Reason for failure of validation. |
| **FIA_X509_EXT.2** | | |
| **FIA_X509_EXT.2/WLAN** | | |
| **FIA_X509_EXT.3** | | |
| **FMT_MOF_EXT.1** | None. | |
| **FMT_SMF_EXT.1** | [***None***]. | |
| **FMT_SMF_EXT.1/BT** | | |
| **FMT_SMF_EXT.1/WLAN** | | |
| **FMT_SMF_EXT.2** | | |
| **FMT_SMF_EXT.3** | | |
| **FPT_AEX_EXT.1** | None. | |
| **FPT_AEX_EXT.2** | None. | |
| **FPT_AEX_EXT.3** | None. | |
| **FPT_AEX_EXT.4** | | |
| **FPT_AEX_EXT.5** | | |
| **FPT_BBD_EXT.1** | | |
| **FPT_BLT_EXT.1** | | |
| **FPT_JTA_EXT.1** | None. | |
| **FPT_KST_EXT.1** | None. | |
| **FPT_KST_EXT.2** | None. | |
| **FPT_KST_EXT.3** | None. | |
| **FPT_NOT_EXT.1** | [***None***]. | [***No additional information***]. |
| **FPT_STM.1** | None. | |
| **FPT_TST_EXT.1** | Initiation of self-test. Failure of self-test. | [***none***]. |
| **FPT_TST_EXT.1/WLAN** | None. | |
| **FPT_TST_EXT.2/POSTKERNEL** | | |
| **FPT_TST_EXT.2/PREKERNEL** | Start-up of TOE. [***none***]. | No additional information. [***No additional information***]. |
| **FPT_TST_EXT.3** | None. | |
| **FPT_TUD_EXT.1** | . | |
| **FPT_TUD_EXT.2** | None. | |
| **FPT_TUD_EXT.3** | | |
| **FPT_TUD_EXT.4** | | |
| **FTA_SSL_EXT.1** | None. | |
| **FTA_TAB.1** | | |
| **FTA_WSE_EXT.1** | All attempts to connect to access points. | Identity of access point being connected to as well as success and failures (including reason for failure). |
| **FTP_BLT_EXT.1** | None. | |
| **FTP_BLT_EXT.2** | None. | |
| **FTP_BLT_EXT.3/BR** | None. | |
| **FTP_BLT_EXT.3/LE** | None. | |
| **FTP_ITC_EXT.1** | | |
| **FTP_ITC_EXT.1/WLAN** | All attempts to establish a trusted channel. [deleted by TD0194: | Identification of the non-TOE endpoint of the channel. |

| | | |
|---|---|---|
| | Detection of modification of channel data.] | |

**Table 2 Audit Events**

#### 5.1.1.2 Audit Review (MDFPP32:FAU_SAR.1)

**MDFPP32:FAU_SAR.1.1**
The TSF shall provide the administrator with the capability to read all audited events and record contents from the audit records.

**MDFPP32:FAU_SAR.1.2**
The TSF shall provide the audit records in a manner suitable for the user to interpret the information.

#### 5.1.1.3 Audit Storage Protection (MDFPP32:FAU_STG.1)

**MDFPP32:FAU_STG.1.1**
The TSF shall protect the stored audit records in the audit trail from unauthorized deletion.

**MDFPP32:FAU_STG.1.2**
The TSF shall be able to prevent unauthorized modifications to the stored audit records in the audit trail.

#### 5.1.1.4 Prevention of Audit Data Loss (MDFPP32:FAU_STG.4)

**MDFPP32:FAU_STG.4.1**
The TSF shall overwrite the oldest stored audit records if the audit trail is full.

### 5.1.2 Cryptographic support (FCS)

#### 5.1.2.1 Cryptographic Key Generation (MDFPP32:FCS_CKM.1)

**MDFPP32:FCS_CKM.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 FIPS PUB 186-4, 'Digital Signature Standard (DSS)', Appendix B.3,***
- ***ECC schemes using ['NIST curves' P-384 and [P-256, P-521] that meet the following: FIPS PUB 186-4, 'Digital Signature Standard (DSS)', Appendix B.4]***].

#### 5.1.2.2 Cryptographic Key Generation (Symmetric Keys for WPA2 Connections) (WLANCEP10:FCS_CKM.1/WLAN)

**WLANCEP10:FCS_CKM.1.1/WLAN**
Refinement: The TSF shall generate symmetric cryptographic keys in accordance with a specified cryptographic key generation algorithm PRF-384 and [***no other***] and specified cryptographic key sizes 128 bits and [***no other key sizes***] using a Random Bit Generator as specified in FCS_RBG_EXT.1 that meet the following: IEEE 802.11-2012 and [***no other standards***].

#### 5.1.2.3 Cryptographic Key Establishment (MDFPP32:FCS_CKM.2/LOCKED)

**MDFPP32:FCS_CKM.2.1/LOCKED**
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'***]
for the purposes of encrypting sensitive data received while the device is locked.

#### 5.1.2.4 Cryptographic Key Establishment (MDFPP32:FCS_CKM.2/UNLOCKED)

**MDFPP32:FCS_CKM.2.1/UNLOCKED**
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 [RSAES-PKCS1-v1_5 as specified in Section 7.2 of RFC 8017, 'Public-Key Cryptography Standards (PKCS) #1:RSA Cryptography Specifications Version 2.2'],***
***- Elliptic curve-based key establishment schemes that meets the following: NIST Special Publication 800-56A Revison 3, 'Recommendation for Pair-Wise Key Establishment Schemes Using Discrete Logarithm Cryptography'***].

#### 5.1.2.5 Cryptographic Key Distribution (GTK) (WLANCEP10:FCS_CKM.2/WLAN)

**WLANCEP10:FCS_CKM.2.1/WLAN**
Refinement: The TSF shall decrypt Group Temporal Key in accordance with a specified cryptographic key distribution method AES Key Wrap in an EAPOL-Key frame that meets the following: RFC 3394 for AES Key Wrap, 802.11-2012 for the packet format and timing considerations and does not expose the cryptographic keys.

#### 5.1.2.6 Cryptographic Key Support (MDFPP32:FCS_CKM_EXT.1)

**MDFPP32:FCS_CKM_EXT.1.1**
The TSF shall support [***immutable hardware***] REK(s) with a [***symmetric***] key of strength [***256 bits***].
**MDFPP32:FCS_CKM_EXT.1.2**
Each REK shall be hardware-isolated from the OS on the TSF in runtime.
**MDFPP32:FCS_CKM_EXT.1.3**
Each REK shall be generated by a RBG in accordance with FCS_RBG_EXT.1.

#### 5.1.2.7 Cryptographic Key Random Generation (MDFPP32:FCS_CKM_EXT.2)

**MDFPP32: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.

#### 5.1.2.8 Cryptographic Key Generation (MDFPP32:FCS_CKM_EXT.3)

**MDFPP32:FCS_CKM_EXT.3.1**
The TSF shall use [***asymmetric KEKs of [128 bits] security strength, symmetric KEKs of [256-bit] security strength corresponding to at least the security strength of the keys encrypted by the KEK***].
**MDFPP32:FCS_CKM_EXT.3.2**
The TSF shall generate all KEKs using one of the following methods:
- Derive the KEK from a Password Authentication Factor using according to FCS_COP.1.1/CONDITION and [***Generate the KEK using an RBG that meets this profile (as specified in FCS_RBG_EXT.1), Generate the KEK using a key generation scheme that meets this profile (as specified in FCS_CKM.1), Combine the KEK from other KEKs in a way that preserves the effective entropy of each factor by [concatenating the keys and using a KDF (as described in SP 800-108), encrypting one key with another]***].

#### 5.1.2.9 Key Destruction (MDFPP32:FCS_CKM_EXT.4)

**MDFPP32: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
  - For volatile memory, the destruction shall be executed by a single direct overwrite [***consisting of zeroes***].
  - 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 by a read-verify.
  - For non-volatile flash memory, that is not wear-leveled, the destruction shall be executed [***by a block erase that erases the reference to memory that stores data as well as the data itself***].
  - For non-volatile flash memory, that is wear-leveled, the destruction shall be executed [***by a block erase***].
  - For non-volatile memory other than EEPROM and flash, the destruction shall be executed by a single direct overwrite with a random pattern that is changed before each write.

**MDFPP32:FCS_CKM_EXT.4.2**

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

#### 5.1.2.10 TSF Wipe (MDFPP32:FCS_CKM_EXT.5)

**MDFPP32:FCS_CKM_EXT.5.1**

The TSF shall wipe all protected data by [***Cryptographically erasing the encrypted DEKs and/or the KEKs in nonvolatile memory by following the requirements in FCS_CKM_EXT.4.1, Overwriting all Protected Data according to the following rules:***
***-- For 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 by a read-verify.***
***-- For flash memory, that is not wear-leveled, the destruction shall be executed [by a single direct overwrite consisting of zeros followed by a read-verify].***
***-- For flash memory, that is wear-leveled, the destruction shall be executed [by a block erase].***
***-- For non-volatile memory other than EEPROM and flash, the destruction shall be executed by a single direct overwrite with a random pattern that is changed before each write.***].

**MDFPP32:FCS_CKM_EXT.5.2**

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

#### 5.1.2.11 Salt Generation (MDFPP32:FCS_CKM_EXT.6)

**MDFPP32:FCS_CKM_EXT.6.1**

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

#### 5.1.2.12 Bluetooth Key Generation (BT10:FCS_CKM_EXT.8)

**BT10:FCS_CKM_EXT.8.1**

The TSF shall generate public/private ECDH key pairs every [**Bluetooth connection establishment**].

#### 5.1.2.13 Cryptographic Operation (MDFPP32:FCS_COP.1/CONDITION)

**MDFPP32:FCS_COP.1.1/CONDITION**

The TSF shall perform conditioning in accordance with a specified cryptographic algorithm

HMAC-[***SHA-256***] using a salt, and [***[key stretching with scrypt]***] and output cryptographic key sizes [***256***] that meet the following: [***no standard***].

#### 5.1.2.14 Cryptographic Operation (MDFPP32:FCS_COP.1/ENCRYPT)

**MDFPP32:FCS_COP.1.1/ENCRYPT**

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-XTS (as defined in NIST SP 800-38E) mode, no other modes***]
and cryptographic key sizes 128-bit key sizes and [***256-bit key sizes***].

#### 5.1.2.15 Cryptographic Operation (MDFPP32:FCS_COP.1/HASH)

**MDFPP32:FCS_COP.1.1/HASH**

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 bits***] that meet the following: FIPS Pub 180-4.

#### 5.1.2.16 Cryptographic Operation (MDFPP32:FCS_COP.1/KEYHMAC)

**MDFPP32:FCS_COP.1.1/KEYHMAC**

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'.

#### 5.1.2.17 Cryptographic Operation (MDFPP32:FCS_COP.1/SIGN)

**MDFPP32:FCS_COP.1.1/SIGN**

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,***
***-ECDSA schemes using 'NIST curves' P-384 and [P-256, P-521] that meet the following: FIPS PUB 186-4, 'Digital Signature Standard (DSS)', Section 5***].

#### 5.1.2.18 HTTPS Protocol (MDFPP32:FCS_HTTPS_EXT.1)

**MDFPP32:FCS_HTTPS_EXT.1.1**

The TSF shall implement the HTTPS protocol that complies with RFC 2818.

**MDFPP32:FCS_HTTPS_EXT.1.2**

The TSF shall implement HTTPS using TLS as defined in the Package for Transport Layer Security.

**MDFPP32:FCS_HTTPS_EXT.1.3**

The TSF shall notify the application and [***not establish the connection***] if the peer certificate is deemed invalid.

#### 5.1.2.19 Initialization Vector Generation (MDFPP32:FCS_IV_EXT.1)

**MDFPP32:FCS_IV_EXT.1.1**

The TSF shall generate IVs in accordance with Table 13: References and IV Requirements for NIST-approved Cipher Modes.

| Cipher Mode | Reference | IV Requirements |
|---|---|---|
| Electronic Codebook (ECB) | SP 800-38A | No IV |
| Counter (CTR) | SP 800-38A | 'Initial Counter' shall be non-repeating. No counter value shall be repeated across multiple messages with the same secret key. |
| Cipher Block Chaining (CBC) | SP 800-38A | IVs shall be unpredictable. Repeating IVs leak information about whether the first one or more blocks are shared between two messages, so IVs should be non-repeating in such situations. |
| Output Feedback (OFB) | SP 800-38A | IVs shall be non-repeating and shall not be generated by invoking the cipher on another IV. |
| Cipher Feedback (CFB) | SP 800-38A | IVs should be non-repeating as repeating IVs leak information about the first plaintext block and about common shared prefixes in messages. |
| XEX (XOR Encrypt XOR) Tweakable Block Cipher with Ciphertext Stealing (XTS) | SP 800-38E | No IV. Tweak values shall be non-negative integers, assigned consecutively, and starting at an arbitrary non-negative integer. |
| Cipher-based Message Authentication Code (CMAC) | SP 800-38B | No IV |
| Key Wrap and Key Wrap with Padding | SP 800-38F | No IV |
| Counter with CBC-Message Authentication Code (CCM) | SP 800-38C | No IV. Nonces shall be non-repeating. |
| Galois Counter Mode (GCM) | SP 800-38D | IV shall be non-repeating. The number of invocations of GCM shall not exceed 2^32 for a given secret key unless an implementation only uses 96-bit IVs (default length). |

#### 5.1.2.20 Random Bit Generation (MDFPP32:FCS_RBG_EXT.1)

**MDFPP32:FCS_RBG_EXT.1.1**

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

**MDFPP32:FCS_RBG_EXT.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.

**MDFPP32:FCS_RBG_EXT.1.3**

The TSF shall be capable of providing output of the RBG to applications running on the TSF that request random bits.

#### 5.1.2.21 Cryptographic Algorithm Services (MDFPP32:FCS_SRV_EXT.1)

**MDFPP32: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/LOCKED
- The following algorithms in FCS_COP.1/ENCRYPT: AES-CBC, [***AES-GCM***]
- All mandatory and selected algorithms in FCS_COP.1/SIGN
- All mandatory and selected algorithms in FCS_COP.1/HASH
- All mandatory and selected algorithms in FCS_COP.1/KEYHMAC

[***No other cryptographic operations***].

#### 5.1.2.22 Cryptographic Algorithm Services (MDFPP32:FCS_SRV_EXT.2)

**MDFPP32:FCS_SRV_EXT.1.1**
The TSF shall provide a mechanism for applications to request the TSF to perform the following cryptographic operations:
- Algorithms in FCS_COP.1/ENCRYPT
- Algorithms in FCS_COP.1/SIGN
by keys stored in the secure key storage.

#### 5.1.2.23 Cryptographic Key Storage (MDFPP32:FCS_STG_EXT.1)

**MDFPP32:FCS_STG_EXT.1.1**
The TSF shall provide [***software-based***] secure key storage for asymmetric private keys and [***symmetric keys, persistent secrets***].

**MDFPP32: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***].

**MDFPP32:FCS_STG_EXT.1.3**
The TSF shall be capable of destroying keys/secrets in the secure key storage upon request of [***the user, the administrator***].

**MDFPP32: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***].

**MDFPP32: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***].

#### 5.1.2.24 Encrypted Cryptographic Key Storage (MDFPP32:FCS_STG_EXT.2)

**MDFPP32:FCS_STG_EXT.2.1**
The TSF shall encrypt all DEKs, KEKs, [**Wi-Fi and Bluetooth**] and [***all software-based key storage***] by KEKs that are
[***Protected by the REK with [encryption by a REK, encryption by a KEK that is derived from a REK], Protected by the REK and the password with [encryption by a REK and the password-derived KEK, encryption by a KEK that is derived from a REK and the password-derived or biometric-unlocked KEK]***].

**MDFPP32:FCS_STG_EXT.2.2**
DEKs, KEKs, [**Wi-Fi and Bluetooth**] and [***all software-based key storage***] shall be encrypted using one of the following methods:
[***using a SP800-56B key establishment scheme, using AES in the [GCM, CCM]***].

#### 5.1.2.25 Integrity of Encrypted Key Storage (MDFPP32:FCS_STG_EXT.3)

**MDFPP32:FCS_STG_EXT.3.1**
The TSF shall protect the integrity of any encrypted DEKs and KEKs and [***long-term trusted channel key material, all software-based key storage***] by [***[GCM, CCM] cipher mode for encryption according to FCS_STG_EXT.2***].

**MDFPP32:FCS_STG_EXT.3.2**
The TSF shall verify the integrity of the [***MAC***] of the stored key prior to use of the key.

#### 5.1.2.26 TLS Protocol (PKGTLS11:FCS_TLS_EXT.1)

**PKGTLS11:FCS_TLS_EXT.1.1**
The product shall implement [***TLS as a client,***]

#### 5.1.2.27 TLS Client Protocol (PKGTLS11:FCS_TLSC_EXT.1)

**PKGTLS11:FCS_TLSC_EXT.1.1**
The product shall implement TLS 1.2 (RFC 5246) and [***no earlier TLS versions***] as a client that supports the cipher suites [
***TLS_RSA_WITH_AES_256_GCM_SHA384 as defined in RFC 5288,***
***TLS_ECDHE_ECDSA_WITH_AES_128_GCM_SHA256 as defined in RFC 5289,***
***TLS_ECDHE_ECDSA_WITH_AES_256_GCM_SHA384 as defined in RFC 5289,***
***TLS_ECDHE_RSA_WITH_AES_128_GCM_SHA256 as defined in RFC 5289,***
***TLS_ECDHE_RSA_WITH_AES_256_GCM_SHA384 as defined in RFC 5289***]
and also supports functionality for [***mutual authentication, session renegotiation***] (TD0442 applied) (TD0588 applied)

**PKGTLS11:FCS_TLSC_EXT.1.2**
The product shall verify that the presented identifier matches the reference identifier according to RFC 6125.

**PKGTLS11:FCS_TLSC_EXT.1.3**
The product shall not establish a trusted channel if the server certificate is invalid [***with no exceptions***]

#### 5.1.2.28 Extensible Authentication Protocol-Transport Layer Security (WLANCEP10:FCS_TLSC_EXT.1/WLAN)

**WLANCEP10:FCS_TLSC_EXT.1.1/WLAN**
The TSF shall implement [***TLS 1.0 (RFC 2246), TLS 1.1 (RFC 4346), TLS 1.2 (RFC 5246)***] in support of the EAP-TLS protocol as specified in RFC 5216 supporting the following ciphersuites:
[***TLS_RSA_WITH_AES_128_CBC_SHA as defined in RFC 5246,***
***TLS_RSA_WITH_AES_256_CBC_SHA as defined in RFC 5246,***
***TLS_RSA_WITH_AES_128_GCM_SHA256 as defined in RFC 5288,***
***TLS_RSA_WITH_AES_256_GCM_SHA384 as defined in RFC 5288,***
***TLS_ECDHE_ECDSA_WITH_AES_128_GCM_SHA256 as defined in RFC 5289,***
***TLS_ECDHE_ECDSA_WITH_AES_256_GCM_SHA384 as defined in RFC 5289,***
***TLS_ECDHE_RSA_WITH_AES_128_GCM_SHA256 as defined in RFC 5289,***
***TLS_ECDHE_RSA_WITH_AES_256_GCM_SHA384 as defined in RFC 5289***].
(TD0492 applied)

**WLANCEP10:FCS_TLSC_EXT.1.2/WLAN**
The TSF shall generate random values used in the EAP-TLS exchange using the RBG specified in FCS_RBG_EXT.1.

**WLANCEP10:FCS_TLSC_EXT.1.3/WLAN**
The TSF shall use X509 v3 certificates as specified in FIA_X509_EXT.1/WLAN. (TD0517 applied)

**WLANCEP10:FCS_TLSC_EXT.1.4/WLAN**
The TSF shall verify that the server certificate presented includes the Server Authentication purpose (id-kp 1 with OID 1.3.6.1.5.5.7.3.1) in the extendedKeyUsage field.

**WLANCEP10:FCS_TLSC_EXT.1.5/WLAN**
The TSF shall allow an authorized administrator to configure the list of CAs that are allowed to sign authentication server certificates that are accepted by the TOE.

**WLANCEP10:FCS_TLSC_EXT.1.6/WLAN**
Removed by TD0492.

#### 5.1.2.29 TLS Client Support for Mutual Authentication (PKGTLS11:FCS_TLSC_EXT.2)

**PKGTLS11:FCS_TLSC_EXT.2.1**
The product shall support mutual authentication using X.509v3 certificates.

#### 5.1.2.30 TLS Client Protocol (WLANCEP10:FCS_TLSC_EXT.2/WLAN)

**WLANCEP10:FCS_TLSC_EXT.2.1/WLAN**
The TSF shall present the Supported Elliptic Curves Extension in the Client Hello with the following NIST curves: [***secp256r1, secp384r1***]. (TD0244 applied)

#### 5.1.2.31 TLS Client Support for Renegotiation (PKGTLS11:FCS_TLSC_EXT.4)

**PKGTLS11:FCS_TLSC_EXT.4.1**
The product shall support secure renegotiation through use of the 'renegotiation_info' TLS extension in accordance with RFC 5746.

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

**PKGTLS11:FCS_TLSC_EXT.5.1**
The product shall present the Supported Groups Extension in the Client Hello with the supported groups [***secp256r1, secp384r1***]

### 5.1.3 User data protection (FDP)

#### 5.1.3.1 Access Control for System Services (MDFPP32:FDP_ACF_EXT.1)

**MDFPP32:FDP_ACF_EXT.1.1**
The TSF shall provide a mechanism to restrict the system services that are accessible to an application.
**MDFPP32:FDP_ACF_EXT.1.2**
The TSF shall provide an access control policy that prevents [***application, groups of applications***] from accessing [***all***] data stored by other [***application, groups of applications***]. Exceptions may only be explicitly authorized for such sharing by [***a common application developer, no one***].

#### 5.1.3.2 Access Control for System Resources (MDFPP32:FDP_ACF_EXT.2)

**MDFPP32:FDP_ACF_EXT.2.1**
The TSF shall provide a separate [***address book, calendar, [keychain]***] for each application group and only allow applications within that process group to access the resource. Exceptions may only be explicitly authorized for such sharing by [***the administrator, no one***].

#### 5.1.3.3 Protected Data Encryption (MDFPP32:FDP_DAR_EXT.1)

**MDFPP32:FDP_DAR_EXT.1.1**
Encryption shall cover all protected data.
**MDFPP32:FDP_DAR_EXT.1.2**
Encryption shall be performed using DEKs with AES in the [***XTS***] mode with key size [***256***] bits.

#### 5.1.3.4 Sensitive Data Encryption (MDFPP32:FDP_DAR_EXT.2)

**MDFPP32:FDP_DAR_EXT.2.1**
The TSF shall provide a mechanism for applications to mark data and keys as sensitive.
**MDFPP32: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.
**MDFPP32: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.
**MDFPP32: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.

#### 5.1.3.5 Subset Information Flow Control - per TD0596 (MDFPP32:FDP_IFC_EXT.1)

**MDFPP32:FDP_IFC_EXT.1.1**

The TSF shall [***provide an interface which allows a VPN client to protect all IP traffic using IPsec***] with the exception of IP traffic needed to manage the VPN connection, and [***[ICMP responses on local subnet, captive portion traffic needed for the correct functioning of the TOE]***], when the VPN is enabled.

#### 5.1.3.6 User Data Storage (MDFPP32:FDP_STG_EXT.1)

**MDFPP32:FDP_STG_EXT.1.1**

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

#### 5.1.3.7 Inter-TSF User Data Transfer Protection (Applications) (MDFPP32:FDP_UPC_EXT.1/APPS)

**MDFPP32:FDP_UPC_EXT.1.1/APPS**

The TSF shall provide a means for non-TSF applications executing on the TOE to use
- mutually authenticated TLS as defined in the Package for Transport Layer Security,
- HTTPS,
and
- [***no other protocol***]
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.

**MDFPP32:FDP_UPC_EXT.1.2/APPS**

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

#### 5.1.3.8 Inter-TSF User Data Transfer Protection (Bluetooth) (MDFPP32:FDP_UPC_EXT.1/BLUETOOTH)

**MDFPP32:FDP_UPC_EXT.1.1/BLUETOOTH**

The TSF shall provide a means for non-TSF applications executing on the TOE to use
- Bluetooth BR/EDR in accordance with the PP-Module for Bluetooth,
and
- [***Bluetooth LE in accordance with the PP-Module for Bluetooth***]
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.

**MDFPP32:FDP_UPC_EXT.1.2/BLUETOOTH**

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

### 5.1.4 Identification and authentication (FIA)

#### 5.1.4.1 Authentication Failure Handling (MDFPP32:FIA_AFL_EXT.1)

**MDFPP32:FIA_AFL_EXT.1.1**

The TSF shall consider password and [***no other***] as critical authentication mechanisms.

**MDFPP32:FIA_AFL_EXT.1.2**

The TSF shall detect when a configurable positive integer within [**0-50**] of [***non-unique***] unsuccessful authentication attempts occur related to last successful authentication for each authentication mechanism.

**MDFPP32:FIA_AFL_EXT.1.3**
The TSF shall maintain the number of unsuccessful authentication attempts that have occurred upon power off.

**MDFPP32:FIA_AFL_EXT.1.4**
When the defined number of unsuccessful authentication attempts has exceeded the maximum allowed for a given authentication mechanism, all future authentication attempts will be limited to other available authentication mechanisms, unless the given mechanism is designated as a critical authentication mechanism.

**MDFPP32:FIA_AFL_EXT.1.5**
When the defined number of unsuccessful authentication attempts for the last available authentication mechanism or single critical authentication mechanism has been surpassed, the TSF shall perform a wipe of all protected data.

**MDFPP32:FIA_AFL_EXT.1.6**
The TSF shall increment the number of unsuccessful authentication attempts prior to notifying the user that the authentication was unsuccessful.

#### 5.1.4.2 Bluetooth User Authorization (BT10:FIA_BLT_EXT.1)

**BT10:FIA_BLT_EXT.1.1**
The TSF shall require explicit user authorization before pairing with a remote Bluetooth device.

#### 5.1.4.3 Bluetooth Mutual Authentication (BT10:FIA_BLT_EXT.2)

**BT10:FIA_BLT_EXT.2.1**
The TSF shall require Bluetooth mutual authentication between devices prior to any data transfer over the Bluetooth link.

#### 5.1.4.4 Rejection of Duplicate Bluetooth Connections (BT10:FIA_BLT_EXT.3)

**BT10:FIA_BLT_EXT.3.1**
The TSF shall discard pairing and session initialization attempts from a Bluetooth device address (BD_ADDR) to which an active session already exists.

#### 5.1.4.5 Secure Simple Pairing (BT10:FIA_BLT_EXT.4)

**BT10:FIA_BLT_EXT.4.1**
The TOE shall support Bluetooth Secure Simple Pairing, both in the host and the controller.

**BT10:FIA_BLT_EXT.4.2**
The TOE shall support Secure Simple Pairing during the pairing process.

#### 5.1.4.6 Trusted Bluetooth Device User Authorization (BT10:FIA_BLT_EXT.6)

**BT10:FIA_BLT_EXT.6.1**
The TSF shall require explicit user authorization before granting trusted remote devices access to services associated with the following Bluetooth profiles: [**OPP, MAP**].

#### 5.1.4.7 Untrusted Bluetooth Device User Authorization (BT10:FIA_BLT_EXT.7)

**BT10:FIA_BLT_EXT.7.1**
The TSF shall require explicit user authorization before granting untrusted remote devices access to services associated with the following Bluetooth profiles: [**OPP, MAP**].

#### 5.1.4.8 Port Access Entity Authentication (WLANCEP10:FIA_PAE_EXT.1)

**WLANCEP10:FIA_PAE_EXT.1.1**
The TSF shall conform to IEEE Standard 802.1X for a Port Access Entity (PAE) in the 'Supplicant' role.

#### 5.1.4.9 Password Management (MDFPP32:FIA_PMG_EXT.1)

**MDFPP32: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.

#### 5.1.4.10 Authentication Throttling (MDFPP32:FIA_TRT_EXT.1)

**MDFPP32:FIA_TRT_EXT.1.1**

The TSF shall limit automated user authentication attempts by [***enforcing a delay between incorrect authentication attempts***] for all authentication mechanisms selected in FIA_UAU.5.1. The minimum delay shall be such that no more than 10 attempts can be attempted per 500 milliseconds.

#### 5.1.4.11 Multiple Authentication Mechanisms (MDFPP32:FIA_UAU.5)

**MDFPP32:FIA_UAU.5.1**

The TSF shall provide password and [***no other mechanism***] to support user authentication.

**MDFPP32:FIA_UAU.5.2**

The TSF shall authenticate any user's claimed identity according to the [**To authenticate unlocking the device immediately after boot (first unlock after reboot):**
- **User passwords are required after reboot to unlock the user's Credential encrypted (CE files) and keystore keys.**

**To authenticate unlocking the device after device lock (not following a reboot):**
- **The TOE verifies user credentials (password) via the gatekeeper trusted application (running inside the Trusted Execution Environment, TEE), which compares the entered credential to a derived value.**

**To change protected settings or issue certain commands:**
- **The TOE requires password after a reboot, when changing settings (Screen lock and Smart Lock settings), and when factory resetting.**].

#### 5.1.4.12 Re-authenticating (Credential Change) (MDFPP32: FIA_UAU.6/CREDENTIAL) – per TD0706

**MDFPP32:FIA_UAU.6.1/CREDENTIAL**

The TSF shall re-authenticate the user via the Password Authentication Factor under the conditions attempted change to any supported authentication mechanisms..

#### 5.1.4.13 Re-authenticating (TSF Lock) (MDFPP32:FIA_UAU.6/LOCKED) – per TD0706

**MDFPP32:FIA_UAU.6.1/LOCKED**

The TSF shall re-authenticate the user via an authentication factor defined in FIA_UAU.5.1 under the conditions TSF-initiated lock, user-initiated lock, [**no other conditions**].

#### 5.1.4.14 Protected Authentication Feedback (MDFPP32:FIA_UAU.7)

**MDFPP32: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.

#### 5.1.4.15 Authentication for Cryptographic Operation (MDFPP32:FIA_UAU_EXT.1)

**MDFPP32: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 [***long-term trusted channel key material, all software-based key storage***] at startup.

#### 5.1.4.16 Timing of Authentication (MDFPP32:FIA_UAU_EXT.2)

**MDFPP32:FIA_UAU_EXT.2.1**

The TSF shall allow [***/***

- ***Take screen shots (stored internally)***
- ***Enter password to unlock***
- ***Make/receive emergency calls***
- ***Take pictures (stored internally) - unless the camera was disabled***
- ***Turn the TOE off***
- ***Restart the TOE***
- ***Warm Battery Swap***
- ***Touch Mode operation toggle***
- ***Lockdown device***
- ***Enable Airplane mode***
- ***See notifications (note that some notifications identify actions, for example to view a screenshot; however, selecting those notifications highlights the password prompt and require the password to access that data)***
- ***Set the volume (up and down) for ringtone***
- ***Choosing of the keyboard input method***
- ***Make emergency phone calls***
- ***Receive calls***
- ***Access notification widgets (without authentication):***
  - ***Flashlight toggle***
  - ***Do not disturb toggle***
  - ***Auto rotate toggleSound (on, mute, vibrate)***
  - ***Night light filter toggle***
  - ***Wifi Toggle***
  - ***Bluetooth Toggle***
  - ***DND Toggle***
  - ***Battery Saver Toggle/***] on behalf of the user to be performed before the user is authenticated.

**MDFPP32: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.

#### 5.1.4.17 X.509 Validation of Certificates (MDFPP32:FIA_X509_EXT.1)

**MDFPP32: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, that the CA flag is set to TRUE for all CA certificates, and that any path constraints are met.
- The TSF shall validate that any CA certificate includes caSigning purpose in the key usage field.
- The TSF shall validate the revocation status of the certificate using [***OCSP as specified in RFC 6960***]
- The TSF shall validate the extendedKeyUsage field according to the following rules:

- Certificates used for trusted updates and executable code integrity verification shall have the Code Signing purpose (id-kp 3 with OID 1.3.6.1.5.5.7.3.3) in the extendedKeyUsage field.
- Server certificates presented for TLS shall have the Server Authentication purpose (id-kp 1 with OID 1.3.6.1.5.5.7.3.1) in the extendedKeyUsage field.
- (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 EKU field.
- Client certificates presented for TLS shall have the Client Authentication purpose (id-kp 2 with OID 1.3.6.1.5.5.7.3.2) in the EKU field.
- OCSP certificates presented for OCSP responses shall have the OCSP Signing purpose (id-kp 9 with OID 1.3.6.1.5.5.7.3.9) in the EKU field. [conditional]

**MDFPP32: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.

#### 5.1.4.18 X.509 Certificate Validation (WLANCEP10:FIA_X509_EXT.1/WLAN)

**WLANCEP10:FIA_X509_EXT.1.1/WLAN**

The TSF shall validate certificates for EAP-TLS 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 extendedKeyUsage field according to the following rules:
Server certificates presented for TLS shall have the Server Authentication purpose (id-kp 1 with OID 1.3.6.1.5.5.7.3.1) in the extendedKeyUsage field
Client certificates presented for TLS shall have the Client Authentication purpose (id-kp 2 with OID 1.3.6.1.5.5.7.3.2) in the extendedKeyUsage field.
(TD0439 applied)

**WLANCEP10:FIA_X509_EXT.1.2/WLAN**

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. (TD0439 applied)

#### 5.1.4.19 X.509 Certificate Authentication - per TD0623 (MDFPP32:FIA_X509_EXT.2)

**MDFPP32:FIA_X509_EXT.2.1**

The TSF shall use X.509v3 certificates as defined by RFC 5280 to support authentication for mutually authenticated TLS as defined in the Package for Transport Layer Security, HTTPS, [***no other protocol***], and [***no additional uses***].

**MDFPP32:FIA_X509_EXT.2.2**

When the TSF cannot establish a connection to determine the revocation status of a certificate, the TSF shall [***not accept the certificate***].

#### 5.1.4.20 X.509 Certificate Authentication (EAP-TLS) (WLANCEP10:FIA_X509_EXT.2/WLAN)

**WLANCEP10:FIA_X509_EXT.2.1/WLAN**

The TSF shall use X.509v3 certificates as defined by RFC 5280 to support authentication for EAP-TLS exchanges.

**WLANCEP10:FIA_X509_EXT.2.2/WLAN**

FIA_X509_EXT.2.2 deleted by TD0517. (NOTE: FIA_X509_EXT.2.2/WLAN not in EP)

#### 5.1.4.21 Request Validation of Certificates (MDFPP32:FIA_X509_EXT.3)

**MDFPP32:FIA_X509_EXT.3.1**

The TSF shall provide a certificate validation service to applications.

**MDFPP32:FIA_X509_EXT.3.2**

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

## 5.1.5 Security management (FMT)

### 5.1.5.1 Management of Security Functions Behavior (MDFPP32:FMT_MOF_EXT.1)

**MDFPP32:FMT_MOF_EXT.1.1**

The TSF shall restrict the ability to perform the functions in column 3 of Table 6 to the user.

**MDFPP32:FMT_MOF_EXT.1.2**

The TSF shall restrict the ability to perform the functions in column 5 of Table 6 to the administrator when the device is enrolled and according to the administrator-configured policy.

### 5.1.5.2 Specification of Management Functions (MDFPP32:FMT_SMF_EXT.1)

**MDFPP32:FMT_SMF_EXT.1.1**

The TSF shall be capable of performing the following management functions: (Status Markers: M - Mandatory, O - Claimed Optional/Objective, X - Unclaimed Optional/Objective)

**Table 3 Security Management Functions**

| Management Function<br><br>**Status Markers:**<br>M – Mandatory<br>I – Implemented optional function | FMT_SMF_EXT.1.1 | FMT_MOF_EXT.1.1 | Administrator | FMT_MOF_EXT.1.2 |
|---|---|---|---|---|
| 1. configure password policy:<br>a. minimum password length<br>b. minimum password complexity<br>c. maximum password lifetime<br><br>The administrator can configure the required password characteristics (minimum length, complexity, and lifetime) using the Android MDM APIs.<br><br>Length: an integer value of characters<br>Complexity: Unspecified, Something, Numeric, Alphabetic, Alphanumeric, Complex.<br>Lifetime: an integer value of seconds (0 = no maximum). | M | | M | M |
| 2. configure session locking policy:<br>a. screen-lock enabled/disabled<br>b. screen lock timeout<br>c. number of authentication failures<br><br>The administrator can configure the session locking policy using the Android MDM APIs.<br>Screen lock timeout: an integer number of minutes before the TOE locks (0 = no lock timeout)<br>Authentication failures: an integer number (-2,147,483,648 to 2,147,483,648 [negative integers and zero means no limit]). | M | | M | M |
| 3. enable/disable the VPN protection:<br>a. across device<br>[ **c.** ***no other method***] | M | | I | I |

| | | | | |
|---|---|---|---|---|
| Both users (using the TOE's settings UI) and administrator (using the TOE's MDM APIs) can configure a third-party VPN client and then enable the VPN client to protect traffic. The User can set up VPN protection, but if an admin enables VPN protection, the user cannot disable it. | | | | |
| 4. enable/disable [**Bluetooth,**<br>**NFC, Wi-Fi, cellular**]<br><br>The administrator can disable the radios using the TOE's MDM APIs. Once Bluetooth is disabled, a user cannot enable it; all other radios can be re-enabled by the user. The TOE's radios operate at frequencies of 2.4 GHz (NFC/Bluetooth), 2.4/5 GHz (Wi-Fi), and 850, 900, 1800, 1900 MHz (4G/LTE). | M<br>M | I | I | I |
| 5. enable/disable [**microphone, camera**]:<br>a. across device (microphone,camera),<br>[ **b.** ***on a per-app basis (microphone, camera)***]<br><br>An administrator can enable/disable the device's microphone via an MDM API. Once the microphone has been disabled, the user cannot re-enable it until the administrator enables it.<br><br>In the user's settings, a user can view a permission by type (i.e. camera, microphone). The user can access this by going to "Settings" -> "App Permissions" -> Selecting the permission and revoking any applications. | M<br>M | | I | I |
| 6. transition to the locked state<br><br>Both users (using the TOE's settings UI) and administrators (using the TOE's MDM APIs) can transition the TOE into a locked state. | M | | M | |
| 7. full wipe of protected data<br><br>Both users (using the TOE's settings UI) and administrators (using the TOE's MDM APIs) can force the TOE to perform a full wipe (factory reset) of data. | M | | M | |
| 8. configure application installation policy by:<br>a. ***restricting the sources of applications,***<br>c***. denying installation of applications***]<br><br>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 Play Store. | M | | M | M |
| 9. import keys/secrets into the secure key storage<br><br>Both users (using the TOE's settings UI) and administrators (using the TOE's MDM APIs) can import secret keys into the secure key storage. | M | | I | |
| 10. destroy imported keys/secrets and [***no other keys/secrets***] in the secure key storage<br>Both users and administrators (using the TOE's MDM APIs) can destroy secret keys in the secure key storage. | M | | I | |
| 11. import X.509v3 certificates into the Trust Anchor Database<br><br>Both users (using the TOE's settings UI) and administrators (using the TOE's MDM APIs) can import X.509v3 certificates into the Trust Anchor Database. | M | | M | |
| 12. remove imported X.509v3 certificates and [***no other certificates***] in the Trust Anchor Database<br><br>Both users (using the TOE's settings UI) and administrators (using the TOE's MDM APIs) can remove imported X.509v3 certificates from the Trust Anchor Database as well as disable any of the TOE's default Root CA certificates (in the latter case, the CA certificate still resides in the | M | | I | |

| | | | | |
|---|---|---|---|---|
| TOE's read-only system partition; however, the TOE will treat that Root CA certificate and any certificate chaining to it as untrusted). | | | | |
| 13. enroll the TOE in management<br><br>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 | I | | |
| 14. remove applications<br><br>Both users (using the TOE's settings UI) and administrators (using the TOE's MDM APIs) can uninstall user and administrator installed applications on the TOE. | M | | M | |
| 15. update system software<br><br>Users can check for updates and cause the device to update if an update is available. An administrator can use MDM APIs to query the version of the TOE and query the installed applications and an MDM agent on the TOE could issue pop-ups, initiate updates, block communication, etc. until any necessary updates are completed. | M | | M | |
| 16. install applications<br><br>Both users and administrators (using the TOE's MDM APIs) can install applications on the TOE. | M | | M | |
| 17. remove Enterprise applications<br><br>An administrator (using the TOE's MDM APIs) can uninstall Enterprise installed applications on the TOE. | M | | M | |
| 18. enable/disable display notification in the locked state of: [<br>**f.** ***all notifications***]<br><br>Notifications can be configured to display in the following formats:<br>Users & administrators: show all notification content<br>Users: hide sensitive content<br>Users & administrators: hide notifications entirely<br><br>If the administrator sets any of the above settings, the user cannot change it. | M | | I | I |
| 19. enable data-at rest protection<br><br>The TOE always encrypts its user data storage. | M | | | |
| 20. enable removable media's data-at-rest protection<br>c<br>The device does not support removable media. | | | | |
| 21. enable/disable location services:<br>a. across device<br>[**d.** ***no other method***]<br><br>The administrator (using the TOE's MDM APIs) can enable or disable location services.<br><br>An additional MDM API can prohibit TOE users ability to enable and disable location services. | M | | I | I |
| 22. Enable/disable the use of [Biometric Authentication Factor] | | | | |
| 23. configure whether to allow/disallow establishment of a trusted channel if the peer/server certificate is deemed invalid. | | | | |
| 24. enable/disable all data signaling over [USB] | | | | |
| 25. enable/disable [**Wi-Fi hotspot, USB tethering, and Bluetooth tethering**] | I | | I | I |

| | | | | |
|---|---|---|---|---|
| The administrator (using the TOE's MDM APIs) can enable/disable all tethering methods (i.e. all or none disabled).<br><br>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. | | | | |
| 26. enable/disable developer modes<br><br>The administrator (using the TOE's MDM APIs) can disable Developer Mode.<br><br>Unless disabled by the administrator, TOE users can enable and disable Developer Mode. | I | | I | I |
| 27. enable/disable bypass of local user authentication<br><br>N/A – It is not possible to bypass local user auth for this TOE | | | | |
| 28. wipe Enterprise data<br><br>An administrator can remove Enterprise applications and their data. | I | | I | |
| 29. approve [***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 | I | | I | |
| 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 | I | | I | |
| 37. configure the auditable items | | | | |
| 38. retrieve TSF-software integrity verification values | | | | |
| 39. enable/disable [<br>a. ***USB mass storage mode,***] | I | | I | |
| 40. enable/disable backup to [***all applications***] *to* [***remote system, locally connected system***] | I | | I | I |
| 41. enable/disable [<br>a. ***Hotspot functionality authenticated by [pre-shared key],***<br>b. ***USB tethering authenticated by [no authentication]***]<br><br>The administrator (using the TOE's MDM APIs) can disable the Wi-Fi hotspot and USB tethering.<br><br>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 [***groups of application***] | I | | I | I |
| 43. place applications into application process groups based on [assignment: enterprise configuration settings] | | | | |
| 44. Unenroll the TOE from management | I | | I | I |
| 45. Enable/disable the Always On VPN protection | I | | I | I |
| 46. Revoke Biometric template | | | | |
| 47. [assignment: list of other management functions to be provided by the TSF] | | | | |

### 5.1.5.3 Specification of Management Functions (BT10:FMT_SMF_EXT.1/BT)

**BT10:FMT_SMF_EXT.1.1/BT**

The TSF shall be capable of performing the following Bluetooth management functions:

**Table 4 Bluetooth Security Management Functions**

| Management Function | Function | Available to User role | Available to Admin | Restricted to Admin |
|---|---|---|---|---|
| 48. Configure the Bluetooth trusted channel. - Disable/enable the Discoverable (for BR/EDR) and Advertising (for LE) modes; | M | | I | I |
| 49. Change the Bluetooth device name (separately for BR/EDR and LE). | | | | |
| 50. Provide separate controls for turning the BR/EDR and LE radios on and off; | | | | |
| 51. Allow/disallow the following additional wireless technologies to be used with Bluetooth: [***selection: Wi-Fi, NFC, [assignment: other wireless technologies]***]; | | | | |
| 52. Configure allowable methods of Out of Band pairing (for BR/EDR and LE); | | | | |
| 53. Disable/enable the Discoverable (for BR/EDR) and Advertising (for LE) modes separately; | | | | |
| 54. Disable/enable the Connectable mode (for BR/EDR and LE); | | | | |
| 55. Disable/enable the Bluetooth [**assignment: list of Bluetooth service and/or profiles available on the OS (for BR/EDR and LE)**]; | | | | |
| 56. Specify minimum level of security for each pairing (for BR/EDR and LE); | | | | |

### 5.1.5.4 Specification of Management Functions (Wireless LAN) (WLANCEP10:FMT_SMF_EXT.1/WLAN)

**WLANCEP10:FMT_SMF_EXT.1.1/WLAN**

The TSF shall be capable of performing the following management functions:

**Table 5 WLAN Security Management Functions**

| Management Function | Function | Available to User role | Available to Admin | Restricted to Admin |
|---|---|---|---|---|
| 57. configure security policy for each wireless network:<br>a. [***specify the CA(s) from which the TSF will accept WLAN authentication server certificate(s)***]<br>b. security type<br>c. authentication protocol<br>d. client credentials to be used for authentication | M | | I | |

| | | | | |
|---|---|---|---|---|
| 58. specify wireless networks (SSIDs) to which the TSF may connect;<br><br>An administrator can specify a list of wireless networks to which the TOE may connect and can restrict the TOE to only allow a connection to the specified networks. | M | | M | |
| 59. enable/disable certificate revocation list checking; | | | | |
| 60. disable ad hoc wireless client-to-client connection capability, | | | | |
| 61. disable wireless network bridging capability (for example, bridging a connection between the WLAN and cellular radios on a smartphone so it can function as a hotspot); | | | | |
| 62. disable roaming capability; | | | | |
| 63. enable/disable IEEE 802.1X pre-authentication; | | | | |
| 64. enable/disable and configure PMK caching:<br>a. set the amount of time (in minutes) PMK entries are cached;<br>b. set the maximum number of PMK entries that can be cached. | | | | |

(TD0470 applied)

#### 5.1.5.5 Specification of Remediation Actions (MDFPP32:FMT_SMF_EXT.2)

**MDFPP32:FMT_SMF_EXT.2.1**

The TSF shall offer [***wipe of protected data, wipe of sensitive data, remove Enterprise applications, remove all device-stored Enterprise resource data***] upon un-enrollment and [***no other triggers***].

#### 5.1.5.6 Current Administrator (MDFPP32:FMT_SMF_EXT.3)

**MDFPP32:FMT_SMF_EXT.3.1**

The TSF shall provide a mechanism that allows users to view a list of currently authorized administrators and the management functions that each administrator is authorized to perform.

### 5.1.6 Protection of the TSF (FPT)

#### 5.1.6.1 Application Address Space Layout Randomization (MDFPP32:FPT_AEX_EXT.1)

**MDFPP32:FPT_AEX_EXT.1.1**

The TSF shall provide address space layout randomization ASLR to applications.

**MDFPP32:FPT_AEX_EXT.1.2**

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

#### 5.1.6.2 Memory Page Permissions (MDFPP32:FPT_AEX_EXT.2)

**MDFPP32:FPT_AEX_EXT.2.1**

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

#### 5.1.6.3 Stack Overflow Protection (MDFPP32:FPT_AEX_EXT.3)

**MDFPP32: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.

#### 5.1.6.4 Domain Isolation (MDFPP32:FPT_AEX_EXT.4)

**MDFPP32:FPT_AEX_EXT.4.1**

The TSF shall protect itself from modification by untrusted subjects.

**MDFPP32:FPT_AEX_EXT.4.2**

The TSF shall enforce isolation of address space between applications.

#### 5.1.6.5 Kernel Address Space Layout Randomization (MDFPP32:FPT_AEX_EXT.5)

**MDFPP32:FPT_AEX_EXT.5.1**
The TSF shall provide address space layout randomization (ASLR) to the kernel.

**MDFPP32:FPT_AEX_EXT.5.2**
The base address of any kernel-space memory mapping will consist of [**4**] unpredictable bits.

#### 5.1.6.6 Application Processor Mediation (MDFPP32:FPT_BBD_EXT.1)

**MDFPP32: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.

#### 5.1.6.7 JTAG Disablement (MDFPP32:FPT_JTA_EXT.1)

**MDFPP32:FPT_JTA_EXT.1.1**
The TSF shall [***control access by a signing key***] to JTAG.

#### 5.1.6.8 Key Storage (MDFPP32:FPT_KST_EXT.1)

**MDFPP32:FPT_KST_EXT.1.1**
The TSF shall not store any plaintext key material in readable non-volatile memory.

#### 5.1.6.9 No Key Transmission (MDFPP32:FPT_KST_EXT.2)

**MDFPP32:FPT_KST_EXT.2.1**
The TSF shall not transmit any plaintext key material outside the security boundary of the TOE.

#### 5.1.6.10 No Plaintext Key Export (MDFPP32:FPT_KST_EXT.3)

**MDFPP32:FPT_KST_EXT.3.1**
The TSF shall ensure it is not possible for the TOE user(s) to export plaintext keys.

#### 5.1.6.11 Self-Test Notification (MDFPP32:FPT_NOT_EXT.1)

**MDFPP32: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***]

#### 5.1.6.12 Reliable time stamps (MDFPP32:FPT_STM.1)

**MDFPP32:FPT_STM.1.1**
The TSF shall be able to provide reliable time stamps for its own use.

#### 5.1.6.13 TSF Cryptographic Functionality Testing (MDFPP32:FPT_TST_EXT.1)

**MDFPP32: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.

#### 5.1.6.14 TSF Cryptographic Functionality Testing (Wireless LAN) (WLANCEP10:FPT_TST_EXT.1/WLAN)

**WLANCEP10:FPT_TST_EXT.1.1/WLAN**
The [***TOE platform***] shall run a suite of self-tests during initial start-up (on power on) to demonstrate the correct operation of the TSF.

**WLANCEP10:FPT_TST_EXT.1.2/WLAN**
The [***TOE platform***] shall provide the capability to verify the integrity of stored TSF executable code when it is loaded for execution through the use of the TSF-provided cryptographic services.

#### 5.1.6.15 TSF Integrity Checking (Post-Kernel) (MDFPP32:FPT_TST_EXT.2/POSTKERNEL)

**MDFPP32:FPT_TST_EXT.2.1/POSTKERNEL**
The TSF shall verify the integrity of [***[the /system and /vendor partition]***] stored in mutable media prior to its execution through the use of [***hardware-protected hash***].

#### 5.1.6.16 TSF Integrity Checking (Pre-Kernel) (MDFPP32:FPT_TST_EXT.2/PREKERNEL)

**MDFPP32:FPT_TST_EXT.2.1/PREKERNEL**
The TSF shall verify the integrity of the bootchain up through the Application Processor OS kernel stored in mutable media prior to its execution through the use of [***an immutable hardware hash of an asymmetric key***].

#### 5.1.6.17 Trusted Update: TSF Version Query (MDFPP32:FPT_TUD_EXT.1)

**MDFPP32:FPT_TUD_EXT.1.1**
The TSF shall provide authorized users the ability to query the current version of the TOE firmware/software.

**MDFPP32: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.

**MDFPP32:FPT_TUD_EXT.1.3**
The TSF shall provide authorized users the ability to query the current version of installed mobile applications.

#### 5.1.6.18 TSF Update Verification (MDFPP32:FPT_TUD_EXT.2)

**MDFPP32:FPT_TUD_EXT.2.1**
The TSF shall verify software updates to the Application Processor system software and [***[baseband processor]***] using a digital signature verified by the manufacturer trusted key prior to installing those updates.

**MDFPP32:FPT_TUD_EXT.2.2**
The TSF shall [***update only by verified software***] the TSF boot integrity [***key***].

**MDFPP32:FPT_TUD_EXT.2.3**
The TSF shall verify that the digital signature verification key used for TSF updates [***matches an immutable hardware public key***].

#### 5.1.6.19 Application Signing (MDFPP32:FPT_TUD_EXT.3)

**MDFPP32:FPT_TUD_EXT.3.1**
The TSF shall verify mobile application software using a digital signature mechanism prior to installation.

## 5.1.7 TOE access (FTA)

### 5.1.7.1 TSF- and User-initiated Locked State (MDFPP32:FTA_SSL_EXT.1)

**MDFPP32:FTA_SSL_EXT.1.1**
The TSF shall transition to a locked state after a time interval of inactivity.

**MDFPP32:FTA_SSL_EXT.1.2**
The TSF shall transition to a locked state after initiation by either the user or the administrator.

**MDFPP32: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**].

### 5.1.7.2 Default TOE Access Banners (MDFPP32:FTA_TAB.1)

**MDFPP32:FTA_TAB.1.1**
Before establishing a user session, the TSF shall display an advisory warning message regarding unauthorized use of the TOE.

### 5.1.7.3 Wireless Network Access (WLANCEP10:FTA_WSE_EXT.1)

**WLANCEP10:FTA_WSE_EXT.1.1**
The TSF shall be able to attempt connections only to wireless networks specified as acceptable networks as configured by the administrator in FMT_SMF_EXT.1.1/WLAN.

## 5.1.8 Trusted path/channels (FTP)

### 5.1.8.1 Bluetooth Encryption (BT10:FTP_BLT_EXT.1)

**BT10:FTP_BLT_EXT.1.1**
The TSF shall enforce the use of encryption when transmitting data over the Bluetooth trusted channel for BR/EDR and [***LE***].

**BT10:FTP_BLT_EXT.1.2**
The TSF shall use key pairs per FCS_CKM_EXT.8 for Bluetooth encryption.

### 5.1.8.2 Persistence of Bluetooth Encryption (BT10:FTP_BLT_EXT.2)

**BT10:FTP_BLT_EXT.2.1**
The TSF shall [***terminate the connection***] if the remote device stops encryption while connected to the TOE.

### 5.1.8.3 Bluetooth Encryption Parameters (BR/EDR) (BT10:FTP_BLT_EXT.3/BR)

**BT10:FTP_BLT_EXT.3.1/BR**
The TSF shall set the minimum encryption key size to [**128 bits**] for BR/EDR and not negotiate encryption key sizes smaller than the minimum size.

### 5.1.8.4 Bluetooth Encryption Parameters (LE) (BT10:FTP_BLT_EXT.3/LE)

**BT10:FTP_BLT_EXT.3.1/LE**
The TSF shall set the minimum encryption key size to [128 bits] for LE and not negotiate encryption key sizes smaller than the minimum size

#### 5.1.8.5 Trusted Channel Communication (MDFPP32:FTP_ITC_EXT.1)

**MDFPP32:FTP_ITC_EXT.1.1**

The TSF shall use
- 802.11-2012 in accordance with the Extended Package for WLAN Clients,
- 802.1X in accordance with the Extended Package for WLAN Clients,
- EAP-TLS in accordance with the Extended Package for WLAN Clients,
- mutually authenticated TLS as defined in the Package for Transport Layer Security

and

[***HTTPS***]

protocols 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.

**MDFPP32:FTP_ITC_EXT.1.2**

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

**MDFPP32: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***].

#### 5.1.8.6 Trusted Channel Communication (Wireless LAN) (WLANCEP10:FTP_ITC_EXT.1/WLAN)

**WLANCEP10:FTP_ITC_EXT.1.1/WLAN**

The TSF shall use 802.11-2012, 802.1X, and EAP-TLS to provide a trusted communication channel between itself and a wireless access point 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.

**WLANCEP10:FTP_ITC_EXT.1.2/WLAN**

The TSF shall initiate communication via the trusted channel for wireless access point connections.

## 5.2 TOE Security Assurance Requirements

The SARs for the TOE are the 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: Labeling 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 6 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 describe the purpose and method of use for each SFR-enforcing and SFR-supporting TSFI.

**ADV_FSP.1.4c**
The functional specification shall identify all parameters associated with each SFR-enforcing and SFR-supporting TSFI.

**ADV_FSP.1.5c**
The functional specification shall provide rationale for the implicit categorization of interfaces as SFR-non-interfering.

**ADV_FSP.1.6c**
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 useraccessible 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, the useraccessible functions and privileges that should be controlled in a secure processing environment, including appropriate warnings.

**AGD_OPE.1.3c**
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.1.4c**
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.1.5c**
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.1.6c**
The operational user guidance shall identify all possible modes of operation of the OS (including operation following failure or operational error), their consequences, and implications for maintaining secure operation.

**AGD_OPE.1.7c**
The operational user guidance shall, for each user role, describe the security measures to be followed in order to fulfill the security objectives for the operational environment as described in the ST.

**AGD_OPE.1.8c**
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 acceptance of the delivered TOE in accordance with the developer's delivery procedures.

**AGD_PRE.1.3c**
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 Labeling 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.1.2c**
The TOE shall be labeled with a unique reference.

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

#### 5.2.3.2 TOE CM Coverage (ALC_CMS.1)

**ALC_CMS.1.1d**
The developer shall provide a configuration list for the TOE.

**ALC_CMS.1.1c**
The configuration list shall include the following: the TOE itself; and the evaluation evidence required by the SARs.

**ALC_CMS.1.2c**
The configuration list shall include the following: the TOE itself; and the evaluation evidence required by the SARs.

**ALC_CMS.1.3c**
The configuration list shall uniquely identify the configuration items.

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

#### 5.2.3.3 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.

### 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.1.2c**
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.

**ATE_IND.1.3e**
The evaluator shall confirm that the information provided meets all requirements for content and presentation of evidence.

**ATE_IND.1.4e**
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.1.2c**
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 confirm that the information provided meets all requirements for content and presentation of evidence.

**AVA_VAN.1.4e**

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

**AVA_VAN.1.5e**

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:

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

## 6.1 Security audit

**MDFPP32/BT10/WLANEP10:FAU_GEN.1:**

The TOE uses different forms of logs to meet all the required management logging events specified in Table 1 and Table 2 of the MDFPP:

1. Security Logs
2. Logcat Logs

Each of the above logging methods are described below.

- *Security Logs:* A full list of all auditable events (for MDFPP31/WLANCEP10) can be found here: https://developer.android.com/reference/android/app/admin/SecurityLog#constants_1. Values found in this list represent Security Log keywords used in this logging function along with a description of what the log means and any additional information/variables present in the audit record. Additionally, the following link provides the additional information that can be grabbed when an MDM requests a copy of the logs: https://developer.android.com/reference/android/app/admin/SecurityLog.SecurityEvent. Each log contains a keyword or phrase describing the event, the date and time of the event, and further event-specific values that provide success, failure, and other information relevant to the event.
- *Logcat Logs:* Similar to Security Logs, Logcat Logs contain date, time, and further even-specific values within the logs. In addition, Logcat Logs provide a value that maps to a user ID to identify which user caused the event that generated the log. Finally, Logcat Logs are descriptive and do not require the administrator to know the template of the log to understand its values. Logcat Logs cannot be exported but can be viewed by an administrator via an MDM agent.

Both types of logs, when full, wrap around and overwrite the oldest log (as the start of the buffer).

One can find the TOE audit records for each protection profile as follows in Table 2 Audit Events.

Some audit records, while logged, are unavailable to the administrator due to a number of reasons. Such audits and their explanations are identified below:

- MDFPP32:FAU_GEN.1 – Shutdown of the audit functions: Upon log shutdown, the security log buffer is deallocated and no longer available to be read, rendering the viewing of such an audit unavailable for the administrator to view.
- MDFPP32:FAU_GEN.1 – Shutdown of the Rich OS: Since security logs are stored in memory, a shutdown of the system clears the audit record that is generated stating that the system is shutting down.
- MDFPP32:FPT_TST_EXT.1 - Failure of self-test: Self-tests take place prior to the initialization of audit records. While the self-test success/failure audit is queued up to be logged upon security logs being

initialized, when a self-test failure occurs the boot process is halted prior to security logs being initialized.

**MDFPP32:FAU_SAR.1:**

The TOE provides an MDM API to allow a Device-Owner MDM agent to read the security logs.

**MDFPP32:FAU_STG.1:**

For security logs, the TOE stores all audit records in memory, making it only accessible to the logd daemon, and only device owner applications can call the MDM API to retrieve a copy of the logs. Additionally, only new logs can be added. There is no designated method allowing for the deletion or modification of logs already present in memory, but reading the security logs clears the buffer at the time of the read.

The TOE stores Logcat Logs in memory and only allows access by an administrator via an MDM Agent. The TOE prevents deletion of these logs by any method other than USB debugging (and enabling USB Debugging takes the phone out of the evaluated configuration).

**MDFPP32:FAU_STG.4:**

The security logs and logcat logs are stored in memory in a circular log buffer of 10KB/64KB, respectively. Logcat logs alone have a configurable size, able to be set by an MDM API. There is no limit to the size that the Logcat log buffer can be configured to and it is limited to the size of the system's memory. Once the log is full, it begins overwriting the oldest message and continues overwriting the oldest message with each new auditable event. These logs persist until they are either overwritten or the device is restarted.

## 6.2 Cryptographic support

**MDFPP32:FCS_CKM.1:**

The TOE provides generation of asymmetric keys including

| Algorithm | Key/Curve Sizes | Usage |
|---|---|---|
| RSA, FIPS 186-4 | 2048/3072 | API/Application & Sensitive Data Protection (DAR.2) |
| ECDSA, FIPS 186-4 | P-256/384/521 | API/Application |
| ECDHE keys (not domain parameters) | P-256/384 | TLS KeyEx (WPA2 w/ EAP-TLS & HTTPS) |

**Table 7 Asymmetric Key Generation**

The TOE's cryptographic algorithm implementations have received NIST algorithm certificates (please see tables in FCS_COP.1 for all of the TOE'S algorithm certificates). The TOE itself does not generate any RSA/ECDSA authentication key pairs for TOE functionality (the user or administrator must load certificates for use with WPA2 with EAP-TLS authentication); however, the TOE provides key generation APIs to mobile applications to allow them to generate RSA/ECDSA key pairs. The TOE generates only ECDH key pairs (as BoringSSL does not support DH/DHE cipher suites) and does not generate domain parameters (curves) for use in TLS Key Exchange.

The TOE will provide a library for application developers to use for Sensitive Data Protection (SDP). This library (class) generates asymmetric RSA keys for use to encrypt and decrypt data that comes to the device while in a locked state. Any data received for a specified application (that opts into SDP via this library), is encrypted using the public key and stored until the device is unlocked. The public key stays in memory no matter the state of the device (locked or unlocked). However, when the device is locked, the private key is evicted from memory and unavailable for use until the device is unlocked. Upon unlock, the private key is re-derived and used to decrypt data received and encrypted while locked.

**WLANCEP10:FCS_CKM.1/WLAN:**

The TOE adheres to IEEE 802.11-2012 for key generation. The TOE's wpa_supplicant provides PRF384 for WPA2 derivation of 128-bit AES Temporal Key (using the HMAC implementation provided by BoringSSL) and employs

its BoringSSL AES-256 DRBG when generating random values used in the EAP-TLS and 802.11 4-way handshake. The TOE supports the AES-128 CCMP encryption mode. The TOE has successfully completed certification (including WPA2 Enterprise) and received Wi-Fi CERTIFIED Interoperability Certificates from the Wi-Fi Alliance. The Wi-Fi Alliance maintains a website providing further information about the testing program: http://www.wi-fi.org/certification.

| Device Name | Model Number | Wi-Fi Alliance Certificate Numbers |
|---|---|---|
| 6375 Mobile Handhelds | ET40 | WFA120159 |
| 6375 Mobile Handhelds | ET45 | WFA119406 |

**Table 8 Device WFA Certificates**

**MDFPP32:FCS_CKM.2/LOCKED:**

The TOE performs key establishment as part of EAP-TLS and TLS session establishment. **Table 7 Asymmetric Key Generation** enumerates the TOE'S supported key establishment implementations (RSA/ECDH for TLS/EAP-TLS).

**MDFPP32:FCS_CKM.2/UNLOCKED:**

The TOE provides an SDP library for applications that uses a hybrid crypto scheme based on 3072-bit RSA based key establishment. Applications can utilize this library to implement SDP that encrypts incoming data received while the phone is locked in a manner compliant with this requirement.

**WLANCEP10:FCS_CKM.2/WLAN:**

The TOE adheres to RFC 3394 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).

**MDFPP32:FCS_CKM_EXT.1:**

The TOE includes a Root Encryption Key (REK) stored in a 256-bit fuse bank within the application processor. The TOE generates the REK/fuse value during manufacturing using its hardware DRBG. The application processor protects the REK by preventing any direct observation of the value and prohibiting any ability to modify or update the value. The application processor loads the fuse value into an internal hardware crypto register and the Trusted Execution Environment (TEE) provides trusted applications the ability to derive KEKs from the REK (using an SP 800-108 KDF to combine the REK with a salt). Additionally, the when the REK is loaded, the fuses for the REK become locked, preventing any further changing or loading of the REK value. The TEE does not allow trusted applications to use the REK for encryption or decryption, only the ability to derive a KEK from the REK. The TOE includes a TEE application that calls into the TEE in order to derive a KEK from the 256-bit REK/fuse value and then only permits use of the derived KEK for encryption and decryption as part of the TOE key hierarchy. More information regarding Trusted Execution Environments may be found here: http://www.globalplatform.org/mediaguidetee.asp.

**MDFPP32:FCS_CKM_EXT.2:**

The TOE utilizes its approved RBGs to generate DEKs. When generating AES keys for itself (for example, the TOE'S sensitive data encryption keys or for the Secure Key Storage), the TOE utilizes the RAND_bytes() API call from its BoringSSL AES-256 CTR_DRBG to generate a 256-bit AES key. The TOE also utilizes that same DRBG when servicing API requests from mobile applications wishing to generate AES keys (either 128 or 256-bit).

In all cases, the TOE generates DEKs using a compliant RBG seeded with sufficient entropy so as to ensure that the generated key cannot be recovered with less work than a full exhaustive search of the key space.

**MDFPP32:FCS_CKM_EXT.3:**

The TOE takes the user-entered password and conditions/stretches this value before combining the factor with other KEK.

The TOE generates all non-derived KEKs using the RAND_bytes() API call from its BoringSSL AES-256 CTR_DRBG to ensure a full 128/256-bits of strength for asymmetric/symmetric keys, respectively. And the TOE combines KEKs by encrypting one KEK with the other so as to preserve entropy.

**MDFPP32:FCS_CKM_EXT.4:**

The TOE clears sensitive cryptographic material (plaintext keys, authentication data, other security parameters) from memory when no longer needed or when transitioning to the device's locked state (in the case of the Sensitive Data Protection keys). Public keys (such as the one used for Sensitive Data Protection) can remain in memory when the phone is locked, but all crypto-related private keys are evicted from memory upon device lock. No plaintext cryptographic material resides in the TOE'S Flash as the TOE encrypts all keys stored in Flash. When performing a full wipe of protected data, the TOE cryptographically erases the protected data by clearing the Data-At-Rest DEK. Because the TOE'S keystore resides within the user data partition, the TOE effectively cryptographically erases those keys when clearing the Data-At-Rest DEK. In turn, the TOE clears the Data-At-Rest DEK and Secure Key Storage SEK through a secure direct overwrite (BLKSECDISCARD ioctl) of the wear-leveled Flash memory containing the key followed by a read-verify.

**MDFPP32:FCS_CKM_EXT.5:**

The TOE stores all protected data in encrypted form within the user data partition (either protected data or sensitive data). Upon request, the TOE cryptographically erases the Data-At-Rest DEK protecting the user data partition and the SDP Master KEK protecting sensitive data files in the user data partition, clears those keys from memory, reformats the partition, and then reboots. The TOE's clearing of the keys follows the requirements of FCS_CKM_EXT.4.

**MDFPP32:FCS_CKM_EXT.6:**

The TOE generates salt nonces (which are just salt values used in WPA2) using its /dev/urandom.

| Salt value and size | RBG origin | Salt storage location |
|---|---|---|
| User password salt (128-bit) | BoringSSL's AES-256 CTR_DRBG | Flash filesystem |
| TLS client_random (256-bit) | BoringSSL's AES-256 CTR_DRBG | N/A (ephemeral) |
| TLS pre_master_secret (384-bit) | BoringSSL's AES-256 CTR_DRBG | N/A (ephemeral) |
| TLS ECDHE private value (256, 384) | BoringSSL's AES-256 CTR_DRBG | N/A (ephemeral) |
| WPA2 4-way handshake supplicant nonce (SNonce) | BoringSSL's AES-256 CTR_DRBG | N/A (ephemeral) |

**BT10:FCS_CKM_EXT.8**

The TOE generates public/private ECDH key pairs every blue connection establishment.

**MDFPP32:FCS_COP.1:**

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

The TOE's BoringSSL Library (version 7f02881e96e51f1873afcf384d02f782b48967ca, with both Processor Algorithm Accelerators (PAA) and without PAA) provides the following algorithms:

| SFR | Algorithm | NIST Standard | Cert# |
|---|---|---|---|
| FCS_CKM.1 (Key Gen) | RSA IFC Key Generation – 2048/3072 bits | FIPS 186-4, RSA | A3326 |
| | ECDSA ECC Key Generation - P-256/384/521 | FIPS 186-4, ECDSA | A3326 |
| FCS_CKM.2 (Key Establishment) | RSA-based Key Exchange | Vendor affirm 800-56B | N/A |
| | ECC-based Key Exchange - P-256/384/521 | SP 800-56A, CVL KAS ECC | A3326 |

| SFR | Algorithm | NIST Standard | Cert# |
|---|---|---|---|
| FCS_COP.1(1) (AES) | AES - 128/256 CBC, GCM, KW | FIPS 197, SP 800-38A/D/F | A3326 |
| FCS_COP.1(2) (Hash) | SHA Hashing - 1/256/384/512 | FIPS 180-4 | A3326 |
| FCS_COP.1(3) (Sign/Verify) | RSA Sign/Verify - 2048/3072 bits | FIPS 186-4, RSA | A3326 |
| | ECDSA Sign/Verify - P-256/384/521 | FIPS 186-4, ECDSA | A3326 |
| FCS_COP.1(4) (Keyed Hash) | HMAC-SHA -1/256/384/512 | FIPS 198-1 & 180-4 | A3326 |
| FCS_RBG_EXT.1 (Random) | DRBG Bit Generation – 256 bits | SP 800-90A (Counter) | A3326 |

**Table 9 BoringSSL Cryptographic Algorithms**

Android's LockSettings service (version 77561fc30db9aedc1f50f5b07504aa65b4268b88) provides the TOE'S SP 800-108 key based key derivation function for deriving KEKs.

| SFR | Algorithm | NIST Standard | Cert# |
|---|---|---|---|
| FCS_CKM_EXT.3 | LockSettings service KBKDF | SP 800-108 | A1978 |

**Table 10 LockSettings Service KDF Cryptographic Algorithms**

The TOE's Wi-Fi chipset (BCM43752) provides an AES-CCMP implementation, and the TOE's application processor (Snapdragon 695 [SM6375]) provides additional cryptographic algorithms.

| SFR | Algorithm | NIST Standard | Cert# |
|---|---|---|---|
| FCS_COP.1(1) (AES) (Wi-Fi) | AES 128/256 CCM | FIPS 197, SP 800-38C | 4791 |
| FCS_COP.1(1) (AES) (QTI CEC*) | AES 128/256 CBC | FIPS 197, SP 800-38A | A805 |
| FCS_COP.1(1) (AES) (QTI UFS**) | AES 128/256 XTS | FIPS 197, SP 800-38E | A771 A772 |
| FCS_COP.1(2) (Hash) (QTI CEC) | SHA 1/256 Hashing | FIPS 180-4 | A805 |
| FCS_COP.1(2) (Hash) (DRBG) | SHA 256 Hashing | FIPS 180-4 | A1630 |
| FCS_COP.1(4) (Keyed Hash) (QTI CEC) | HMAC-SHA-1/256 | FIPS 198-1 & 180-4 | A805 |
| FCS_RBG_EXT.1 (Random) (DRBG) | DRBG Bit Generation | SP 800-90A (Hash-256) | A1630 |

*QTI CEC – Qualcomm Technologies, Inc. Crypto Engine Core v5.6.0
**QTI UFS - Qualcomm Technologies, Inc. Inline Crypto Engine (UFS) v3.2.0

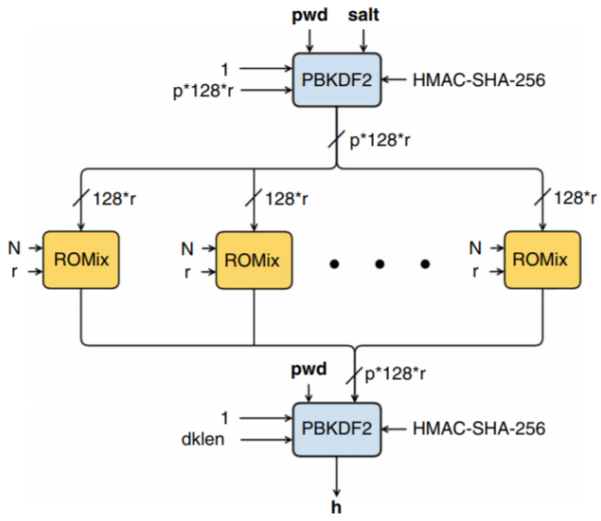
**Table 11 SM6365 Hardware Cryptographic Algorithms**

**MDFPP32:FCS_COP.1/CONDITION:**

The TOE stretches the user's password to create a password derived key. The TOE stretching function uses a series of steps to increase the memory required for key derivation (thus thwarting GPU-acceleration, off-line brute force, and precomputed dictionary attacks) and ensure proper conditioning and stretching of the user's password.

The TOE conditions the user's password using two iterations of PBKDFv2 w HMAC-SHA-256 in addition to some ROMix operations in an algorithm named scrypt. Scrypt consists of one iteration of PBKDFv2, followed by a series of ROMix operations, and finished with a final iteration of PBKDFv2. The ROMix operations increase the memory required for key derivation, thus thwarting GPU-acceleration (which can greatly decrease the time needed to brute force PBKDFv2 alone). The time needed to derive keying material does not impact or lessen the difficulty faced by an attacker's exhaustive guessing as the combination of the password derived KEK with REK value entirely prevents offline attacks and the TOE's maximum incorrect login attempts.

The following scrypt diagram shows how the password and salt are used with PBKDFv2 and ROMix to fulfil the requirements for password conditioning.




The resulting derived key from this operation is combined with keys chaining to the Application Processor REK and then used to decrypt the FBE DEKs and also to derive the User Keystore Daemon Value.

**MDFPP32:FCS_COP.1/ENCRYPT:**

The TOE has received an ACVP certificate for its encryption/decryption routines as described in the tables above.

**MDFPP32:FCS_COP.1/HASH:**

The TOE uses byte-wise hashing operations as part of signatures as well as part of HMAC (keyed hashing) operations.

**MDFPP32:FCS_COP.1/KEYHMAC:**

The TOE uses HMAC as part of the TLS ciphersuites and makes HMAC functionality available to mobile applications. For TLS, the TOE uses HMAC using SHA-1 (with a 160-bit key) to generate a 160-bit MAC, SHA-256 (with a 256-bit key) to generate a 256-bit MAC, SHA-384 (with a 384-bit key) to generate a 384-bit MAC. For mobile applications, the TOE provides all of the previous HMACs as well as SHA-512 (with a 512-bit key) to generate a 512-bit MAC. FIPS 198-1 & 180-4 dictate the block size used, and they specify block sizes/output MAC lengths of 512/160, 512/160, 1024/384, and 1024/512-bits for HMAC-SHA-1, HMAC-SHA-256, HMAC-SHA-384, and HMAC-SHA-512 respectively.

**MDFPP32:FCS_COP.1/KEYHMAC.1:**

**MDFPP32:FCS_COP.1/SIGN:**

The TOE has received an CAVP certificate for its encryption/decryption routines as described in the tables above.

**MDFPP32:FCS_HTTPS_EXT.1:**

The TOE supports the HTTPS protocol (compliant with RFC 2818) so that (mobile and system) applications executing on the TOE can act as HTTPS clients and 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.

**MDFPP32:FCS_IV_EXT.1:** (see KMD for more information)

The TOE generates IVs by reading from /dev/urandom for use with all keys. In all cases, the TOE uses /dev/urandom and generates the IVs in compliance with the requirements of table 13 in Appendix F of MDFPP32.

**MDFPP32: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 Application Processor.

2. An AES-256 CTR_DRBG provided by BoringSSL. This is the only accredited and supported DRBG present in the system and available to independently developed applications. As such, the TOE provides mobile applications access (through an Android Java API) to random data drawn from its AES-256 CTR_DRBG.

The TOE initializes its AP Hash_DRBG with enough data from its hardware noise source to ensure at least 256-bits of entropy. The TOE then uses its AP Hash_DRBG to continuously fill the Linux Kernel Random Number Generator (LKRNG) input pool, and the LKRNG makes entropy easily available to the rest of the system (e.g., the BoringSSL DRBG draws from the LKRNG).

The TOE seeds its BoringSSL AES-256 CTR_DRBG using 384-bits of data from /dev/random, thus ensuring at least 256-bits of entropy. The TOE uses its BoringSSL DRBG for all random generation including salts.

**MDFPP32: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), keyed message digests (HMAC-SHA-256), generation of asymmetric keys for key establishment (RSA and ECDH), and generation of asymmetric keys for signature generation and verification (RSA, ECDSA). The TOE provides access through the Android operating system's Java API, through the native BoringSSL API, and through the application processor module (user and kernel) APIs.

**MDFPP32:FCS_SRV_EXT.2:**

The TOE provides applications with APIs to perform the functions referenced in FCS_COP.1(1) and FCS_COP.1(3).

**MDFPP32:FCS_STG_EXT.1:**

The TOE provides the user, administrator, and mobile applications the ability to import and use asymmetric public and private keys into the TOE'S software-based Secure Key Storage. Certificates are stored in files using UID-based permissions and an API virtualizes the access. Additionally, the user and administrator can request the TOE to destroy the keys stored in the Secure Key Storage. While normally mobile applications cannot use or destroy the keys of another application, applications that share a common application developer (and are thus signed by the same developer key) may do so. In other words, applications with a common developer (and which explicitly declare a shared UUID in their application manifest) may use and destroy each other's keys located within the Secure Key Storage.

The TOE also provides additional protections on keys beyond including key attestation, to allow enterprises and application developers the ability to ensure which keys have been generated securely within the phone.

**MDFPP32:FCS_STG_EXT.2:** (see KMD for more information)

The TOE employs a key hierarchy that protects all DEKs and KEKs by encryption with either the REK or by the REK and password derived KEK.

The TOE encrypts Long-term Trusted channel Key Material (LTTCKM, i.e., Bluetooth and WiFi keys) values using AES-256 GCM encryption and stores the encrypted values within their respective configuration files.

All keys are 256-bits in size. All keys are generated using the TOE'S BoringSSL AES-256 CTR_DRBG or application processor SHA-256 Hash_DRBG. By utilizing only 256-bit KEKs, the TOE ensures that all keys are encrypted by an equal or larger sized key.

In the case of Wi-Fi, the TOE utilizes the 802.11-2012 KCK and KEK keys to unwrap (decrypt) the WPA2 Group Temporal Key received from the access point. The TOE protects persistent Wi-Fi keys (user certificates and private keys) by storing them in the Android Key Store.

**MDFPP32:FCS_STG_EXT.3:**

The TOE protects the integrity of all DEKs and KEKs (including LTTCKM keys) stored in Flash by using authenticated encryption/decryption methods (CCM, GCM).

**PKGTLS11:FCS_TLS_EXT.1:**

**PKGTLS11:FCS_TLSC_EXT.1/2:**

The TOE provides mobile applications (through its Android API) the use of TLS version 1.2 as a client including support for the selections in chosen in section 5 for FCS_TLSC_EXT.1 (and the TOE requires no configuration other than using the appropriate library APIs as described in the Admin Guidance).

When an application uses the combined APIs provided in the Admin Guide to attempt to establish a trusted channel connection based on TLS or HTTPS, the TOE supports only Subject Alternative Name (SAN) (DNS and IP address) as reference identifiers (the TOE does not accept reference identifiers in the Common Name[CN]). The TOE supports client (mutual) authentication. The TOE in its evaluated configuration and, by design, supports elliptic curves for TLS (P-256 and P-384) and has a fixed set of supported curves (thus the admin cannot and need not configure any curves).

No additional configuration is needed to restrict allow the device to use the supported cipher suites, as only the claimed cipher suites are supported in the aforementioned library as each of the aforementioned ciphersuites are supported on the TOE by default or through the use of the TLS library.

While the TOE supports the use of wildcards in X.509 reference identifiers (SAN only), the TOE does not support certificate pinning. If the TOE cannot determine the revocation status of a peer certificate, the TOE rejects the certificate and rejects the connection.

**WLANCEP10:FCS_TLSC_EXT.1/2/WLAN:**

The TSF supports TLS versions 1.0, 1.1, and 1.2 and also supports the selected ciphersuites utilizing SHA-1, SHA-256, and SHA-384 (see the selections in section 5 for FCS_TLSC_EXT.1/WLAN) for use with EAP-TLS as part of WPA2. The TOE in its evaluated configuration and, by design, supports only evaluated elliptic curves (P-256 & P-384 and no others) and has a fixed set of supported curves (thus the admin cannot and need not configure any curves).

The TOE, allows the user to load and utilize authentication certificates for EAP-TLS used with WPA. The Android UI (Settings->Security->Credential storage: Install from device storage) allows the user to import an RSA or ECDSA certificate and designate its use as WiFi.

**PKGTLS11:FCS_TLSC_EXT.4:**

The TOE includes the ‘renegotiation_info’ TLS extension in its TLS client hello message.

**PKGTLS11:FCS_TLSC_EXT.5:**

The TOE supports the secp256r1, and secp384r1 groups in its TLS client hello message ‘supported_groups’ extension.

## 6.3 User data protection

**MDFPP32:FDP_ACF_EXT.1:**

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

1. Normal - A lower-risk permission that gives an application access to isolated application-level features, with minimal risk to other applications, the system, or the user. The system automatically grants this type of permission 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 - A higher-risk permission that would give a requesting application access to private user data or control over the device that can negatively impact the user. Because this type of permission introduces potential risk, the system cannot automatically grant it to the requesting application. For example, any dangerous permissions requested by an application will be displayed to the user and require confirmation before proceeding or some other approach can be taken to avoid the user automatically allowing the use of such facilities.
3. Signature - A permission that the system is to grant 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. SignatureOrSystem - A permission that the system is to grant only to packages in the Android system image or that are signed with the same certificates. Please avoid using this option, as the signature protection level should be sufficient for most needs and works regardless of exactly where applications are installed. This permission is used for certain special situations where multiple vendors have applications built in to a system image which need to share specific features explicitly because they are being built together.

An example of a normal permission is the ability to check whether the device is connected to a network: android.permission.ACCESS_NETWORK_STATE. This permission allows an application to query whether the phone currently has a network connection (whether that be through Wi-Fi, USB tethering, cellular, etc.), and an application that does not request (or declare) this permission have its query rejected (and would not learn the device's networking state).

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 running 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 application is allowed to continue running. If the user disapproves, the devices continues to run, but cannot use the services protected by the denied permissions. 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 (2nd installed app) that requests this permission and that has been signed with the same developer key used to sign the application (1st installed app) declaring the permission (in the case of the example, the Android Framework itself).

An example of a signatureOrSystem permission is the android.permission.CONTROL_LOCATION_UPDATES, which allows the system to allow or disallow the cellular radio to update the device's location. 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 systemOrSignature permissions by Signature or by virtue of the requesting application being part of the "system image".

Additionally, Android includes the following flags that layer atop the base categories.

1. privileged - this permission can also be granted to any applications installed as privileged apps on the system image. Please avoid using this option, as the signature protection level should be sufficient for most needs and works regardless of exactly where applications are installed. This permission flag is used for certain special situations where multiple vendors have applications built in to a system image which need to share specific features explicitly because they are being built together.
2. system - Old synonym for 'privileged'.
3. development - this permission can also (optionally) be granted to development applications (e.g., to allow additional location reporting during beta testing).
4. appop - this permission is closely associated with an app op for controlling access.
5. pre23 - this permission can be automatically granted to apps that target API levels below API level 23 (Marshmallow/6.0).
6. installer - this permission can be automatically granted to system apps that install packages.
7. verifier - this permission can be automatically granted to system apps that verify packages.
8. preinstalled - this permission can be automatically granted to any application pre-installed on the system image (not just privileged apps) (the TOE does not prompt the user to approve the permission).

For older applications (those targeting Android's pre-23 API level, i.e., API level 22 [lollipop] and below), the TOE will prompt a user at the time of application installation whether they agree to grant the application access to the requested services. Thereafter (each time the application is run), the TOE will grant the application access to the

services specified during install.

For newer applications (those targeting API level 23 or later), the TOE grants individual permissions at application run-time by prompting the user for confirmation of each permissions category requested by the application (and only granting the permission if the user chooses to grant it).

The Android 11.0 (Level 30) API (details found here https://developer.android.com/reference/packages) provides services to mobile applications.

While Android provides a large number of individual permissions, they are generally grouped into categories or features that provide similar functionality. **Table 12** shows system service categories.

| System Services | Description |
|---|---|
| Camera | Imaging capture (video and photos) |
| Microphone | Audio capture |
| Location | Location (GPS, GLONASS, etc) |
| Data Storage Protection | App data, App cache |
| Contacts/Address Book | Contacts and Address book |
| Calendar | Calendar |
| File / Storage Access | Access to internal and external storage data |
| Device Identifier information | Unique device identifying information (IMEI, Device/CPU ID, etc.) |
| Text Messages | SMS and RMS messaging |
| Telephony | Calls, call logs, voicemail |
| Cellular (Mobile) data | Cellular networks |
| Bluetooth | Bluetooth devices, paired devices, current connections. |
| NFC | Near Field Communications |
| Network Access | Current network state and network details |
| VPN | Virtual Private Networks |
| Body Sensors | Accelerometer, Bluetooth monitors |
| Fingerprint / Biometrics | Fingerprint, Iris, other authentication biometrics |
| Credential Access | Password metrics, quality, complexity |

**Table 12 Functional Categories**

**MDFPP32:FDP_ACF_EXT.1.2:**

Applications with a common developer have the ability to allow sharing of data between their applications. A common application developer can sign their generated APK with a common certificate or key and set the permissions of their application to allow data sharing. When the different applications' signatures match and the proper permissions are enabled, information can then be shared as needed.

The TOE supports Enterprise profiles to provide additional separation between application and application data belonging to the Enterprise profile. Applications installed into the Enterprise versus Personal profiles cannot access each other's secure data, applications, and can have separate device administrators/managers. This functionality is built into the device by default and does not require an application download. The Enterprise administrative app (an MDM agent application installed into the Enterprise Profile) may enable cross-profile contacts search, in which case, the device owner can search the address book of the enterprise profile. Please see the Admin Guide for additional details regarding how to set up and use Enterprise profiles. Ultimately, the enterprise profile is under control of the personal profile. The personal profile can decide to remove the enterprise profile, thus deleting all information and applications stored within the enterprise profile. However, despite the "control" of the personal profile, the personal profile cannot dictate the enterprise profile to share applications or data with the personal profile; the enterprise profile MDM must allow for sharing of contacts before any information can be shared.

**MDFPP32:FDP_ACF_EXT.2:**

The TOE allows an administrator to allow sharing of the enterprise profile address book with the normal profile. Each application group (profile) has its own calendar as well as keychain (keychain is the collection of user [not application] keys, and only the user can grant the user's applications access to use a given key in the user's keychain), thus Android's personal and work profiles do not share calendar appointments nor keys.

**MDFPP32:FDP_DAR_EXT.1:**

The TOE provides Data-At-Rest AES-256 XTS hardware encryption for all data stored on the TOE in the user data partition (which includes both user data and TSF data). The TOE also has TSF data relating to key storage for TSF keys not stored in the system's Android Key Store. The TOE separately encrypts those TSF keys and data. Additionally, the TOE includes read-only filesystems (system and vendor) in which the TOE'S system executables, libraries, and their configuration data reside.

For its Data-At-Rest encryption of the data partition on the internal Flash (where the TOE stores all user data and all application data), the TOE uses an AES-256 bit DEK with XTS feedback mode to encrypt each file in the data partition using dedicated application processor hardware. The TOE uses File Based Encryption (FBE) to encrypt files either using Device Encryption (DE) or Credential Encryption (CE), where the latter files the TOE combines a key chaining to the REK with the user's password to derive the CE encryption keys.

**MDFPP32:FDP_DAR_EXT.2:**

The TOE provides a Java library for Sensitive Data Protection (SDP) that application developers can use to opt-in for sensitive data protection. When developers opt-in for SDP, all data that is received on the device destined for that application is treated as sensitive. This library calls into the TOE to generate an RSA key that acts as a master KEK for the SDP encryption process. When an application that has opted-in for SDP receives incoming data while the device is locked, an AES symmetric DEK is generated to encrypt that data. The public key from the master RSA KEK above is then used to encrypt the AES DEK. Once the device is unlocked, the RSA KEK private key is re-derived and can be used to decrypt the AES DEK for each piece of information that was stored while the device was locked. The TOE then takes that decrypted data and re-encrypts it following FDP_DAR_EXT.1.

**MDFPP32:FDP_IFC_EXT.1:**

The TOE will route all traffic other than traffic necessary to establish the VPN connection to the VPN gateway (when the gateway's configuration specifies so). The TOE includes an interceptor kernel module that controls inbound and output packets. When a VPN is active, the interceptor will route all incoming packets to the VPN and conversely route all outbound packets to the VPN before they are output.

Note that when the TOE tries to connect to a Wi-Fi network, it performs a standard captive portal check which sends traffic that bypasses the full tunnel VPN configuration in order to detect whether the Wi-Fi network restricts Internet access until one has authenticated or agreed to usage terms through a captive portal. If the administrator wishes to deactivate the captive portal check (in order to prevent the plaintext traffic), they may do this by following the instructions in the Admin Guide.

The only exception to all traffic being routed to the VPN is in the instance of ICMP echo requests. The TOE uses ICMP echo responses on the local subnet to facilitate network troubleshooting and categorizes it as a part of ARP. As such, if an ICMP echo request is issued on the subnet the TOE is part of, it will respond with an ICMP echo response, but no other instances of traffic will be routed outside of the VPN.

**MDFPP32:FDP_STG_EXT.1:**

The TOE'S Trusted Anchor Database consists of the built-in certs and any additional user or admin/MDM loaded certificates. The built-in certs are individually stored in the device's read-only system image in the /system/etc/security/cacerts directory, and the user can individually disable certs through Android's user interface [Settings->Security-> Trusted Credentials]. Because the built-in CA certificates reside on the read-only system partition, the TOE places a copy of any disabled built-in certificate into the /data/misc/user/X/cacerts-removed/ directory, where 'X' represents the user's number (which starts at 0). The TOE stores added CA certificates in the corresponding /data/misc/user/X/cacerts-added/ directory and also stores a copy of the CA certificate in the user's Secure Key Storage (residing in the /data/misc/keystore/user_X/ directory). The TOE uses Linux file permissions that prevent any mobile application or entity other than the TSF from modifying these files. Only applications registered as an administrator (such as an MDM Agent Application) have the ability to access these files, staying in accordance to the permissions established in FMT_SMF_EXT.1 and FMT_MOF_EXT.1.

**MDFPP32:FDP_UPC_EXT.1/APPS:**

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

SSL:

javax.net.ssl.SSLContext:

https://developer.android.com/reference/javax/net/ssl/SSLSocket

Developers then need to swap SocketFactory for SecureSocketFactory, part of a private library provided by Google.

Developers can request this library by emailing: niapsec@google.com

HTTPS:

javax.net.ssl.HttpsURLConnection:

https://developer.android.com/reference/javax/net/ssl/HttpsURLConnection

Developers then need to swap HTTPSUrlConnections for SecureUrl part of a private library provided by Google.

Developers can request this library by emailing: niapsec@google.com

**MDFPP32:FDP_UPC_EXT.1/BLUETOOTH:**

The TOE supports a means for non-TSF applications to initiate Bluetooth BD/EDR and LE connections.

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

Bluetooth:

android.bluetooth:

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

## 6.4 Identification and authentication

**MDFPP32:FIA_AFL_EXT.1:**

The TOE maintains in persistent storage, for each user, the number of failed password logins since the last successful login (the phone, in its evaluated configuration, only supports password authentication), and upon reaching the maximum number of incorrect logins, the TOE performs a full wipe of all protected data (and in fact, wipes all user data). An administrator can adjust the number of failed logins for the cryptlock screen from the default of ten failed logins to a value between 0 (deactivate wiping) and 50 through an MDM. The TOE validates passwords by providing them to Android's Gatekeeper (which runs in the Trusted Execution Environment). If the presented password fails to validate, the TOE increments the incorrect password counter before displaying a visual error to the user. Android's Gatekeeper keeps this password counter in persistent secure storage and increments the counter before validating the password. Upon successful validation of the password, this counter is reset back to zero. By storing the counter persistently, and by incrementing the counter prior to validating it, the TOE ensures a correct tally of failed attempts even if it loses power.

**BT10: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). The TOE requires this authorization (via manual input) for mobile application use of the Bluetooth trusted channel and in situations where temporary (non-bonded) connections are formed.

**BT10:FIA_BLT_EXT.2:**

The TOE prevents data transfer of any type until Bluetooth pairing has completed. Additionally, the TOE supports OBEX (OBject Exchange) through L2CAP (Logical Link Control and Adaptation Protocol). Any new connection or data transfer attempts must first be accepted by the user, else the connection/data is rejected. In the event that there is no user input, the request times out and defaults to rejecting the attempt.

**BT10:FIA_BLT_EXT.3:**

The TOE rejects duplicate Bluetooth connections by only allowing a single session per paired device. This ensures that when the TOE receives a duplicate session attempt while the TOE already has an active session with that device, then the TOE ignores the duplicate session.

**BT10:FIA_BLT_EXT.4:**

The TOE'S Bluetooth host and controller supports Bluetooth Secure Simple Pairing and the TOE utilizes this pairing method when the remote host also supports it.

**BT10:FIA_BLT_EXT.6:**

The TOE requires explicit user authorization before granting trusted remote devices access to services associated with the OPP and MAP Bluetooth profiles. Additionally, the TOE requires explicit user authorization before granting untrusted remote devices access to services associated with all following Bluetooth profiles.

**BT10:FIA_BLT_EXT.7:**

The TOE requires explicit user authorization before granting trusted remote devices access to services associated with any available Bluetooth profile.

**WLANCEP10: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).

**MDFPP32:FIA_PMG_EXT.1:**

The TOE authenticates the user through a password consisting of basic Latin characters (upper and lower case, numbers, and the special characters noted in the selection (see the selections in section 5 for FIA_PMG_EXT.1)). The TOE defaults to requiring passwords to have a minimum of four characters but no more than sixteen, contain at least one letter; however, an MDM application can change these defaults. The Smart Lock feature is not allowed in the evaluated configuration as this feature circumvents the requirements for FIA_PMG_EXT.1 and many others.

**MDFPP32:FIA_TRT_EXT.1:**

Android's GateKeeper throttling is used to prevent brute-force attacks. After a user enters an incorrect password, GateKeeper APIs return a value in milliseconds (500ms default) in which the caller must wait before attempting to validate another password. Any attempts before the defined amount of time has passed will be ignored by GateKeeper. Gatekeeper also keeps a count of the number of failed validation attempts since the last successful attempt. These two values together are used to prevent brute-force attacks of the TOE's password.

**MDFPP32:FIA_UAU.5:**

The TOE, in its evaluated configuration, allows the user to authenticate using a password. Upon boot, the first unlock screen presented requires the user to enter their password to unlock the device.

Upon device lock during normal use of the device, the user has the ability to unlock the phone by entering their password. Throttling of this input can be read about in the FIA_AFL_EXT.1 section. The entered password is

compared to a value derived as described in the key hierarchy and key table above (FCS_STG_EXT.2 and FCS_CKM_EXT.4, respectively).

Some security related user settings (e.g. changing the password, setting up SmartLock, etc.) and actions (e.g. factory reset) require the user to enter their password before modifying these settings or executing these actions.

The TOE's evaluated configuration disallows other authentication mechanisms, such as pattern, PIN, or Smart Lock mechanisms (on-body detection, trusted places, trusted devices, trusted face, trusted voice).

**MDFPP32:FIA_UAU.6/CREDENTIAL, MDFPP32:FIA_UAU.6/LOCKED:**

The TOE requires the user to enter their password to unlock the TOE. Additionally, the TOE requires the user to confirm their current password when accessing the "Settings->Display->LockScreen->Screen Security->Select screen lock" menu in the TOE's user interface. The TOE can disable Smart Lock through management controls. Only after entering their current user password can the user then elect to change their password.

**MDFPP32: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.

**MDFPP32:FIA_UAU_EXT.1:**

As described before, the TOE's key hierarchy requires the user's password in order to derive the KEK_* keys in order to decrypt other KEKs and DEKs. Thus, until it has the user's password, the TOE cannot decrypt the DEK utilized for Data-At-Rest encryption, and thus cannot decrypt the user's protected data.

**MDFPP32:FIA_UAU_EXT.2:**

The TOE, when configured to require a user password, allows a user to perform the actions assigned in FIA_UAU_EXT.2.1 (see selections in section 5 for FIA_UAU_EXT.2) without first successfully authenticating. Choosing the input method allows the user to select between different keyboard devices (say, for example, if the user has installed additional keyboards). Note that the TOE automatically names and saves (to the internal Flash) any screen shots or photos taken from the lock screen, and the TOE provides the user no opportunity to name them or change where they are stored.

When configured, the user can also launch Google Assistant to initiate some features of the phone. However, if the command requires access to the user's data (e.g. contacts for calls or messages), the phone requires the user to manually unlock the phone before the action can be completed.

Beyond those actions, a user cannot perform any other actions other than observing notifications displayed on the lock screen until after successfully authenticating. Additionally, the TOE provides the user the ability to hide the contents of notifications once a password (or any other locking authentication method) is enabled.

**MDFPP32:FIA_X509_EXT.1:**

**WLANCEP10:FIA_X509_EXT.1/WLAN:**

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. Additionally, the TOE verifies the extendedKeyUsage Server Authentication purpose during WPA2/EAP-TLS negotiation. 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 if not yet valid, whether the certificate contains the appropriate X.509 extensions [e.g., the CA flag in the basic constraints extension for a CA certificate, or that a server certificate contains the Server Authentication purpose in the ExtendedKeyUsagefield]), then verifies each certificate in the chain (applying the same rules as above, but also ensuring that the Issuer of each certificate matches the Subject in the next rung "up" in the chain and that the chain ends in a self-signed certificate present in either the TOE'S trusted anchor database or matches a specified Root CA), and finally the TOE performs revocation checking for all certificates in the chain.

**MDFPP32:FIA_X509_EXT.2:**

**WLANCEP10:FIA_X509_EXT.2/WLAN:**

The TOE uses X.509v3 certificates during EAP-TLS, TLS, and HTTPS. The TOE comes with a built-in set of default 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 and an administrator/MDM can import a new trusted CA certificate into the Trust Anchor Database (the TOE stores the new CA certificate in the Security Key Store). Users and administrators/MDMs can also import new client certificates as well via the settings UI and the TOE's MDM APIs, respectively. Users then select which client certificate to present during configuration of the connection while administrators configure it while creating Wi-Fi connection profiles.

The TOE does not establish TLS connections itself (beyond EAP-TLS used for WPA2 Wi-Fi 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 be assured that the TOE will not establish the trusted connection if the peer certificate cannot be verified (including validity, certification path, and revocation [through OCSP]). If, during the process of certificate verification, the TOE cannot establish a connection with the server acting as the OCSP Responder, the TOE will not deem the server's certificate as valid and will not establish a TLS connection with the server.

The user or administrator explicitly specifies the trusted CA that the TOE will use for EAP-TLS authentication of the server's certificate. For mobile applications, the application developer will specify whether the TOE should use the Android system Trusted CAs, use application-specified trusted CAs, or a combination of the two. In this way, the TOE always knows which trusted CAs to use.

The TOE, when acting as a WPA2 supplicant uses X.509 certificates for EAP-TLS authentication. Because the TOE may not have network connectivity to a revocation server prior to being admitted to the WPA2 network and because the TOE cannot determine the IP address or hostname of the authentication server (the Wi-Fi access point proxies the supplicant's authentication request to the server), the TOE will accept the certificate of the server.

**MDFPP32:FIA_X509_EXT.3:**

The NIAPSEC library created by the vendor provides the following functions to allow for certificate path validation and revocation checking:

- public boolean isValid(List<Certificate> certs)
- public Boolean isValid(Certificate cert)

The first function allows for validation and revocation checking against a list of certificates, while the second checks a singular certificate. Revocation checking is completed using OCSP. Please see the FIA_X509_EXT.2/WLAN section for a further explanation on how the TOE handles revocation checking.

## 6.5 Security management

**MDFPP32:FMT_MOF_EXT.1:**

**MDFPP32:FMT_SMF_EXT.1:**

The TOE provides the management functions described in **Table 3 Security Management Functions** in section 5. The table includes annotations describing the roles that have access to each service and how to access the service. The TOE enforces administrative configured restrictions by rejecting user configuration (through the UI) when attempted. It is worth noting that the TOE'S ability to specify authorized application repositories takes the form of allowing enterprise applications (i.e., restricting applications to only those applications installed by an MDM Agent).

**BT10:FMT_SMF_EXT.1/BT:**

The TOE provides the management function described in **Table 4 Bluetooth Security Management Functions** in section 5. As with **Table 3 Security Management Functions**, the table includes annotations describing the roles that have access to each service and how to access the service. The TOE enforces administrative configured restrictions by

rejecting user configuration (through the UI) when attempted. It is worth noting that the TOE'S ability to specify authorized application repositories takes the form of allowing enterprise applications (i.e., restricting applications to only those applications installed by an MDM Agent).

**WLANCEP10:FMT_SMF_EXT.1/WLAN:**

The TOE provides the management functions described in **Table 5 WLAN Security Management Functions** in section 5. As with **Table 3 Security Management Functions**, the table includes annotations describing the roles that have access to each service and how to access the service. The TOE enforces administrative configured restrictions by rejecting user configuration (through the UI) when attempted. It is worth noting that the TOE'S ability to specify authorized application repositories takes the form of allowing enterprise applications (i.e., restricting applications to only those applications installed by an MDM Agent).

**MDFPP32:FMT_SMF_EXT.2:**

The TOE offers MDM agents the ability to wipe protected data, wipe sensitive data, remove Enterprise applications, and remove all device stored Enterprise resource data upon un-enrollment. The TOE offers MDM agents the ability to wipe protected data (effectively wiping the device) at any time. Similarly, the TOE also offers the ability to remove Enterprise applications and a full wipe of managed profile data of the TOE'S Enterprise data/applications at any time.

**MDFPP32:FMT_SMF_EXT.3:**

The TOE offers MDM agents and the user (through the "Settings->Security->Device administrators" menu) the ability to view each application that has been granted admin rights, and further to see what operations each admin app has been granted.

## 6.6 Protection of the TSF

**MDFPP32:FPT_AEX_EXT.1:**

The Linux kernel of the TOE'S Android operating system provides address space layout randomization utilizing the get_random_int(void) kernel random function to provide eight unpredictable bits to the base address of any user-space memory mapping. The random function, though not cryptographic, ensures that one cannot predict the value of the bits.

**MDFPP32:FPT_AEX_EXT.2:**

The TOE utilizes a 5.4 Linux kernel (https://source.android.com/devices/architecture/kernel/modular-kernels#core-kernel-requirements), 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. The Android operating system (as of Android 2.3) sets the ARM No eXecute (XN) bit on memory pages and the TOE'S ARMv8 Application Processor's Memory Management Unit (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 D.5 of the ARMv8 Architecture Reference Manual contains additional details about the MMU of ARM-based processors: http://infocenter.arm.com/help/index.jsp?topic=/com.arm.doc.ddi0487a.f/index.html.

**MDFPP32:FPT_AEX_EXT.3:**

The TOE's Android operating system provides explicit mechanisms to prevent stack buffer overruns in addition to taking advantage of hardware-based No eXecute to prevent code execution on the stack and heap. Specifically, the vendor builds the TOE (Android and support libraries) using gcc-fstack-protector compile option to enable stack overflow protection and Android takes advantage of hardware-based eXecute-Never to make the stack and heap non-executable. The vendor applies these protections to all TSF executable binaries and libraries.

**MDFPP32: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 its REK by limiting access to only trusted applications within the TEE (Trusted Execution Environment). The TOE key manager includes a TEE module which utilizes the REK 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.

Additionally, the TOE'S Android operating system provides 'sandboxing' that ensures that each third-party 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 or files and cannot access the memory space or files of other applications (notwithstanding access between applications with a common application developer).

The TOE has a locked bootloader, which restricts a user to installing a new software image through the Zebra's proscribed OTA (Over The Air) methods. The TOE allows an operator to download and install an OTA update through the system settings (Settings->System->Advanced->System update->Check for update) while the phone is fully booted, or by separately downloading an OTA image, and then "sideloading via ADB" the OTA update from Android's recovery mode. In both cases, the TOE will verify the digital signature of the new OTA before applying the new firmware.

No USSD nor MMI codes are available to be used while the phone is in the locked state. The user can only be presented with a dialer from the lock screen by selecting the "Emergency" button. From this dialer, the user is only allowed to dial a specific set of emergency phone numbers; any attempts to enter a USSD or MMI code results in a pop-up message stating "Can't call. <Phone number> is not an emergency number." and the call is not made/the USSD or MMI code is not submitted.

**MDFPP32:FPT_AEX_EXT.5:**

The TOE models provide Kernel Address Space Layout Randomization (KASLR) as a hardening feature to randomize the location of kernel data structures at each boot, including the core kernel as a random physical address, mapping the core kernel at a random virtual address in the vmalloc area, loading kernel modules at a random virtual address in the vmalloc area, and mapping system memory at a random virtual address in the linear area. The entropy used to dictate the randomization is based on the hardware present within the phone. For ARM devices, such as the TOE, 13–25 bits of entropy are generated on boot, from which the starting memory address is generated.

**MDFPP32:FPT_BBD_EXT.1:**

The TOE'S hardware and software architecture ensures separation of the application processor (AP) from the baseband or communications processor (CP) through internal controls of the TOE'S SoC, which contains both the AP and the CP. The AP restricts hardware access control through a protection unit that restricts software access from the baseband processor through a dedicated 'modem interface'. The protection unit combines the functionality of the Memory Protection Unit (MPU), the Register Protection Unit (RPU), and the Address Protection Unit (APU) into a single function that conditionally grants access by a master to a software defined area of memory, to registers, or to a pre-decoded address region, respectively. The modem interface provides a set of APIs (grouped into five categories) to enable a high-level OS to send messages to a service defined on the modem/baseband processor. The combination of hardware and software restrictions ensures that the TOE'S AP prevents software executing on the modem or baseband processor from accessing the resources of the application processor (outside of the defined methods, mediated by the application processor).

**MDFPP32:FPT_JTA_EXT.1:**

The TOE'S prevents access to its processor's JTAG interface by requiring use of a signing key to authenticate prior to gaining JTAG access. Only a JTAG image with the accompanying device serial number (which is different for each mobile device) that has been signed by the vendor's private key can be used to access a device's JTAG interface. The private key corresponds to the vendor's RSA 2048-bit public key (a SHA-256 hash of which is fused into the TOE'S application processor).

**MDFPP32:FPT_KST_EXT.1:** (KMD)

The TOE does not store any plaintext key material in its internal Flash; 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 depletes or is removed), all keys stored in the internal Flash are wrapped with a KEK. Please

refer to section 6.2 of the TSS for further information (including the KEK used) regarding the encryption of keys stored in the internal Flash. As the TOE encrypts all keys stored in Flash, upon boot-up, the TOE presents a password authentication screen before any functionality is unlocked. Prior to the user authenticating with the password, all DEKs, stored keys, and data remain encrypted. Upon user authentication, the password is used in conjunction to the REK to decrypt all DEKs, stored keys, and data and they become available for use. Further information about this process can be seen in the FDE Key Hierarchy slide in the KMD.

**MDFPP32:FPT_KST_EXT.2:**

The TOE itself (i.e., the mobile device) comprises a cryptographic module that utilizes cryptographic libraries including BoringSSL, application processor cryptography (which leverages AP hardware), and the following system-level executables that utilize KEKs: vold, wpa_supplicant, and the Android Key Store.

1. vold and QCT's application processor hardware provides Data-At-Rest encryption of the user data partition in Flash
2. wpa_supplicant provides 802.11-2014/WPA2 services
3. the Android Key Store application provides key generation, storage, deletion services to mobile applications and to user through the UI

The TOE ensures that plaintext key material is not exported by not allowing the REK to be exported 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_EXT.4).

**MDFPP32:FPT_KST_EXT.3:**

The TOE does not provide any way to export plaintext DEKs or KEKs (including all keys stored in the Android Key Store) as the TOE chains or directly encrypts all KEKs to the REK.

Furthermore, the components of the device are designed to prevent transmission of key material outside the device. Each internal system component requiring access to a plaintext key (for example the Wi-Fi driver) must have the necessary precursor(s), whether that be a password from the user or file access to key in Flash (for example the encrypted AES key used for encryption of the Flash data partition). With those appropriate precursors, the internal system-level component may call directly to the system-level library to obtain the plaintext key value. The system library in turn requests decryption from a component executing inside the trusted execution environment and then directly returns the plaintext key value (assuming that it can successfully decrypt the requested key, as confirmed by the CCM/GCM verification) to the calling system component. That system component will then utilize that key (in the example, the kernel which holds the key in order to encrypt and decrypt reads and writes to the encrypted user data partition files in Flash). In this way, only the internal system components responsible for a given activity have access to the plaintext key needed for the activity, and that component receives the plaintext key value directly from the system library.

For a user's mobile applications, those applications do not have any access to any system-level components and only have access to keys that the application has imported into the Android Key Store. Upon requesting access to a key, the mobile application receives the plaintext key value back from the system library through the Android API. Mobile applications do not have access to the memory space of any other mobile application so it is not possible for a malicious application to intercept the plaintext key value to then log or transmit the value off the device.

**MDFPP32:FPT_NOT_EXT.1:**

When the TOE encounters a critical failure (either a self-test failure or TOE software integrity verification failure), a failure is message is displayed to the screen, and the TOE attempts to reboot. If the failure persists between boots, 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.

**MDFPP32:FPT_STM.1:**

The TOE requires time for the Package Manager (which installs and verifies APK signatures and certificates), image verifier, wpa_supplicant, and Android Key Store applications. These TOE components obtain time from the TOE

using system API calls [e.g., time() or gettimeofday()]. An application (unless a system application is residing in /system/priv-app or signed by the vendor) 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. Further, this stored time is used both for the time/date tags in audit logs and is used to track inactivity timeouts that force the TOE into a locked state.

**MDFPP32:FPT_TST_EXT.1:**

**WLANCEP10:FPT_TST_EXT.1/WLAN:**

The TOE automatically performs known answer power on self-tests (POST) on its cryptographic algorithms to ensure that they are functioning correctly. Each component providing cryptography (application processor, and BoringSSL) performs known answer tests on their cryptographic algorithms to ensure they are working correctly. Should any of the tests fail, the TOE displays an error message stating "Boot Failure" and halts the boot process, displays an error to the screen, and forces a reboot of the device.

| Algorithm | Implemented in | Description |
|---|---|---|
| AES encryption/decryption | BoringSSL | Comparison of known answer to calculated value |
| ECDH key agreement | BoringSSL | Comparison of known answer to calculated value |
| DRBG random bit generation | BoringSSL | Comparison of known answer to calculated value |
| ECDSA sign/verify | BoringSSL | Comparison of known answer to calculated value |
| HMAC-SHA | BoringSSL | Comparison of known answer to calculated value |
| RSA sign/verify | BoringSSL | Comparison of known answer to calculated value |
| SHA hashing | BoringSSL | Comparison of known answer to calculated value |
| AES encryption/decryption | Application Processor | Comparison of known answer to calculated value |
| HMAC-SHA | Application Processor | Comparison of known answer to calculated value |
| DRBG random bit generation | Application Processor | Comparison of known answer to calculated value |
| SHA hashing | Application Processor | Comparison of known answer to calculated value |
| AES-XTS encrypt/decrypt | Application Processor | Comparison of known answer to calculated value |

**Table 13 Power-up Cryptographic Algorithm Known Answer Tests**

The WLAN's supplicant links against BoringSSL, so it utilizes the same KAT self-tests described above. All TSF-related modules are subject to these self-tests, which ensures that all TSF functionality is verified with each boot.

All executable modules stored on the TOE are verified for integrity via dm-verity, a file system integrity checking module. The dm-verity feature looks at a block device, the underlying storage layer of the file system, and determines if it matches its expected configuration. It does this using a cryptographic hash tree. For every block (typically 4k), there is a SHA256 hash. This partition-wide integrity verification applies to the partition that houses all TSF function executable modules (BoringSSL and, by association, WLAN supplicant), guaranteeing that these modules remain unmodified upon boot.

Should dm-verity's integrity check return a failure, the boot process halts and the device reboots, preventing an attacker from successfully loading and running a compromised module onto the TOE.

**MDFPP32:FPT_TST_EXT.2/POSTKERNEL:**

**MDFPP32:FPT_TST_EXT.2/PREKERNEL:**

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 it loads. The TOE performs checking of the entire /system and /vendor partition through use of Android's dm_verity mechanism (and while the TOE will still operate, it will log any blocks/executables that have been modified). Dm_verity looks at the underlying storage layer of the file system, and determine if it matches its expected configuration using a cryptographic hash tree.

One can consider the TOE's bootloader mode as an auxiliary boot mode, and upon the user pressing a specific combination of physical buttons, the TOE halts its boot process while in the bootloader (and the automatic boot of Android. Until the user has booted to Android, authenticated, and then elected to unlock the bootloader (a process

that wipes all phone data), the TOE's bootloader mode only provides to additional status commands. As the TOE always executes the bootloader during its normal boot process, the TOE always checks its integrity, and (typically automatically) then verifies the integrity of the Android kernel and boots it.

**MDFPP32: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 software version) 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.

**MDFPP32:FPT_TUD_EXT.2:**

The TOE verifies all OTA (Over The Air) updates to the TOE software (which includes baseband processor updates) using a public key chaining ultimately to the Root Public Key, a hardware protected key whose SHA-256 hash resides inside the application processor. Should this verification fail, the software update will fail and the update will not be installed.

The application processor 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 Root Public Key, 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.

**MDFPP32:FPT_TUD_EXT.3:**

Android verifies the authenticity of applications by verifying the Android APK signature prior to installing the file (additionally, Android ensures that updates to existing applications use the same signing certificate).

**MDFPP32:ALC_TSU_EXT.1:**

Google supports a bug filing system for the Android OS outlined here: https://source.android.com/setup/contribute/report-bugs. This allows developers or users to search for, file, and vote on bugs that need to be fixed. This helps to ensure that all bugs that affect large numbers of people get pushed up in priority to be fixed.

The vendor also supports their own form of bug reporting, via their website: zebra.com/us/en/about-zebra/contact-zebra/contact-tech-support.html

Google publishes monthly security updates which the vendor reviews and implements on their devices, releasing as a part of their own monthly security update cycle. Once updates are available, they are immediately made available on Zebra's website here: https://www.zebra.com/us/en/support-downloads.html.

## 6.7 TOE access

**MDFPP32:FTA_SSL_EXT.1:**

The TOE transitions to its locked state either immediately after a User initiates a lock by pressing the power button (if configured) or after a (also configurable) period of inactivity, and as part of that transition, the TOE will display a lock screen to obscure the previous contents and play a "lock sound" to indicate the phone's transition; however, the TOE'S lock screen still displays email notifications, calendar appointments, user configured widgets, 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 the actions assigned in FIA_UAU_EXT.2.1 (see selections in section 5).

Note that during power up, the TOE presents the user with an unlock screen stating "unlock for all features and data". While at this screen, the TOE has already decrypted Device Encrypted (DE) files within the user's data partition, but cannot yet decrypt the user's Credential Encrypted (CE) files. The user can only access a subset of device functionality before authenticating (e.g. the user can making an emergency call, receive incoming calls, receiving

alarms, and any other "direct boot" functionality). After the user enters their password, the TOE decrypts the user's CE files within the user data partition and the user has unlocked the full functionality of the phone. After this initial authentication, upon (re)locking the phone, the TOE presents the user with the previously mentioned KeyGuard lock screen. While locked, the actions described in FIA_UAU_EXT.2.1 are available for the user to utilize.

**MDFPP32:FTA_TAB.1:**

The TOE can be configured to display a user-specified message on the Lock screen, and additionally an administrator can also set a Lock screen message using an MDM.

**WLANCEP10: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, the security type, authentication protocol, and the client credentials to be used for authentication. 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 user 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 Wi-Fi radio.

## 6.8 Trusted path/channels

**MOD_BT_V1.0:FTP_BLT_EXT.1:**
**MOD_BT_V1.0:FTP_BLT_EXT.3/BR:**
**MOD_BT_V1.0:FTP_BLT_EXT.3/LE:**

The TOE provides support for both Bluetooth BR/EDR and Bluetooth LE connections. The TSF uses 128-bit keys to encrypt Bluetooth connections (BR/EDR and LE) and provides no method to configure alternate key sizes. Bluetooth encryption is enabled by default.

**MOD_BT_V1.0:FTP_BLT_EXT.2:**

The TOE requires an encrypted connection between itself and another Bluetooth device, and should a remote device stop encryption, the TSF will terminate the connection. The remote device can only attempt to re-establish a new, encrypted channel (and if the connection were no encrypted, the TOE would refuse the connection).

**MDFPP32:FTP_ITC_EXT.1:**

**WLANCEP10:FTP_ITC_EXT.1/WLAN:**

The TOE provides secured (encrypted and mutually authenticated) communication channels between itself and other trusted IT products through the use of IEEE 802.11-2012, 802.1X, and EAP-TLS and TLS, HTTPS. The TOE permits itself and applications to initiate communicate via the trusted channel, and the TOE initiates communications via the WPA2 (IEEE 802.11-2012, 802.1X with EAP-TLS) trusted channel for connection to a wireless access point. The TOE provides mobile applications and MDM agents access to HTTPS and TLS via published APIs, thus facilitating administrative communication and configured enterprise connections. These APIs are accessible to any application that needs an encrypted end-to-end trusted channel.