SONY

Common Criteria

Security target for Xperia™ devices

XPERIA

1 About this document

This Xperia™ in Business guide describes enterprise services and features that Sony Mobile offers in its Xperia devices. For specific details about supported products and software versions, visit www.sonymobile.com/global-en/xperia/business/it-support/.

1.1 Limitations to services and features

Some of the services and features described in this document might not be supported in all countries/regions or by all networks and/or service providers in all areas. Please contact your network operator or service provider to determine availability of any specific service or feature and whether additional access or usage fees apply.

1.2 Legal limitations

This document provides general information about developing a corporate mobile IT framework. It is a tool to help your company start judging its needs. It is not intended as a standalone guide for development of a mobile IT framework, does not replace the need for advice from qualified professionals, and is not in any way tailored to meet your company's specific needs. Sony Mobile may update this document at any time without notice. This document and its contents are provided "as is", so use at own risk. Sony Mobile expressly disclaims all warranties, express, implied, statutory, or otherwise. By continuing to read and enjoy this document you agree that Sony Mobile shall never be liable for any direct, indirect, special, consequential, or punitive damages, loss, or harm you may incur (including loss of data, use, profits, and/or business opportunity) whether or not Sony Mobile has been advised or was aware of the possibility of such damage, loss, or harm.

1.3 Trademarks and acknowledgements

All product and company names mentioned herein are the trademarks or registered trademarks of their respective owners. Any rights not expressly granted herein are reserved. All other trademarks are property of their respective owners.

Visit www.sonymobile.com/global-en/legal/ for more information.

1.4 Document release date

January 25, 2017.

This document is published by:
Sony Mobile Communications Inc., 4–12–3 Higashi-Shinagawa, Shinagawa-ku, Tokyo, 140–0002, Japan
www.sonymobile.com

First released version (October 2016)

Revision 1.2

2 Contents

3 Introduction

This document is the Common Criteria Security Target for Sony Xperia X and Sony Xperia X Performance for the hardware platforms listed in section 1.2. The Security Target is compliant with the NIAP Mobile Device Fundamentals Protection Profile Version 2.0, dated 17 September 2014 [MDFPP].

3.1 Security Target Identification

Title:	Common Criteria Security Target for Sony Xperia Devices
Version:	1.2
Date:	January 25, 2017
Developer:	Sony Mobile Communications
Keywords:	Mobile Phone, Smartphone, Common Criteria

3.2 TOE Identification

The TOE are the two smartphones Sony Xperia X and Sony Xperia X Performance. They both use Android™ version 6.0.1 (Marshmallow MR1). The kernel version is 3.18.20 in Xperia X Performance and the kernel version is 3.10.84 in Xperia X. Details on the two smartphone devices are shown below.

Device	Type number	Build	Qualcomm platform
Xperia X	PM-0930-BV	34.0.A.1.268	8956 (Snapdragon 650)
Xperia X Performance	PM-0940-BV	35.0.A.1.227	8996 (Snapdragon 820)

3.3 TOE Overview

The TOE is a Sony Xperia smartphone, the Sony Xperia X and Sony Xperia X Performance. The smartphones use Android 6.0.1 (Marshmallow MR1) with modification and extensions from Sony Mobile. The mobile operating system manages the device hardware and provides a platform for running Android apps.

The TOE provides the following security functionality:

- **Protected storage:** The TOE provides encryption of data and keys stored on the device to prevent unauthorized access to encrypted data. Access to encrypted data is only provided once the user has been successfully authenticated.

- **Mobile device configuration:** The TOE provides the capability to configure and apply security policies defined by the user and the administrators. The administrator may perform administration locally or using mobile device management system ensuring that administrator security policies have in precedence of user specified security policies.

- **Protected communication:** The TOE provides a trusted and encrypted channel between the TOE and remote networks for the protection of user and Enterprise data, as well as for configuration data.

- **Authentication:** The TOE provides user authentication and require that users are authenticated before accessing certain protected functionality and data. The TOE will automatically lock following a configured period of inactivity in an attempt to ensure authorization will be required in the event of the device being lost or stolen.

- **Mobile device integrity:** The TOE provide secure boot and performs self-tests to ensure the integrity of critical functionality, software/firmware and data has been maintained. The user is notified of any failure of these self-tests.

The TOE is intended to be used in enterprise environment that ensure that the TOE is configured and operated in accordance to the specific Common Criteria mode as described by the guidance.

3.4 TOE Description

The smartphones are based on Android 6.0 (Marshmallow). The mobile operating system manages the device hardware and provides a platform for running Android apps[1].

Since Android is a modern mobile platform that was designed to be open, it was possible for Sony Mobile to adopt and extend this platform with Sony specific features. The Android platform has been further developed by a number of Sony Mobile specific enhancements, including several security enhancements. Such as Xperia in Business that provides a package and a configuration of those features to the corporate user.

The main Android platform building blocks are:

- **Device Hardware**: Android runs on a wide range of hardware configurations including smartphones, tablets, and set-top-boxes. Android is processor-agnostic, but it takes advantage of hardware-specific security capabilities such as ARM v6 eXecute-Never.

- **Android Operating System**: The core operating system is built on top of the Linux kernel. All device resources, like camera functions, GPS data, Bluetooth® functions, telephony functions, network connections, etc. are accessed through the operating system.

- **Android Application Runtime**: Android applications are most often written in the Java™ programming language and run in the Dalvik™ virtual machine. However, many applications, including core Android services and applications are native applications or include native libraries. Both Dalvik and native applications run within the same security environment, contained within the Application Sandbox. Applications get a dedicated part of the file system in which they can write private data, including databases and raw files.

Android applications extend the core Android operating system. There are two primary sources for applications:

- **Pre-Installed Applications**: Android and Xperia in Business includes a set of pre-installed applications including phone, email, calendar, web browser, and contacts. These function both as user applications and to provide key device capabilities that can be accessed by other applications. Pre-installed applications are the ones that are part of the open source Android platform and of the Xperia in Business for these specific devices.

- **User-Installed Applications**: Android provides an open development environment supporting any third-party application. Google Play offers users hundreds of thousands of applications. No user-installed application are part of the TOE or required as part of the TOE environment.

Google provides a set of cloud-based services that are available to any compatible Android device. The primary services are:

- **Google Play™**: Google Play is a collection of services that allow users to discover, install, and purchase applications from their Android device or the web. Google Play makes it easy for developers to reach Android users and potential customers. Google Play also provides community review, application license verification, application security scanning, and other security services.

- **Android Updates**: The Android update service delivers new capabilities and security updates to Android devices, including updates through the web or over the air (OTA).

- **Application Services**: Frameworks that allow Android applications to use cloud capabilities such as (backing up) application data and settings and cloud-to-device messaging (C2DM) for push messaging.

These services, including some specific Xperia in Business services, such as Xperia Care, are out of scope of the TOE.

[1] Much of the information provided in this section originates from the Android Security documentation available here https://source.android.com/security/index.html.

3.4.1 TOE security architecture

This section describes the overall security architecture and the functionality of the Xperia android implementation. For a description of the logical and physical scope of the TOE see section 3.4.2 and 3.4.4. Securing an open platform requires a robust security architecture and rigorous security programs. Android was designed with multi-layered security that provides the flexibility required for an open platform, while providing protection for all users of the platform.

Xperia devices from Sony provide a multi-layer security architecture:

- System security - Xperia devices offer Linux kernel-level security from Android with Sony Mobile enhancements like Runtime integrity, HW and SW integrity and Secure Boot Chain.
- Secure storage - Devices are be protected by passwords and data on the devices is encrypted.
- Network security - Transmissions are encrypted and Xperia devices have built-in support for industry-standard VPN protocols.
- Device security - Administrators can control the use of certain features or apps on devices, e.g. data from lost devices can be wiped.
- Digital certificates - Xperia devices support digital certificates to enable authentication and authorisation of users connecting to corporate networks.

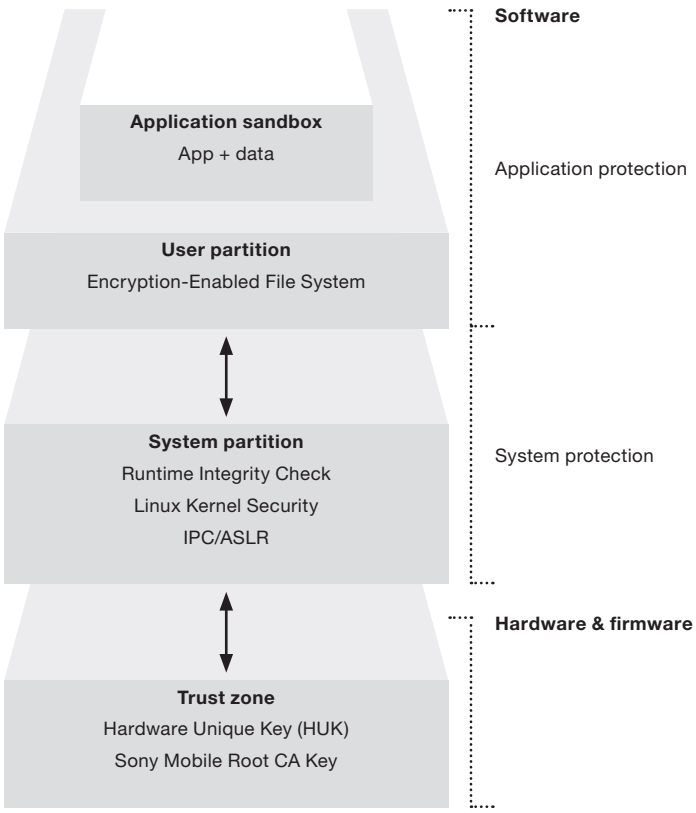

Figure 1 - The Android architecture used by Xperia

3.4.1.1 System security

The Android operating system offers a well-defined security architecture. As the Android OS is based on the Linux operating system, it takes advantage of the proven Linux kernel-level security model. The OS uses the Android Application Sandbox, which isolates application data and code execution from other applications. Since applications cannot interact with each other, and have limited access to the OS, sensitive information is protected if the user doesn't permit access. Android uses Security Enhanced (SE) Linux mandatory access control mechanism for increased separation at kernel level.

3.4.1.1.1 Application sandbox

The Application Sandbox in the OS kernel protects native code and OS applications. All software above the kernel, including libraries, application runtime and applications, runs inside the Application Sandbox. The fact that the Android platform does not allow applications on the device to interact with each other and limits their access to the OS is key to enforcing security in Android devices. This system is referred to as the Android Application Sandbox. The Android OS assigns a unique Linux user ID to each application. The application then runs as a unique Linux user in a separate process. This means that if one application tries to read data or start a process in another application without permission, this action is stopped by the OS since the instigating application doesn't have the appropriate user privileges.

3.4.1.1.2 Application code signing

Each application that is used in the Application Sandbox on an Android device must be signed. Without a legitimate signature, an application cannot be installed; it will get rejected by either Google Play or by the package installer on the Android device. The certificate of the signed application defines the user ID that is associated with that application. Application signing also ensures that one application cannot access any other application except through well-defined inter-process communication (IPC). In addition, apps available from Google Play are automatically scanned for malware.

3.4.1.1.3 User-based permissions for applications

Without a user's explicit permission, an Android application cannot access any system resources or sensitive APIs, with a few limited exceptions. Trusted applications can use sensitive APIs, but only after the user has given permission. Examples of sensitive APIs are camera functions; location data (GPS); Bluetooth wireless technology; telephony functions; SMS and MMS functions; and network and data connections. These API resources are only accessible through the OS. To be able to use a sensitive API on the device, an application must state which capabilities it needs. One of the steps when installing an Android application is to judge whether you want to approve the permissions that the application requests. At this stage in the installation process you can deny the application access and interrupt the installation. The permissions are only granted as long as the application is installed. If the user uninstalls the application, the permissions are removed.

3.4.1.1.4 Address space layout randomiser

The task of the Address Space Layout Randomiser (ASLR) is to make sure that system applications and libraries are stored in random locations in the memory. The Android OS uses this randomisation to protect the device against exploitation of the memory, and against malware getting installed on the device with the risk of corrupting the memory. ASLR prevents Return-Oriented Programming (ROP) attacks. Most binaries are randomised when executed because they are linked with the PIE (Position Independent Executable) flag. The linkers are randomised in the process address space. The Android OS has full stack, heap/brk, lib/mmap, linker, and executable ASLR.

3.4.1.1.5 Sony Mobile secure boot chain

Each step of the boot-up and the software update processes contains components that are cryptographically signed by Sony to ensure integrity of the core Android components, which is the basis for trusting the security functionality it provides.

3.4.1.1.6 Runtime integrity check

To further improve security in Xperia devices, Sony has introduced a runtime integrity check to detect runtime attacks. The runtime integrity check is integrated in the kernel and verifies that the mount table has not been modified. This is to prevent attackers from, for example, storing executables that remount and modify the system partition of the memory to make root access permanent.

3.4.1.2 Secure storage

Xperia devices provide proven methods for protecting sensitive information. Passwords prevent unauthorised use and data stored on the device is encrypted, making it unreadable to anyone but the intended user. The combined efforts of a strong password and encryption capabilities guarantee robust protection of sensitive data stored on Xperia devices, and a lost device can be remotely locked and wiped to protect sensitive content.

3.4.1.2.1 Encryption

Xperia devices offer full encryption with 256-bit AES for all user data in the internal storage, as well as any external SD card. This means that any data saved by and to applications, for example, email messages, email attachments, text and multimedia messages and contacts, is protected with a hardware encryption key against unauthorised access. A phone that ends up in the wrong hands does not risk having its file system broken into, thanks to the full file system encryption available in Xperia devices.

All data is encrypted and protected by the user password. If a device gets lost, confidential corporate information stays safe, and can only be accessed by knowing the password.

In addition, Xperia devices can defend themselves from dictionary password attacks by enforcing password complexity based on rules that your IT department can set.

The applications on Xperia devices, such as email, can use TLS to encrypt data sent between the Android OS and corporate services. To further enhance security in email, the Xperia email application offers S/MIME (Secure/Multipurpose Internet Mail Extensions). Note that this application is outside of the scope of the TOE for this Security Target.

3.4.1.2.2 Remote wipe

On an Xperia device that gets lost, stolen or otherwise compromised, the administrator can remotely remove all data from the device and deactivate it. This remote wipe procedure can be performed by users or administrators using a third party MDM solution or the 'my Xperia' service

3.4.1.2.3 Local wipe

Xperia devices can be set up to wipe all data and shut down after a set number of failed login attempts. This ability, known as local wipe, makes sure that brute force cracking attempts are unsuccessful. The number of failed attempts allowed before a local wipe occurs can be set by the user or the administrator using an MDM.

Small- and medium-sized companies that don't want to rely on Microsoft® Exchange ActiveSync® or MDM solutions for synchronisation and remote storage of information can use free-of-charge software from Sony Mobile that is made for Microsoft® Windows® and Mac OS computers. These tools can be used to back up and restore data locally on a computer. There is no need to create accounts or access the Internet.

3.4.1.2.4 Tools for Microsoft Windows and Mac® OS users

Sony Mobile provides additional products such as PC Companion for Microsoft Windows or Bridge for Mac to help the users managing the Xperia device.

Both PC Companion for Microsoft Windows and Bridge for Mac offers local backup and restore functions. When the software is installed on a computer, users simply connect their Xperia device to the computer using a Wi-Fi® or using a USB cable. Both products features software update and software repair functionality. Users can also use the tools to synchronise their device contacts and calendars directly with computers or local servers. The Backup & Restore function supports the backup of the call log as well as contacts stored locally on the device memory. Users can also back up text messages, bookmarks, system settings, application settings and data (availability depends on the application) and media files. The backup and restore procedure is performed between the Xperia device and the computer or a local server, that is, using a local connection that does not require Internet access.

Note that none of these tools, application or Mobile Device Management systems or agents on the phone are considered part of the TOE. The TOE only provides the API and capabilities to run and use these applications.

3.4.1.3 Network security

Xperia users within businesses and various organisations expect to be able to access corporate networks wherever they are. At the same time, they require that their data be protected over a reliable connection, with robust user-authorising methods in place. Xperia devices meet these security requirements whether users are connected via a mobile network or a Wi-Fi connection.

Using tethering, an Xperia device can also be turned into a mobile hotspot to access the Internet safely from a computer. In the corporate environment different network access methods and levels can be set to match corporate IT policies, depending on where the device is used and which tools are available.

3.4.1.3.1 Secure connections

To encrypt communication between Xperia devices and corporate services, Xperia devices uses the security standard Transport Layer Security (TLS v1.0, v1.1 and v1.2). Internet-based applications, such as the web browser, Email application and the Calendar, use TLS to encrypt information sent over the Internet. There is also support for StartTLS with IMAP/POP3 accounts.

3.4.1.3.2 Virtual private network (VPN)

Xperia users can connect to a corporate network with VPN access by using industry-standard protocols and user authentication. Xperia devices and the Android OS support several VPN technologies, which makes the integration of Xperia devices into an existing VPN solution easy. Compatibility with a wide selection of VPN

technologies combined with the Xperia device support for user authentication using the X.509 Digital Certificate Standard results in robust protection for all remote connections.

3.4.1.3.3 Wi-Fi

To provide a high level of protection for data transmissions over a Wi-Fi connection, Xperia devices use WPA2 Enterprise with 128-bit AES encryption. In addition to the encryption, protection is enhanced by requiring authentication for access to a wireless network. X.509 digital client certificates authenticate a user as a valid user before permitting access to the network. Xperia devices also support 802.1x wireless authentication methods, which means that they can be used with numerous RADIUS authentication solutions.

To enable easy setup when connecting to Wi-Fi networks, Xperia devices support Wi-Fi Protected Setup. Xperia devices can be set to automatically connect to Wi-Fi networks within range. Once set up, networks that require login credentials or other information are quickly accessed via automatic identification and web browser support. Once login credentials have been entered, they are reapplied when needed as long as the original login window in the web browser is kept in the background. In addition, Xperia devices support roaming based on the RSSI (Received Signal Strength Indicator) level, which improves the Wi-Fi connection and authentication for devices moving between access points. RSSI-level roaming improves connection reliability by automatically switching from an access point with a weakening signal to a neighbouring access point with a stronger signal.

3.4.1.3.4 Bluetooth

Xperia devices have support for Bluetooth® version 4.0, which enables secure connections to other devices, like computers, tablets, phones, printers or headsets, supported by Bluetooth technology. Xperia devices supply faster data transfer with Enhanced Data Rate (EDR) and they also have support for Secure Simple Pairing (SSP), enabling Public Key Infrastructure (PKI) encryption that protects against Man-in-the- middle (MITM) eavesdropping attacks and safeguards the integrity of the communication.

3.4.1.3.5 Enterprise single sign on

Enterprise single sign on (SSO) enables user credentials to be used across apps. Each new app configured with SSO verifies user permissions for enterprise resources, and logs users in without the need to re-enter passwords. The SSO solution used in Xperia devices is Kerberos compliant and uses Active Directory verification. Please, note that this security functionality is not within the scope of this evaluation.

3.4.1.4 Device Security

The screen lock combined with a passcode (a PIN or an alphanumeric password) is the first security barrier in preventing unauthorised users from gaining access to the entire device. It protects business as well as personal information. The passcode can be set by the user or enforced by the IT department. The complexity of the password and other password-related requirements is configurable, e.g. by using an MDM system.

In addition to enforcing passcode policies, the use of device management solutions with Xperia devices enables you to control device policies and device administration features. For example, you can restrict the use of certain features or apps on devices or wipe data from lost devices.

3.4.1.4.1 Passcode policies

IT administrators can choose from a wide range of passcode requirements when deploying Xperia devices in a corporate environment. In addition to requiring that an Xperia device is supplied with a passcode, you can enforce what length a PIN or a password must have through the Minimum password length policy. By using the Restrict password history policy, you can force users to create a new passcode that is different from their current passcode or a recently used passcode. This policy is often combined with the Password expiration timeout policy which forces users to update their passcode after a specified time period.

3.4.1.4.2 Device policies and administration

For an even higher level of security, you can add policies restricting the use of certain features on a device, or determine which features should be disabled or enabled. Security policies developed by Sony Mobile for Xperia devices include encryption of the external SD card. This is an addition to the Android OS support for device policies. You can, for instance, require that the storage of the device has to be encrypted, or that the camera should be disabled.

Xperia devices also support application blacklists and whitelists. This feature allows 3rd party MDMs (Mobile Device Management) to add and remove applications to the lists. Applications on the blacklist are disabled and cannot be started.

Within the device administration area, the Android OS provides a toolbox of administration features, ranging from the possibility to remotely lock a device and wipe its content (including the content on the external SD card) all the way through to remotely installing applications and updating installed applications.

3.4.1.4.3 Enforcing policies

By using device management solutions, an IT administrator can reach the whole fleet of Xperia devices used in a company. By managing devices from one central point, you can guarantee a high level of security by being able to enforce and monitor a wide range of parameters in the devices that access your corporate network and its sensitive data. You can achieve a comprehensive security setup as all devices in your network follow the same set of rules. You can configure different rule sets based on different user types.

When using Xperia devices, you can take advantage of the policies added by Sony Mobile as well as standard policies supported by the Android OS. You can remotely configure password settings, and push out policies to Xperia devices over the air using MDM solutions that support standard Android APIs. If the Xperia device uses a Microsoft Exchange account, you can push Microsoft Exchange ActiveSync policies over a mobile or Wi-Fi network.

3.4.1.4.4 Google Factory Reset Protection

By enabling the Google Factory Reset Protection (GFRP) you can set up your Xperia device to prevent other people from using it if it's been reset to factory settings without your permission. If your device is wiped – for example by performing a software repair using PC Companion or Bridge for Mac – a protective mode called Lockdown will be activated and only someone with your Google account or screen lock information can use the device.

In lockdown mode the device will be completely useless:

- All running applications are stopped.
- You can no longer maximize the notification panel.
- Incoming calls are sent to voicemail (if available).
- Outgoing calls (except emergency calls) are blocked.
- The USB port will only be available for charging. It will not be possible for a PC or Mac to detect the device. This also applies to a software repair or flash.

Lockdown will remain in effect until the user has validated the device using the correct Google credentials. Reloading software or rebooting the device will not disable the service.

3.4.1.5 Digital certificates

Xperia devices support digital certificates, providing businesses and organisations with a way to authenticate and authorise users to securely and efficiently transfer information to and from corporate networks. In addition, digital certificates enable the encryption of data exchanged between servers and permitted devices. The security is built around the Public Key Infrastructure (PKI) framework, which uses trusted encryption keys to protect transmitted data.

Certificates are issued and approved by a Certificate Authority (CA). The CA could be an independent external company, which is recognised and mutually trusted, or an internal organisation within your business. Digital certificates can also authenticate a client or a device interacting with a network, attesting that the device really is the device that it claims to be. Moreover, certificates are used to verify the sender of, for example, email messages or documents, with the option of making sure the content is encrypted.

3.4.1.5.1 Server certificates

Xperia devices support client-server communication using Transport Layer Security (TLS). Authentication with server certificates follows the X.509 digital certificate standard. Server certificates are stored in the internal credential storage. Server certificates enable encrypted communication between the client and the server.

3.4.1.5.2 Client certificates

You can use client certificates as an efficient alternative to authentication by requiring a user name and password, or a token. EAS servers, VPN gateways or Wi-Fi access points can identify Xperia devices using client certificates before giving them access to a corporate network. In this setup, users must obtain and store the

certificate on the Xperia device before they can configure the device to use a VPN gateway or a corporate server. Client certificates may also be used to enable secure messaging using S/MIME. Client certificates are stored in the secure credential storage and protected by a user-selected password.

3.4.1.5.3 Certificate Pinning

Xperia devices support certificate pinning. Pinned domains will receive a certificate validation failure if the certificate does not chain to a set of expected certificates. This protects against possible compromise of Certificate Authorities.

3.4.1.6 Device management

Device management, such as installing and deletion of apps; changing the device settings, such as the passwords and password policies; changing the status of the device such as activate and deactivate Wi-Fi and Bluetooth; wipe and factory reset, are all management functions.

Device management can be performed in different ways: from the user interface of the device or remotely using a mobile device management system (MDM). Local device management is performed by the primary user (owner), secondary user or by the guest. Remote management using an MDM relies on a local MDM agent that is performing the devices management operation using the API of the TOE, such as the Xperia in Business API.

3.4.2 Security functions provided by the TOE

3.4.2.1 Protected storage

Full disk encryption is the process of encoding all user data on an Android device using an encrypted key. Once a device is encrypted, all user-created data is automatically encrypted before committing it to disk and all reads automatically decrypt data before returning it to the calling process. Full disk encryption is a core component of the TOE. It is always activated for the internal storage and cannot be deactivated by the user. The TOE also supports SD card encryption.

The TOE provides encryption of data and keys stored on the device to prevent unauthorized access to encrypted data. Access to encrypted data is only provided once the user has been successfully authenticated.

3.4.2.2 Mobile device configuration

The TOE provides the capability to configure and apply security policies defined by the user and the administrators. The administrator can perform administration using mobile device management system ensuring that administrator security policies have in precedence over any user specified settings.

3.4.2.3 Protected communication

The TOE provides a trusted and encrypted channel between the TOE and remote networks for the protection of user and Enterprise data, as well as for configuration data. Encrypted communication is possible for IP connections using Wi-Fi, mobile data, as well as for Bluetooth connections.

3.4.2.4 Authentication

The TOE provides user authentication and require that users are authenticated before accessing certain protected functionality and data. The TOE will automatically lock following a configured period of inactivity in an attempt to ensure authorization will be required in the event of the device being lost or stolen.

3.4.2.5 Mobile device integrity

The TOE provides a number of security features to protect the integrity of the TOE and TSF data:

- The TOE provides a trust anchor in the hardware protected key storage. The root of trust consists of the public key used to verify the signature on the boot image and the locked state of the device. This key is the basis for the verification of the integrity of the TSF and the TSF data. This key is cryptographically bound to every key managed by Keymaster.

- The TOE provides functions to perform self-tests and software integrity checks to identify corrupt software. If any of the self-tests fail, the TOE will not go into an operational mode. It also includes mechanisms (i.e., verification of the digital signature of each new boot 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.

- The TOE provides protection of memory pages, uses address space layout randomization and stack-based buffer overflow protections to minimize the potential to exploit application flaws. It is also designed to protect itself from modification by applications as well as to isolate the address spaces of applications from one another to protect those applications

- The separation mechanisms of the TOE, such as sandboxing and the SELinux mandatory access control mechanism will improve integrity by preventing interference between applications.

3.4.3 Required non-TOE Hardware/Software/Firmware

No additional hardware and software needed for the TOE to maintain its security.

However, for being able to use the TOE in its intended way, the TOE will need SIM card and a subscription with a mobile operator, as well as access to a Google play account, most enterprise users will likely also use some sort of MDM system for the configuring and management of the TOE.

3.4.4 Physical Scope of the TOE

The TOE is a mobile device, which is composed of a hardware platform and its system software. Excluded from the TOE are user applications that run on top of the operating system. The TOE is intended to be used as a

mobile device within an enterprise environment where the configuration of the device is managed through a compliant device management solution.

Within the scope of the TOE is the guidance provided to the user and administrator for the use and administration of the TOE within the enterprise environment.

- User guide Xperia™ X, F5121
- User guide Xperia™ X Performance, F8131
- Common Criteria Guide for Xperia™ devices

3.4.5 Logical Scope of the TOE

The TOE provides the security functionality required by [MDFPP]. The security functionality is described in section 3.4.2.

4 Conformance Claim

This Security Target is conformant to Common Criteria [CC] version 3.1 revision 4 Part1, Common Criteria Part 2 extended and Common Criteria Part 3 extended.

This Security Target claims exact conformance to version 2.0 of the Mobile Device Fundamentals Protection Profile [MDFPP]. The security problem definition, security objectives and security requirements in this Security Target are all taken from the Protection Profile performing only operations defined there.

5 Security Problem Definition

The security problem definition defines the security problem that is addressed by the TOE as well as the assumptions on the operational environment that are necessary for the TOE to be able to address the security problem.

The security problem definition has been taken directly from the [MDFPP] and is only reproduced here for the convenience of the reader.

5.1 Threats

T.EAVESDROP	**Network Eavesdropping** An attacker is positioned on a wireless communications channel or elsewhere on the network infrastructure. Attackers may monitor and gain access to data exchanged between the Mobile Device and other endpoints.
T.NETWORK	**Network Attack** An attacker is positioned on a wireless communications channel or elsewhere on the network infrastructure. Attackers may initiate communications with the Mobile Device or alter communications between the Mobile Device and other endpoints in order to compromise the Mobile Device. These attacks include malicious software update of any applications or system software on the device. These attacks also include malicious web pages or email attachments which are usually delivered to devices over the network.
T.PHYSICAL	**Physical Access** The loss or theft of the Mobile Device may give rise to loss of confidentiality of user data including credentials. These physical access threats may involve attacks which attempt to access the device through external hardware ports, through its user interface, and also through direct and possibly destructive access to its storage media. The goal of such attacks is to access data from a lost or stolen device which is not expected to return to its user. Note: Defending against device re-use after physical compromise is out of scope for this protection profile.
T.FLAWAPP	**Malicious or Flawed Application** Applications loaded onto the Mobile Device may include malicious or exploitable code. This code could be included intentionally by its developer or unknowingly by the developer, perhaps as part of a software library. Malicious apps may attempt to exfiltrate data to which they have access. They may also conduct attacks against the platform's system software which will provide them with additional privileges and the ability to conduct further malicious activities. Malicious applications may be able to control the device's sensors (GPS, camera, microphone) to gather intelligence about the user's surroundings even when those activities do not involve data resident or transmitted from the device. Flawed applications may give an attacker access to perform network-based or physical attacks that otherwise would have been prevented.
T.PERSISTENT	**Persistent Presence** Persistent presence on a device by an attacker implies that the device has lost integrity and cannot regain it. The device has likely lost this integrity due to some other threat vector, yet the continued access by an attacker constitutes an on-going threat in itself. In this case the device and its data may be controlled by an adversary at least as well as by its legitimate owner.

5.2 Assumptions

A.CONFIG	It is assumed that the TOE's security functions are configured correctly in a manner to ensure that the TOE security policies will be enforced on all applicable network traffic flowing among the attached networks.
A.NOTIFY	It is assumed that the mobile user will immediately notify the administrator if the Mobile Device is lost or stolen.
A.PRECAUTION	It is assumed that the mobile user exercises precautions to reduce the risk of loss or theft of the Mobile Device.

5.3 Organizational Security Policy

There are no organizational security policies.

6 Security Objectives

The security objectives have been taken directly the [MDFPP] and is only reproduced here for the convenience of the reader.

6.1 Security Objectives for the TOE

O.COMMS	**Protected Communications** To address the network eavesdropping and network attack threats described in Section 3.1, concerning wireless transmission of Enterprise and user data and configuration data between the TOE and remote network entities, conformant TOEs will use a trusted communication path. The TOE will be capable of communicating using one (or more) of these standard protocols: IPsec, TLS, HTTPS, or Bluetooth. The protocols are specified by RFCs that offer a variety of implementation choices. Requirements have been imposed on some of these choices (particularly those for cryptographic primitives) to provide interoperability and resistance to cryptographic attack. While conformant TOEs must support all of the choices specified in the ST, they may support additional algorithms and protocols. If such additional mechanisms are not evaluated, guidance must be given to the administrator to make clear the fact that they were not evaluated.
O.STORAGE	**Protected Storage** To address the issue of loss of confidentiality of user data in the event of loss of a Mobile Device (T.PHYSICAL), conformant TOEs will use data-at-rest protection. The TOE will be capable of encrypting data and keys stored on the device and will prevent unauthorized access to encrypted data.
O.CONFIG	**Mobile Device Configuration** To ensure a Mobile Device protects user and enterprise data that it may store or process, conformant TOEs will provide the capability to configure and apply security policies defined by the user and the Enterprise Administrator. If Enterprise security policies are configured these must be applied in precedence of user specified security policies.
O.AUTH	**Authorization and Authentication** To address the issue of loss of confidentiality of user data in the event of loss of a Mobile Device (T.PHYSICAL), users are required to enter an authentication factor to the device prior to accessing protected functionality and data. Some non-sensitive functionality (e.g., emergency calling, text notification) can be accessed prior to entering the authentication factor. The device will automatically lock following a configured period of inactivity in an attempt to ensure authorization will be required in the event of the device being lost or stolen. Authentication of the endpoints of a trusted communication path is required for network access to ensure attacks are unable to establish unauthorized network connections to undermine the integrity of the device. Repeated attempts by a user to authorize to the TSF will be limited or throttled to enforce a delay between unsuccessful attempts.
O.INTEGRITY	**Mobile Device Integrity** To ensure the integrity of the Mobile Device is maintained conformant TOEs will perform self-tests to ensure the integrity of critical functionality, software/firmware and data has been maintained. The user shall be notified of any failure of these self-tests. (This will protect against the threat T.PERSISTENT.) To address the issue of an application containing malicious or flawed code (T.FLAWAPP), the integrity of downloaded updates to software/firmware will be verified prior to installation/execution of the object on the Mobile Device. In addition, the TOE will restrict applications to only have access to

	the system services and data they are permitted to interact with. The TOE will further protect against malicious applications from gaining access to data they are not authorized to access by randomizing the memory layout.

6.2 Security Objectives for the Operational Environment

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

7 Extended Component Definition

This Security Target does not define any extended components, but relies on the extended components that are defined in the Mobile Device Fundamentals Protection Profile [MDFPP]. It consists of both extend SFR components as well as one extended SAR component. The extended components are used unchanged from the [MDFPP] and are therefore not reproduced. They are listed below only for the convenience of the reader.

Extended SFR	**Summary of the SFR**
FCS_CKM_EXT.1	Root Encryption Key
FCS_CKM_EXT.2	Data Encryption Key Generation
FCS_CKM_EXT.3	Key Encryption Key Generation
FCS_CKM_EXT.4	Key Destruction
FCS_CKM_EXT.5	TSF Wipe
FCS_CKM_EXT.6	Salt Generation
FCS_CKM_EXT.7	Bluetooth Key Generation
FCS_HTTPS_EXT.1	HTTPS Protocol
FCS_IV_EXT.1	Initialization Vector Generation
FCS_RBG_EXT.1	Random Bit Generation
FCS_SRV_EXT.1	Cryptographic Services for Apps
FCS_STG_EXT.1	Key Storage
FCS_TLSC_EXT.1	EAP-TLS Client Protocol
FCS_TLSC_EXT.2	TLS Client Protocol
FDP_ACF_EXT.1	Access Control of System Services & Files
FDP_DAR_EXT.1	Data-at-Rest Encryption
FDP_IFC_EXT.1	No Split-Tunneling VPN
FDP_STG_EXT.1	Trust Anchor Database
FDP_UPC_EXT.1	User Data Trusted Channel Communications
FIA_AFL_EXT.1	Authentication Failure Handling
FIA_BLT_EXT.1	Bluetooth User Authorization
FIA_PAE_EXT.1	802.1x Protocol
FIA_PMG_EXT.1	Password Size/Complexity Support
FIA_TRT_EXT.1	Authentication Throttling
FIA_UAU_EXT.1	Password Required to Decrypt
FIA_UAU_EXT.2	Timing of Authentication
FIA_UAU_EXT.3	Re-Authentication Conditions
FIA_X509_EXT.1	Certificate Validation Rules
FIA_X509_EXT.2	Certificate Usage Requirements
FIA_X509_EXT.3	Certificate Validation Services to Apps
FMT_MOF_EXT.1	Administrator Management Functions
FMT_SMF_EXT.1	All Management Functions
FMT_SMF_EXT.2	Remediation Actions for Unenrollment

FPT_AEX_EXT.1	Address-Space Layout Randomization
FPT_AEX_EXT.2	Memory Page Permissions
FPT_AEX_EXT.3	Overflow Protections
FPT_AEX_EXT.4	Domain Isolation
FPT_KST_EXT.1	No Plaintext Key Storage
FPT_KST_EXT.2	No Plaintext Key Transmission
FPT_KST_EXT.3	No Plaintext Key Export
FPT_NOT_EXT.1	Self-Test Notification and Failure Behavior
FPT_TST_EXT.1	Cryptographic Self-tests
FPT_TST_EXT.2	Secure Boot
FPT_TUD_EXT.1	TSF Version Queries
FPT_TUD_EXT.2	Trusted Software Updates
FTA_SSL_EXT.1	User- and TSF-Initiated Locking
FTA_WSE_EXT.1	Wireless Network Access
FTP_ITC_EXT.1	TSF Trusted Channel Communications
Extended SAR	**Summary of the SAR**
ALC_TSU_EXT.1	Timely Security Updates

8 Security Functional Requirements

This chapter describes the Security Functional Requirements (SFRs) for the TOE. The SFRs have been taken from [MDFPP], with selections, assignments and potential refinements being applied. Selections and assignments performed as required by [MDFPP] are marked in **bold**. Refinements are marked in ***bold, italics and underlined***.

Note that selections, assignments or refinements already applied in [MDFPP] are not identify in this Security Target. This Security Target also includes SFRs from appendix C of [MDFPP] based on selections made in the baseline SFRs.

8.1 Class: Cryptographic Support (FCS)

8.1.1 Cryptographic Key Management (FCS_CKM)

8.1.1.1 Cryptographic Key Generation

FCS_CKM.1(1) – Cryptographic Key Generation

FCS_CKM.1.1(1) The TSF shall generate asymmetric cryptographic keys in accordance with a specified cryptographic key generation algorithm.

- **RSA schemes using cryptographic key sizes of 2048-bit or greater that meet the following: FIPS PUB 186-4, "Digital Signature Standard (DSS)", Appendix B.3;**
- **ECC schemes using "NIST curves" P-256, P-384 and P-521 that meet the following: FIPS PUB 186-4, "Digital Signature Standard (DSS)", Appendix B.4.**

Application note: Not all keys are used in the same way. Different keys are used for different purposes as is described in the SFRs below.

8.1.1.2 Cryptographic key generation (WLAN)

FCS_CKM.1(2) – Cryptographic key generation

FCS_CKM.1.1(2) The TSF shall generate symmetric cryptographic keys in accordance with a specified cryptographic key generation algorithm PRF-384 and specified cryptographic key sizes 128 bits using a Random Bit Generator as specified in FCS_RBG_EXT.1 that meet the following: IEEE 802.11-2012.

8.1.1.3 Cryptographic Key Establishment

FCS_CKM.2.1(1) – Cryptographic key establishment

FCS_CKM.2.1(1) The TSF shall perform cryptographic key establishment in accordance with a specified cryptographic key establishment method:

- RSA-based key establishment schemes that meets the following: NIST Special Publication 800-56B, "Recommendation for Pair-Wise Key Establishment Schemes Using Integer Factorization Cryptography";

and

- **Elliptic curve-based key establishment schemes that meets the following: NIST Special Publication 800-56A, "Recommendation for Pair-Wise Key Establishment Schemes Using Discrete Logarithm Cryptography".**

8.1.1.3.1 Cryptographic Key Distribution (WLAN)

FCS_CKM.2.1(2) – Cryptographic key distribution

FCS_CKM.2.1(2) The TSF shall decrypt Group Temporal Key (GTK) in accordance with a specified cryptographic key distribution method AES Key Wrap in an EAPOL-Key frame that meets the following: NIST SP 800-38F, IEEE 802.11-2012 for the packet format and timing considerations and does not expose the cryptographic keys.

8.1.1.3.2 Cryptographic Key Support (REK)

FCS_CKM_EXT.1 – Extended: Cryptographic Key Support

FCS_CKM_EXT.1.1 The TSF shall support a **hardware-protected** REK with a key of size **256 bits**.

FCS_CKM_EXT.1.2 System software on the TSF shall be able only to request **NIST SP 800-108 key derivation** by the key and shall not be able to read, import, or export a REK.

FCS_CKM_EXT.1.3 A REK shall be generated by a RBG in accordance with FCS_RBG_EXT.1.

FCS_CKM_EXT.1.4 A REK shall not be able to be read from or exported from the hardware.

Application note: The element FCS_CKM_EXT.1.4 has been added as a selection-based requirement since "hardware-protected REK" was selected in FCS_CKM_EXT.1.1.

8.1.1.3.3 Cryptographic Data Encryption Keys

FCS_CKM_EXT.2 – Extended: Cryptographic Key Random Generation

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

8.1.1.3.4 Cryptographic Key Encryption Keys

FCS_CKM_EXT.3 – Extended: Cryptographic Key Generation

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

FCS_CKM_EXT.3.2 The TSF shall generate all KEKs using one of the following methods:

a) derive the KEK from a Password Authentication Factor using PBKDF and

c) Combine the KEK from other KEKs in a way that preserves the effective entropy of each factor by using an XOR operation, concatenating the keys and use a KDF (as described in SP 800-108), encrypting one key with another.

Application note: The TOE uses only alternative c) for all its KEKs. Alternative a. is provided to applications, that are not part of the TOE, such as the BackupManagerService that use a PBKDF generated KEK.

8.1.1.3.5 Cryptographic Key Destruction

FCS_CKM_EXT.4 – Extended: Key Destruction

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

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

Application note: According to the [MDFPP] plaintext keying material will refer to values derived from passwords for products entering evaluation after Quarter 3, 2015.

8.1.1.3.6 TSF Wipe

FCS_CKM_EXT.5 – Extended: TSF Wipe

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

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

8.1.1.3.7 Cryptographic Salt Generation

FCS_CKM_EXT.6 – Extended: Salt Generation

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

Application note: The salt is generated for specific algorithms and protocols of the TSF that requires salt. Salt is available from /dev/random which obtain its random source from the hardware secure module.

8.1.1.3.8 Cryptographic Key Generation (Bluetooth)

FCS_CKM_EXT.7 – Extended: Bluetooth Key Generation

FCS_CKM_EXT.7.1 The TSF shall randomly generate public/private ECDH key pairs every **time a pairing is done**.

8.1.1.4 Cryptographic Operation (FCS_COP)

8.1.1.4.1 Confidentiality Algorithms

FCS_COP.1(1) – Cryptographic operation

FCS_COP.1.1(1) The TSF shall perform encryption/decryption in accordance with a specified cryptographic algorithm

- AES-CBC (as defined in FIPS PUB 197, and NIST SP 800-38A) mode,
- AES-CCMP (as defined in FIPS PUB 197, NIST SP 800-38C and IEEE 802.11-2012)

and

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

Application note: The encryption algorithm AES-XTS is used for the Sony internal storage encryption.

8.1.1.4.2 Hashing Algorithms

FCS_COP.1(2) – Cryptographic operation

FCS_COP.1.1(2) The TSF shall perform cryptographic hashing in accordance with a specified cryptographic algorithm SHA-1 and **SHA-256** and message digest sizes 160 and **256** that meet the following: FIPS Pub 180-4.

Application note: SHA-1 for generating digital signatures is no longer allowed and SHA-1 for verification of digital signatures is strongly discouraged as there may be risk in accepting these signatures. Products entering into evaluation after Quarter 3, 2015 will be required to include SHA-2 algorithms.

8.1.1.4.3 Signature Algorithms

FCS_COP.1(3) – Cryptographic operation

FCS_COP.1.1(3) The TSF shall perform cryptographic signature services (generation and verification) in accordance with a specified cryptographic algorithm

- RSA schemes using cryptographic key sizes of 2048-bit or greater that meet the following: FIPS PUB 186-4, “Digital Signature Standard (DSS)”, Section 4

and

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

8.1.1.4.4 Keyed Hash Algorithms

FCS_COP.1(4) – Cryptographic operation

FCS_COP.1.1(4) The TSF shall perform keyed-hash message authentication in accordance with a specified cryptographic algorithm HMAC-SHA-1 and **HMAC-SHA-256** and cryptographic key sizes **160, 256** and message digest sizes 160 and **256** bits that meet the following: FIPS Pub 198-1, "The Keyed-Hash Message Authentication Code, and FIPS Pub 180-4, “Secure Hash Standard.

8.1.1.4.5 Password-Based Key Derivation Functions

FCS_COP.1(5) – Cryptographic operation

FCS_COP.1.1(5) The TSF shall perform Password-based Key Derivation Functions in accordance with a specified cryptographic algorithm HMAC-**SHA-1, SHA-256, SHA-384, SHA-512**, with **32768** iterations, and output cryptographic key sizes **256** that meet the following: NIST SP 800-132.

Application note: The PBKDF functionality is provided by the TOE, but is not used by other TSFs for key derivation. Instead for the other TSFs key derivation is provided using scrypt as part of the hardware security module.

8.1.1.5 HTTPS Protocol (FCS_HTTPS)

FCS_HTTPS_EXT.1 – Extended: HTTPS Protocol

FCS_HTTPS_EXT.1.1 The TSF shall implement the HTTPS protocol that complies with RFC 2818.

FCS_HTTPS_EXT.1.2 The TSF shall implement HTTPS using TLS (FCS_TLSC_EXT.2).

FCS_HTTPS_EXT.1.3 The TSF shall notify the application and **request application authorization to establish the connection** if the peer certificate is deemed invalid.

8.1.1.6 Initialization Vector Generation (FCS_IV)

FCS_IV_EXT.1 – Extended: Initialization Vector Generation

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

Application note: The reference to "Table 14: References and IV Requirements for NIST-approved Cipher Modes" is a reference to Table 14 in Annex H of the [MDFPP].

8.1.1.7 Random Bit Generation (FCS_RBG)

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

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)**.

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.

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.

8.1.1.8 Cryptographic Algorithm Services (FCS_SRV)

FCS_SRV_EXT.1 – Extended: Cryptographic Algorithm Services

FCS_SRV_EXT.1.1 The TSF shall provide a mechanism for applications to request the TSF to perform the following cryptographic operations:

- All mandatory and selected algorithms in FCS_CKM.2(1)
- The following algorithms in FCS_COP.1(1): AES-CBC, **AES Key Wrap, AES-GCM, AES_XTS**
- All mandatory and selected algorithms in FCS_COP.1(3)
- All mandatory and selected algorithms in FCS_COP.1(2)
- All mandatory and selected algorithms in FCS_COP.1(4)
- **All mandatory and selected algorithms in FCS_CKM.1(1),**
- **The selected algorithms in FCS_COP.1(5)**.

8.1.1.9 Cryptographic Key Storage (FCS_STG)

8.1.1.9.1 Secure Key Storage

FCS_STG_EXT.1 – Extended: Cryptographic Key Storage

FCS_STG_EXT.1.1 The TSF shall provide **hardware-isolated** secure key storage for asymmetric private keys and **symmetric keys**.

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

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

FCS_STG_EXT.1.4 The TSF shall have the capability to allow only the application that imported the key/secret the use of the key/secret. Exceptions may only be explicitly authorized by **a common application developer**.

FCS_STG_EXT.1.5 The TSF shall allow only the application that imported the key/secret to request that the key/secret be destroyed. Exceptions may only be explicitly authorized by **the user *or* a common application developer**.

8.1.1.10 TLS Client Protocol (FCS_TLSC)

8.1.1.10.1 EAP-TLS Client Protocol

FCS_TLSC_EXT.1 – Extended: EAP TLS Protocol

FCS_TLSC_EXT.1.1 The TSF shall implement TLS 1.0 and **TLS 1.1 (RFC 4346), and TLS 1.2 (RFC 5246)** supporting the following ciphersuites:

- Mandatory Ciphersuites:
  - TLS_RSA_WITH_AES_128_CBC_SHA as defined in RFC 5246
- ***and the following ciphersuites allowed as optional in [MDFPP]***:
  - **TLS_RSA_WITH_AES_256_CBC_SHA as defined in RFC 5246**

FCS_TLSC_EXT.1.2 The TSF shall verify that the server certificate presented for EAP-TLS **chains to one of the specified CAs**.

FCS_TLSC_EXT.1.3 The TSF shall not establish a trusted channel if the peer certificate is invalid.

FCS_TLSC_EXT.1.4 The TSF shall support mutual authentication using X.509v3 certificates.

FCS_TLSC_EXT.1.5 The TSF shall present the Supported Elliptic Curves Extension in the Client Hello with the following NIST curves: **secp256r1, secp384r1** and no other curves.

8.1.1.10.2 TLS Client Protocol

FCS_TLSC_EXT.2 – Extended: TLS Protocol

FCS_TLSC_EXT.2.1 The TSF shall implement TLS 1.2 (RFC 5246) supporting the following ciphersuites:

- Mandatory Ciphersuites:
  - TLS_RSA_WITH_AES_128_CBC_SHA as defined in RFC 5246
- ***and the following ciphersuites allowed as optional in [MDFPP]***:
  - **TLS_RSA_WITH_AES_256_CBC_SHA as defined in RFC 5246**
  - **TLS_DHE_RSA_WITH_AES_128_CBC_SHA as defined in 5246**
  - **TLS_DHE_RSA_WITH_AES_256_CBC_SHA as defined in 4492**
  - **TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA as defined in RFC 4492**
  - **TLS_ECDHE_RSA_WITH_AES_256_CBC_SHA as defined in RFC 4492**
  - **TLS_ECDHE_ECDSA_WITH_AES_128_CBC_SHA as defined in RFC 4492**
  - **TLS_ECDHE_ECDSA_WITH_AES_256_CBC_SHA as defined in RFC 4492**
  - **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**

FCS_TLSC_EXT.2.2 The TSF shall verify that the presented identifier matches the reference identifier according to RFC 6125.

FCS_TLSC_EXT.2.3 The TSF shall not establish a trusted channel if the peer certificate is invalid.

FCS_TLSC_EXT.2.4 The TSF shall support mutual authentication using X.509v3 certificates.

FCS_TLSC_EXT.2.5 The TSF shall present the Supported Elliptic Curves Extension in the Client Hello with the following NIST curves: **secp256r1, secp384r1** and no other curves.

8.2 Class: User Data Protection (FDP)

8.2.1 Access Control (FDP_ACF)

FDP_ACF_EXT.1 – Extended: Security access control

FDP_ACF_EXT.1.1 The TSF shall provide a mechanism to restrict the system services that are accessible to an application.

FDP_ACF_EXT.1.2 The TSF shall provide an access control policy that prevents **application processes** from accessing **all** data stored by other **application processes**. Exceptions may only be explicitly authorized for such sharing by **a common application developer**.

Application note: The Linux UID is determines the access rights. By default, Android will assign each application its own unique user ID. However, if this attribute is set to the same value for two or more applications, they will all share the same ID (provided that they are also signed by the same certificate). Application with the same user ID can access each other's data and, if desired, run in the same process.

8.2.2 Data-At-Rest Protection (FDP_DAR)

FDP_DAR_EXT.1 – Extended: Protected Data Encryption

FDP_DAR_EXT.1.1 Encryption shall cover all protected data.

FDP_DAR_EXT.1.2 Encryption shall be performed using DEKs with AES in the **XTS and CBC** mode with key size **256** bits.

Application note: The AES in XTS mode is used for the internal storage. For encryption of SD cards (extenal storage) AES in CBC mode is used.

8.2.3 Subset Information Flow Control – VPN (FDP_IFC)

FDP_IFC_EXT.1 – Extended: Subset information flow control

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

8.2.4 Certificate Data Storage (FDP_STG)

FDP_STG_EXT.1 – Extended: User Data Storage

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

8.2.5 Inter-TSF User Data Protected Channel (FDP_UPC)

FDP_UPC_EXT.1 – Extended: Inter-TSF user data transfer protection

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

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

8.3 Class: Identification and Authentication (FIA)

8.3.1 Authentication Failures (FIA_AFL)

FIA_AFL_EXT.1 – Extended: Authentication failure handling

FIA_AFL_EXT.1.1 The TSF shall detect when a configurable positive integer within **1 to 100** of unsuccessful authentication attempts occur related to last successful authentication by that user.

FIA_AFL_EXT.1.2 When the defined number of unsuccessful authentication attempts has been surpassed, the TSF shall perform wipe of all protected data.

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

Application note: The number of unsuccessful authentication attempts is configurable as specified as Nr 2 of FMT_SMF_EXT.1.

8.3.2 Bluetooth Authorization and Authentication (FIA_BLT)

FIA_BLT_EXT.1 – Extended: Bluetooth User Authorization

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

8.3.3 Port Access Entity Authentication (FIA_PAE)

FIA_PAE_EXT.1 – Extended: PAE Authentication

FIA_PAE_EXT.1 The TSF shall conform to IEEE Standard 802.1X for a Port Access Entity (PAE) in the "Supplicant" role.

8.3.4 Password Management (FIA_PMG)

FIA_PMG_EXT.1 – Extended: Password Management

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.

Application note: As information to the reader. Please, consider the language setting may determine the number of letter available for passwords as the alphabet differs between the different languages setting.

8.3.5 Authentication Throttling (FIA_TRT)

FIA_TRT_EXT.1 – Extended: Authentication Throttling

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

8.3.6 User Authentication (FIA_UAU)

8.3.7 Protected Authentication Feedback

FIA_UAU.7 – Protected authentication feedback

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.

8.3.8 Authentication for Cryptographic Operation

FIA_UAU_EXT.1 – Extended: Authentication for Cryptographic Operation

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

8.3.9 Timing of Authentication

FIA_UAU_EXT.2 – Extended: Timing of Authentication

FIA_UAU_EXT.2.1 The TSF shall allow:

- **Receive calls**
- **Enter unlocking code**
- **Make Emergency call**
- **See notifications**
- **Power off**
- **Restart**
- **Take screen picture**
- **Record Screen Video**
- **Use Voice assistant**
- **Take picture with Camera**
- **Adjust volume**
- **Adjust display brightness**
- **Set to air plane mode**
- **Turn on and off flashlight**
- **Turn on and off automatic screen rotation**
- **Mute device/ set to not disturb mode**
- **Turn on/off Bluetooth**
- **Turn on/off Wi-Fi**

on behalf of the user to be performed before the user is authenticated.

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

8.3.10 Re-Authentication

FIA_UAU_EXT.3 – Extended: Re-Authentication

FIA_UAU_EXT.3.1 The TSF shall require the user to enter the correct Password Authentication Factor when the user changes the Password Authentication Factor, and following TSF- and user-initiated locking in order to transition to the unlocked state, and **authenticate keystore, change password, change unlock method, allow boot-loader to be unlocked, factory reset from settings menu**.

Application note: In order to allow the boot loader to be unlocked Sony must allow boot loader unlock in SIM lock configuration. Factory reset will disables the anti theft functionality so it must be under password protection.

8.3.11 X509 Certificates (FIA_X509)

8.3.11.1 Validation of Certificates

FIA_X509_EXT.1 – Extended: Validation of certificates

FIA_X509_EXT.1.1 The TSF shall validate certificates in accordance with the following rules:

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

- The TSF shall validate the extendedKeyUsage field according to the following rules:
    - 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 extendedKeyUsage field.

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

8.3.11.2 X509 Certificate Authentication

FIA_X509_EXT.2 – Extended: X509 certificate authentication

FIA_X509_EXT.2.1 The TSF shall use X.509v3 certificates as defined by RFC 5280 to support authentication for EAP-TLS exchanges, and **IPsec, TLS, HTTPS**, and **code signing for system software updates, code signing for mobile applications, code signing for integrity verification, no additional uses**.

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

Application note: The TSF behavior when the TSF cannot establish a connection to determine the validity of a certificate differs between use cases. In case of code signing it will not accept the certificate. In case of HTTPS it will check with calling app, which prompts the user. Anyhow, it will no accept the certificate as such so the SFR is implemented.

8.3.11.3 Request Validation of Certificates

FIA_X509_EXT.3 – Extended: Request Validation of certificates

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

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

8.4 Class: Security Management (FMT)

8.4.1 Management of Functions in TSF (FMT_MOF)

FMT_MOF_EXT.1 – Extended: Management of security functions behavior

FMT_MOF_EXT.1.1 The TSF shall restrict the ability to perform the functions in column ***4*** of Table 1 to the user.

FMT_MOF_EXT.1.2 The TSF shall restrict the ability to perform the functions in column ***6*** of Table 1 to the administrator when the device is enrolled and according to the administrator-configured policy.

8.4.2 Specification of Management Functions (FMT_SMF)

8.4.2.1 Specification of Management Functions

FMT_SMF_EXT.1 – Extended: Specification of Management Functions

FMT_SMF_EXT.1.1 The TSF shall be capable of performing the following management functions:

Nr	Management function	FMT_SMF_EXT.1	*Users (exclusive)*	Administrator	*MDM Policy*

1	Configure password policy: a. minimum password length b. minimum password complexity c. maximum password lifetime Note: Administrators cannot set the policy directly, but using an app with (or an MDM client) they can set the policy for the device, i.e. for all users of the device.	M	–	M	M
2	Configure session locking policy: a. screen-lock enabled/disabled b. screen lock timeout c. number of authentication failures	M	–	M	M
3	Enable/disable the VPN protection: a. across device **c. no other method**	M	–	M	–
4	Enable/disable **NFC, Bluetooth, Wi-Fi and mobile data** Note: Bluetooth can be enabled and disabled for the whole device as well as for specific Bluetooth profiles (types of devices).	M	–	M	X
5	Enable/disable **camera and microphone**: a. across device **c. no other method** Note: While users may specify access to the camera and the microphone respectively on a per app basis, the MDM API can only allow or disallow this respectively for the whole device.	M	–	M	M
6	Specify wireless networks (SSIDs) to which the TSF may connect	M	–	M	X
7	Configure security policy for each wireless network: a. **specify the CA(s) from which the TSF will accept WLAN authentication server certificate(s)** b. security type c. authentication protocol d. client credentials to be used for authentication	M	–	M	–
8	Transition to the locked state	M	–	M	–
9	TSF wipe of protected data	M	–	M	–
10	Configure application installation policy by **a. restricting the sources of applications** **b. specifying a set of allowed applications based on an application whitelist** **c. denying installation of applications** Note: The TOE provides both whitelisting and blacklisting of applications using application names.	M	–	M	M
11	Import keys/secrets into the secure key storage Note: The keys are imported in a user and application context, such as keys will only be available only to specific users and application.	M	X	X	–
12	Destroy imported keys/secrets and **no other keys/secrets** in the secure key storage Note: Users can only destroy all keys/secrets, not individual keys. Administrators can, using the MDM API for factory rest destroy all keys/secrets.	M	X	X	–

13	Import X.509v3 certificates into the Trust Anchor Database	M	–	M	–
14	Remove imported X.509v3 certificates and **default X.509v3 certificates** in the Trust Anchor Database	M	–	X	–
15	Enroll the TOE in management	M	M	–	–
16	Remove applications	M	–	M	X
17	Update system software Note: The TOE supports MDM policy enforcement for preventing system software updates, not to enforce system software updates.	M	–	M	–
18	Install applications	M	–	M	X
19	Remove Enterprise applications Note: Enterprise applications are application that have been installed using an MDM.	M	–	M	–
20	Configure the Bluetooth trusted channel: a. disable/enable the Discoverable mode (for BR/EDR) b. change the Bluetooth device name **i. no other Bluetooth configuration**	M	–	X	–
21	Enable/disable display notification in the locked state of **f. all notifications** **Note:** Users may configure this on a per application basis, but the administrator and MDM interface applies to all types notifications.	M	–	X	X
23	Enable/disable **Wi-Fi tethering, USB tethering, and Bluetooth tethering**	X	–	X	X
35	Approve exceptions for destruction of keys/secrets by applications that did not import the key/secret	X	X	X	–
36	Configure the unlock banner Note: Using the MDM interface the administrator can specify a message that will be displayed for locked devices. This is typically used when specific unlock messages are necessary, i.e. for device that is in an improper state.	X	–	X	X
39	Enable/disable **a. USB mass storage mode**	X	–	X	X
40	Enable/disable backup to **locally connected system *or* remote system**	X	X	–	–
41	Enable/disable **a. Hotspot functionality authenticated by pre-shared key** **b. USB tethering authenticated by no authentication**	X	X	–	–
44	Enable/disable location services: a. across device **b. on a per-app basis** Note: This feature is available for the whole device as well as on a per user basis. Using an MDM it is also possible to prevent the user from adding new users.	M	–	X	X
45	**The following list of additional management functions are provided by the TOE:** **a. Enable/disable voice command control of device functions** **b. Enable/disable fingerprint (Touch ID) for unlock** **c. Add and remove users (secondary and guests only)** Note: Enable/disable voice control is exclusive under the control of the user and there is not APIs available for administrators for this.	 X X X	 X – X	 – X –	 – X –

	Enable and disable fingerprint can be performed by the users, but can be disabled by the administrator and enforced by an MDM policy, preventing the user to use fingerprint. The user can both add and remove users, guests, as well as MDM clients (enrolment). Using the MDM API users can only be removed by a factory reset. Other users, such as secondary can only remove users, but cannot add any users.				

Table 1 - Security Management Functions

Application note: The columns are consistent with the original table from the Protection Profile, but to make it easier to read they have been renamed. Management functions that are provided to the user and not the administrator are listed under "User", indicated in [MDFPP] as FMT_MOF_EXT.1.1; Management functions provided to the administrator (using an MDM agent API) are listed under "Administrator"; Management functions for which the administrator may set a policy that restricts the user from performing that function are listed in "MDM policy", indicated in [MDFPP] as FMT_MOF_EXT.1.2.

The mandatory requirements are all implemented and indicated with an **M** and the optional requirements that are implemented are indicated with an **X**.

8.4.2.2 Specification of Remediation Actions

FMT_SMF_EXT.2 – Extended: Specification of Remediation Actions

FMT_SMF_EXT.2.1 The TSF shall offer **wipe of protected data, wipe of sensitive data, alert the administrator and remove Enterprise applications** upon unenrollment and **no other triggers**.

Application note: Upon unenrollment, the MDM application get a callback and can take actions like wipe it's data, notify admin, uninstall applications (if whitelisted by Sony). Android for work has a container based solution that is wiped during unenrollment.

8.5 Class: Protection of the TSF (FPT)

8.5.1 Anti-Exploitation Services (FPT_AEX)

8.5.1.1 Address-Space Layout Randomization

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

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

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

8.5.1.2 Memory Page Permissions

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

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

8.5.1.3 Overflow Protection

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

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.

8.5.1.4 Domain Isolation

FPT_AEX_EXT.4 – Extended: Domain Isolation

FPT_AEX_EXT.4.1 The TSF shall protect itself from modification by untrusted subjects.

FPT_AEX_EXT.4.2 The TSF shall enforce isolation of address space between applications.

8.5.2 Key Storage (FPT_KST)

8.5.2.1 Plaintext Key Storage

FPT_KST_EXT.1 – Extended: Key Storage

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

Application note: The intention of this requirement is that the TOE will not write plaintext keying material to persistent storage. For the purposes of this requirement, plaintext keying material refers to authentication data, passwords, secret/private symmetric keys, private asymmetric keys, data used to derive keys, etc. These values must be stored encrypted.

8.5.2.2 No Key Transmission

FPT_KST_EXT.2 – Extended: No Key Transmission

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

8.5.2.3 No Plaintext Key Export

FPT_KST_EXT.3 – Extended: No Plaintext Key Export

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

8.5.2.4 Self-Test Notification (FPT_NOT)

FPT_NOT_EXT.1 – Extended: Self-Test Notification

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

8.5.3 Reliable Time Stamps (FPT_STM)

FPT_STM.1 – Reliable time stamps

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

8.5.4 TSF Functionality Testing (FPT_TST)

8.5.4.1 TSF Cryptographic Functionality Testing

FPT_TST_EXT.1 – Extended: TSF Cryptographic Functionality Testing

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.

8.5.4.2 TSF Integrity Testing

FPT_TST_EXT.2 – Extended: TSF Integrity Testing

FPT_TST_EXT.2.1 The TSF shall verify the integrity of the bootchain up through the Application Processor OS kernel, and **all binaries on RAM disk and all binaries on system partition** stored in mutable media prior to its execution through the use of **a digital signature using a hardware-protected asymmetric key**.

Application note: All binaries on RAM disk and code under /system is subject to integrity verification.

8.5.5 Trusted Update (FPT_TUD)

8.5.5.1 Trusted Update: TSF Version Query

FPT_TUD_EXT.1 – Extended: Trusted Update: TSF version query

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

FPT_TUD_EXT.1.2 The TSF shall provide authorized users the ability to query the current version of the hardware model of the device.

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

8.5.5.2 Trusted Update Verification

FPT_TUD_EXT.2 – Extended: Trusted Update Verification

FPT_TUD_EXT.2.1 The TSF shall verify software updates to the Application Processor system software and **communication processor software** using a digital signature by the manufacturer prior to installing those updates.

FPT_TUD_EXT.2.2 The TSF shall **never update** the TSF boot integrity **key**.

FPT_TUD_EXT.2.3 The TSF shall verify that the digital signature verification key used for TSF updates **matches a hardware-protected public key**.

FPT_TUD_EXT.2.4 The TSF shall verify mobile application software using a digital signature mechanism prior to installation.

FPT_TUD_EXT.2.6 The TSF shall not install code if the code signing certificate is deemed invalid.

8.6 Class: TOE Access (FTA)

8.6.1 Session Locking (FTA_SSL)

8.6.1.1 TSF- and User-initiated locked state

FTA_SSL_EXT.1 – Extended: TSF- and User-initiated locked state

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

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

FTA_SSL_EXT.1.3 The TSF shall, upon transitioning to the locked state, perform the following operations:

a) clearing or overwriting display devices, obscuring the previous contents;

b) **no other actions**.

8.6.2 Default TOE Access Banners

FTA_TAB.1 – Default TOE Access Banners

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

8.6.3 Wireless Network Access (FTA_WSE)

FTA_WSE_EXT.1 – Extended: Wireless Network Access

FTA_WSE_EXT.1.1 The TSF shall be able to attempt connections to wireless networks specified as acceptable networks as configured by the administrator in FMT_SMF_EXT.1.

8.7 Class: Trusted Path/Channels (FTP)

8.7.1 Trusted Channel Communication (FTP_ITC)

FTP_ITC_EXT.1 – Extended: Trusted Channel Communication

FTP_ITC_EXT.1.1 The TSF shall use 802.11-2012, 802.1X, and EAP-TLS and **TLS and HTTPS protocol** to provide a communication channel between itself and another trusted IT product that is logically distinct from other communication channels, provides assured identification of its end points, protects channel data from disclosure, and detects modification of the channel data.

FTP_ITC_EXT.1.2 The TSF shall permit the TSF to initiate communication via the trusted channel.

FTP_ITC_EXT.1.3 The TSF shall initiate communication via the trusted channel for wireless access point connections, administrative communication, configured enterprise connections, and **no other connections**.

9 Security Assurance Requirements

The security assurance requirements (SARs) are defined in the [MDFPP] and consist of the assurance components of EAL1 as defined in the CC Part 3, augmented by ASE_SPD.1 and extended by ALC_TSU_EXT.1. The ASE_SPD.1 is defined in CC Part 3 and ALC_TSU_EXT.1 is defined in [MDFPP].

The assurance components are taken unchanged from the [MDFPP] and are therefore not reproduced in the Security Target.

Assurance Class	**Assurance Components**
Security Target (ASE)	Conformance claims (ASE_CCL.1)
	Extended components definition (ASE_ECD.1)
	ST introduction (ASE_INT.1)
	Security objectives for the operational environment (ASE_OBJ.1)
	Stated security requirements (ASE_REQ.1)
	Security Problem Definition (ASE_SPD.1)
	TOE summary specification (ASE_TSS.1)
Development (ADV)	Basic functional specification (ADV_FSP.1)
Guidance documents (AGD)	Operational user guidance (AGD_OPE.1)
	Preparative procedures (AGD_PRE.1)
Life cycle support (ALC)	Labeling of the TOE (ALC_CMC.1)
	TOE CM coverage (ALC_CMS.1)
	Timely Security Updates (ALC_TSU_EXT)
Tests (ATE)	Independent testing – sample (ATE_IND.1)
Vulnerability assessment (AVA)	Vulnerability survey (AVA_VAN.1)

10 TOE Summary Specification

This chapter describes how the security functional requirements are implemented by the security functionality of the TOE. This chapter is structured in accordance with the security functional requirements in chapter 6 of this document, which follows the structure of the structure of the security functional requirements in [MDFPP].

10.1 Cryptographic Support

Cryptographic support is provided in different ways and on different levels by the TOE. On a low level the TOE provides a Trusted Execution Environment (TEE) by using the ARM TrustZone technology. It is a system on a chip (SoC) that provides the TOE with a hardware-backed, strong security services to the Android OS, to platform services and even to third-party apps. The TEE is an isolated environment that runs in parallel with the operating system.

The core components for cryptographic support are shown.

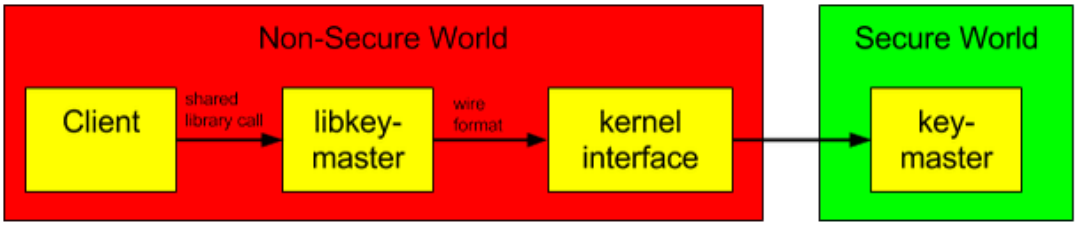

Figure 2, Cryptographic abstraction layer and access to the Keymaster

Access to the Keystore is provided by the Keymaster API, the Hardware Abstraction Layer (HAL). The Keystore provides the following categories of operations:

- Key generation of symmetric and asymmetric keys
- Import and export of asymmetric keys (no key wrapping)
- Import of raw symmetric keys (again, no wrapping)
- Asymmetric encryption and decryption with appropriate padding modes
- Asymmetric signing and verification with digesting and appropriate padding modes
- Symmetric encryption and decryption in appropriate modes, including an AEAD mode
- Generation and verification of symmetric message authentication codes
- Random number generation (not exposed as an API, but internally used for generation of keys, Initialization Vectors, random padding and other elements of secure protocols that require randomness)

Protocol elements, such as purpose, mode and padding, as well as access control constraints, are specified when keys are generated or imported and are permanently bound to the key, ensuring the key cannot be used in any other way.

The TOE is using FIPS 140-2 validated Qualcomm QTI Cryptographic Modules for hardware protection of keys and for cryptographic operations. The Xperia X is using Snapdragon 650 and Xperia X Performance is using Snapdragon 820.

10.1.1 Cryptographic key management

This section describes how keys are generated, derived, combined and destroyed. There are two major types of keys: Data encryption keys (DEKs) and key encryption keys (KEKs). Root encryption keys (REKs) are considered to be KEKs. DEKs are used to protect data (as in the data at rest (DAR) protection. KEKs are used to protect other keys – DEKs, other KEKs and other types of keys stored by the user or applications. The following diagram shows the TOE key hierarchy to help the reader to understand the key management and to demonstrate compliance with [MDFPP].

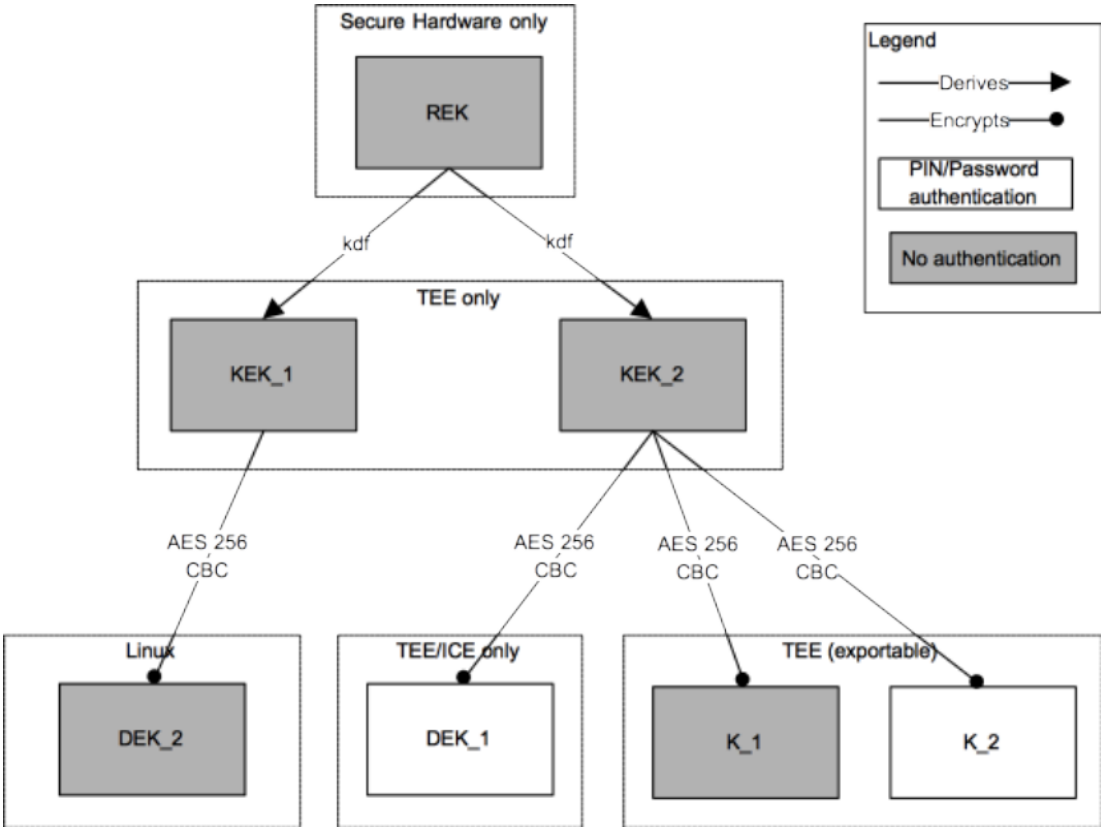

Figure 3 - Key Management Diagram for Full Disk Encryption

The keys in the diagram above are further described in the table below.

Name	Description	Origin	Memory	Destroyed persistently
REK	Root Encryption Key, which is generated in the production facility and written into HW when manufacturing the device.	External RNG (Factory)	Secure Hardware	Never
KEK_1	KEK for SD card file level encryption	Derived	TEE	N/A
KEK_2	KEK for keys in keystore	Derived	TEE	N/A
DEK_1	DEK for internal storage encryption, which is generated in TEE and at first start-up and is stored together with the user password. The user must submit the password during startup to allow TEE to decrypt the file system. It's a 256-bit key used for AES in XTS mode.	RNG (TEE)	TEE	• Factory reset • Flash without keeping user data
DEK_2	DEK for external SD Card storage encryption, which is created when user activates this functionality. It's a 256-bit key used for AES in CBC mode.	RNG (Linux)	Normal	• Factory reset • Flash without keeping user data
K_1	Category of keys for android applications. These keys are stored in HW backed keystore. See https://developer.android.com/training/articles/keystore.html#SecurityFeatures The application creates a key in keystore by calling API to generate a key in keystore or	RNG (TEE) or imported	TEE	• Application calls delete key • Uninstall application • Clear all credentials in settings menu

	by importing an existing key into the keystore.			• Factory reset • Complete flash without keeping user data
K_2	Same as K_1, but in this category the application has tagged the key to accessible only when the user has submitted the password. See https://developer.android.com/training/articles/keystore.html#UserAuthentication	RNG (TEE) or imported	TEE	• Application calls delete key • Uninstall application • Clear all credentials in settings menu • Factory reset • Complete flash without keeping user data

Table 2 - Description of the different types of keys used by the TOE

10.1.2 Cryptographic primitives

Cryptographic primitives provided by the TOE:

RSA

- 2048, 3072 and 4096-bit key support is provided
- Support for public exponent F4 (2^16+1)
- Required padding modes for RSA signing are:
  - No padding
  - RSASSA-PSS (KM_PAD_RSA_PSS)
  - RSASSA-PKCS1-v1_5 (KM_PAD_RSA_PKCS1_1_5_SIGN)
- Required digest modes for RSA signing are:
  - No digest
  - SHA-256
- Required padding modes for RSA encryption/decryption are:
  - Unpadded
  - RSAES-OAEP (KM_PAD_RSA_OAEP)
  - RSAES-PKCS1-v1_5 (KM_PAD_RSA_PKCS1_1_5_ENCRYPT)

ECDSA

- 224, 256, 384 and 521-bit key support are required, using the NIST P-224, P-256, P-384 and P-521 curves, respectively
- Required digest modes for ECDSA are:
  - No digest (deprecated, will be removed in the future)
  - SHA-256

AES

- 128 and 256-bit keys are provided
- CBC, CTR, ECB, GCM and XTS. The GCM implementation does not allow the use of tags smaller than 96 bits or nonce lengths other than 96 bits.
- Padding modes KM_PAD_NONE and KM_PAD_PKCS7 does not support CBC and ECB modes. With no padding, CBC or ECB mode encryption fails if the input isn't a multiple of the block size.
- HMAC SHA-256, with any key size up to at least 32 bytes.

SHA

- SHA-1
- SHA-224

- SHA-384
- SHA-512

10.1.3 Key access control

The TOE provides key access control for the use of keys. Although hardware-based keys that can never be extracted from the device, the unrestricted use of them may cause these keys to be exfiltrated.

Access controls are defined as an "authorization list" of tag/value pairs, where the authorization tags are 32-bit integers and the values are a variety of types. Some tags may be repeated to specify multiple values. Whether a tag may be repeated is specified in the documentation for the tag. When a key is created, the caller specifies an authorization list. The Keymaster implementation underlying Keystore will modify the list to specify some additional information, such as whether the key has rollback protection, and return a final authorization list, encoded into the returned key blob. Any attempt to use the key for any cryptographic operation fails if the final authorization list is modified.

10.1.4 Cryptographic Protocols

The TOE provides a number of cryptographic mechanisms and protocols that are used by the TSF and are available to applications. They are provided by Qualcomm with the QTI Crypto Engine ,QTI Inline Crypto Engine and QTI Software Library; The Xperia OpenSSL library; The BoringSSL library; and the Broadcom implementation.

1. QTI Crypto Engine and QTI SW lib
    a. used in Keystore/Keymaster module, Android Applications, kernel, TEE (trusted execution environment)
    b. accessed through Java API, native APIs, kernel
2. QTI Inline Crypto Engine
    a. Used in full disk encryption
    b. Accessed through kernel
3. Xperia OpenSSl library
    a. Used in VPN and SD card encryption
    b. Accessed through native APIs, kernel
4. BoringSSL library
    a. used in TLS, EAP-TLS, HTTPS, WPA2, Android Applications
    b. can be accessed through Java APIs, native APIs
5. Broadcom
    a. Used and accessed only by the Bluetooth component itself

Below is a summary of the cryptographic mechanisms and their implementation.

QTI Crypto Engine			
Algorithm	**Standard**	**NIST Cert#**	**SFR**
AES 128/256 CBC, ECB, CTR, XTS, CCM	FIPS 197 SP 800-38 A	Cert. 3526	FCS_COP.1(1)
SHS SHA-1/256	FIPS 180-4	Cert. 2908, 2909, 2930	FCS_COP.1(2)
HMAC SHA-1/256	FIPS 198-1 FIPS 180-4	Cert. 2254	FCS_COP.1(4)
DRBG 800-90A	SP 800-90A	Cert. 885	FCS_RBG_EXT.1
QTI Inline Crypto Engine			
Algorithm	**Standard**	**NIST Cert#**	**SFR**

AES 256 XTS	FIPS 197 SP 800-38 A	Cert. 3556, 3558	FCS_COP.1(1)
QTI Software Library			
Algorithm	**Standard**	**NIST Cert#**	**SFR**
RSA Sig(gen)/Sig(ver)/Key(gen)	FIPS 186-2 FIPS 186-4	No, covered during evaluator testing	FCS_CKM.1(1) FCS_COP.1(3)
ECDSA PKG/PKV/Sig(gen)/Sig(ver)	FIPS 186-4	No, covered during evaluator testing	FCS_CKM.1(1) FCS_COP.1(3)
Xperia OpenSSL Library			
Algorithm	**Standard**	**NIST Cert#**	**SFR**
AES 128/256 CBC, ECB, CTR, XTS, CCM, GCM	FIPS 197 SP 800-38 A	Cert. 3329	FCS_COP.1(1)
SHS SHA-1/224/256/384/512	FIPS 180-4	Cert. 2762	FCS_COP.1(2)
HMAC SHA-1/224/256/384/512	FIPS 198-1 FIPS 180-4	Cert. 2120	FCS_COP.1(4)
RSA Sig(gen)/Sig(ver)	FIPS 186-4	Cert. 1709	FCS_CKM.1(1) FCS_COP.1(3)
ECDSA PKG/PKV/Sig(gen)/Sig(ver)	FIPS 186-4	Cert. 658	FCS_CKM.1(1) FCS_COP.1(3)
CVL ECC CDH P-224/256/384/521 K-233/283/571 B-233/283/409/571	SP 800-56A	Cert. 485	FCS_CKM.1(1) FCS_COP.1(3)
BoringSSL Library			
Algorithm	**Standard**	**NIST Cert#**	**SFR**
AES 128/256 CBC, CCM, GCM, KW	FIPS 197 SP 800-38 A	Cert. 4103 Cert. 4105	FCS_COP.1(1)
SHS SHA-1/256/384/512	FIPS 180-4	Cert. 3376 Cert. 3377	FCS_COP.1(2)
HMAC SHA-1/256/384/512	FIPS 198-1 FIPS 180-4	Cert. 2680 Cert. 2681	FCS_COP.1(4)
RSA SIG(gen)/SIG(ver)/Key(gen)	FIPS 186-4	Cert. 2220 Cert. 2221	FCS_CKM.1(1) FCS_COP.1(3)
ECDSA PKG/PKV/Seg(gen)/Sig(ver)	FIPS 186-4	Cert. 929 Cert. 930	FCS_CKM.1(1) FCS_COP.1(3)
CVL ECC CDH P-224/256/384/521	SP 800-56A	Cert. 917 Cert. 918	FCS_CKM.1(1)
Broadcom			
Algorithm	**Standard**	**NIST Cert#**	**SFR**
AES 128 ECB, CBC, CTR	FIPS 197	Cert. 3678	FCS_CKM.1(2) FCS_COP.1(1)

Table 3 - Cryptographic Protocols and their Implementation

10.2 User Data Protection

The Android security model for separation and access control is based in part on the concept of application sandboxes. Each application runs in its own sandbox. Sandboxes are defined by the creation of a unique Linux UID for each application at time of installation. It also uses the mandatory access control (MAC) provided by Security-Enhanced Linux (SELinux) to further define the boundaries of the Android application sandbox. Mandatory access control (MAC) applies to all processes, even processes running with root/superuser privileges (a.k.a. Linux capabilities). SELinux enhances Android security by confining privileged processes and automating security policy creation. SELinux protects and confine system services, control access to application data and system logs, reduce the effects of malicious software, and protect users from potential flaws in code on mobile devices.

These mechanisms of Linux UID and SELinux MAC are used to enforce access control between processes and resources/data, including external interfaces.

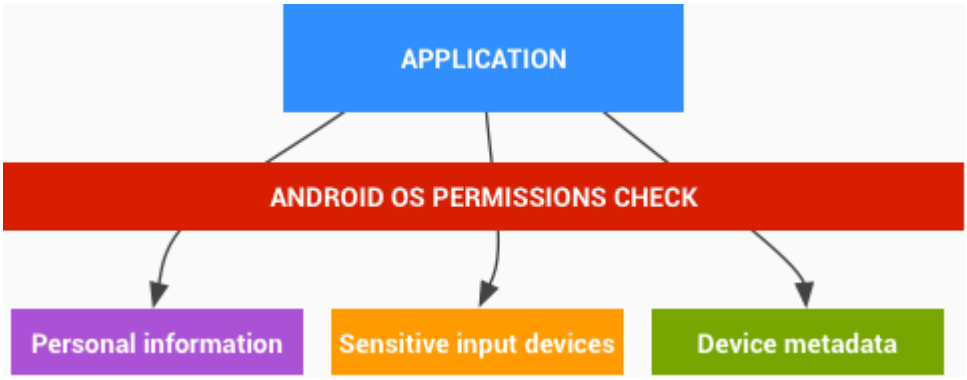

Figure 4 - Access to sensitive user data, input devices and metadata is available only through protected APIs that will perform permission checks

By default, an Android application can only access a limited range of system resources. The system manages Android application access to resources that, if used incorrectly or maliciously, could adversely impact the user experience, the network, or data on the devices.

These restrictions are implemented in one of three different ways. Some capabilities are restricted by an intentional lack of APIs to the sensitive functionality (e.g. there is no Android API for directly manipulating the SIM card). In some instances, the separation of roles provides a security measure, as with the per-application isolation of storage. In other instances, the sensitive APIs are intended for use by trusted applications and protected through the Android permission mechanism.

Access control to devices include access to the following sensitive devices and functions are the following:

- Camera functions (picture and microphone)
- Location data (GPS)
- Bluetooth functions
- Telephony functions
- SMS/MMS functions
- Network/data connections (access and device information)

These resources are only accessible through the operating system. To make use of the protected APIs on the device, an application must be granted the capabilities. Users grant permissions to apps while the app is running, not when they install the app. This approach streamlines the app install process, since the user does not need to grant permissions when they install or update the app. It also gives the user more control over the app's functionality; for example, a user could choose to give a camera app access to the camera but not to the device location. The user can revoke the permissions at any time, by going to the app's Settings screen. This is described in http://developer.android.com/training/permissions/requesting.html . Default permission are specified in the

class `android.Manifest.permission` and described in https://developer.android.com/reference/android/Manifest.permission.html.

Applications may (in addition) declare their own permissions for other applications to use. Such permissions are not listed in the above location.

When defining a permission a `protectionLevel` attribute tells the system how the user is to be informed of applications requiring the permission, or who is allowed to hold a permission. The following protection levels are available:

Definition of protection levels	
normal	The default value. A lower-risk permission that gives requesting applications 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).
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 may not automatically grant it to the requesting application. For example, any dangerous permissions requested by an application may be displayed to the user and require confirmation before proceeding, or some other approach may be taken to avoid the user automatically allowing the use of such facilities.
signature	A permission that the system grants only if the requesting application is signed with the same certificate as the application that declared the permission. If the certificates match, the system automatically grants the permission without notifying the user or asking for the user's explicit approval.
signatureOrSystem	A permission that the system grants only to applications that are in the Android system image *or* that are signed with the same certificate as the application that declared the permission. 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. The "signatureOrSystem" permission is used for certain special situations where multiple vendors have applications built into a system image and need to share specific features explicitly because they are being built together.

See http://developer.android.com/guide/topics/manifest/permission-element.html for more details on how this is implemented and used.

An example of a privilege with “normal” protection level is to give applications to `android.permission.ACCESS_NETWORK_STATE`, which would give applications access information about networks.

An example of a privilege with “dangerous” protection level is to give applications to `android.permission.READ_CALENDAR`, which would allow applications read access to the user's calendar data.

An example of a privilege with “signatureOrSystem” protection level is to give applications to `com.android.voicemail.permission.READ_VOICEMAIL`, which would allow applications read voicemails in the system.

Details on creating and using application specific permissions are described at https://developer.android.com/guide/topics/security/security.html.

There are some device capabilities, such as the ability to send SMS broadcast intents, that are not available to third-party applications, but that may be used by applications pre-installed by the OEM. These permissions use the signatureOrSystem permission.

10.3 Identification and Authentication

The user needs to successfully authenticate before access is given to applications and data. Authentication is performed during booting of the device as well as during unlocking screen-locking. The TOE maintains a counter of failed authentication attempts. After a certain (configurable) number of consecutive failed authentications the TSF will perform a factory reset, which means that it will wipe all data and application from the TOE. The counter is written into non-volatile memory and will remain over power-cycles and will only be reset upon a successful authentication. The number of failed authentications is administrator configurable but should at a minimum be set to 1 and maximum 100 in the evaluated configuration.

The TOE support Bluetooth version 4.0, which enables secure connections to other devices. The TOE supports faster data transfer with Enhanced Data Rate (EDR) and also has support not only LMP-pairing (aka PIN-code based), but also Secure Simple Pairing (SSP), enabling Public Key Infrastructure (PKI) encryption that protects against Man-in-the- middle (MITM) eavesdropping attacks and safeguards the integrity of the communication.

Before pairing with a Bluetooth device the TOE requires the explicit authorization from the user. Pairing with another device using LMP-pairing means that the TOE requires that the user either confirms 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). Secure Simple Pairing (SSP) may use the authentication mode “Just works”. As the name implies, this method just works, without PIN entry, but still requires user confirmation. This method is typically used by headsets with very limited IO capabilities, and is more secure than the fixed PIN mechanism for this limited set of devices uses for legacy pairing.

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).

The Android operating system provides applications the package java.security.cert Java API Class of methods validating certificates and certification paths (certificate chains establishing a trust chain from a certificate to a trust anchor). The API is documented in http://developer.android.com/reference/java/security/cert/package-summary.html

This package provides all the classes and all the interfaces needed to generate, administer and verify X.509 certificates. Functionality for parsing certificate, extracting information from them, validating and verifying the information they contains are provided. Exception are generated mainly for three reasons:

- if the certificate's encoding is broken (CertificateEncodingException)
- if the certificate's time stamp is not valid (CertificateExpiredException)
- or if the validation's path is false (CertPathValidatorException).

10.4 Security Management

The TOE supports the following types of users:

- Primary – The first user added to a device. The primary user cannot be removed except by factory reset. This user also has some special privileges and settings only it can set. The primary user is always running even when other users are in the foreground.
- Secondary – Any user added to the device other than the primary user. They can be removed by either themselves or the primary user and cannot impact other users on a device. Secondary users can run in the background and will continue to have network connectivity when they do.
- Guest – A guest user is a temporary secondary user with an explicit option to quick delete the guest user when its usefulness is over. There can be only one guest user at a time.
- Administrator – The administrator is acting remotely using a Mobile Device Management (MDM) system acting through an MDM agent on the TOE. Neither the MDM nor the MDM agent are parts of the TOE. So role of the administrator is describes in what the MDM API of the TOE can do. There are two different MDM APIs available with different functionality available: The Xperia in Business and the Android for Work.

See also https://source.android.com/devices/tech/admin/multi-user.html for more information on these roles. The user, in the context of this ST, is the primary, secondary and guest user.

There are certain management functions that are provided to the user and not the administrator, because there they are not supported by the administrator APIs. There are management functions for which the administrator may set a policy that restricts the user from performing that function. This is specified into Table 1 of the the FMT_SMF_EXT.1.1.

10.5 Protection of the TSF

10.5.1 Address Space Randomization and Protection against Buffer Overflow

The TOE's Android operating system provides Address Space Layout Randomizer (ASLR) and ensures that system applications and libraries are stored in random locations in the memory. The Android OS uses this randomization to protect the device against exploitation of the memory, and against malware getting installed on the device with the risk of corrupting the memory. ASLR prevents Return-Oriented Programming (ROP) attacks. Most binaries are randomized when executed because they are linked with the PIE (Position Independent Executable) flag. The linkers are randomized in the process address space. The Android OS has full stack, heap/brk, lib/mmap, linker, and executable ASLR. Randomization is done utilizing a random number provided by the Linux kernel to provide 8 unpredictable bits to the base address of any user-space memory mapping.

The TOE's Android operating system provides the ProPolice mechanisms to prevent stack buffer overruns (-fstack-protector) in addition to taking advantage of hardware-based hardware-based No eXecute (NX) to prevent code execution on the stack and heap. This protection is used for all TSF executable binaries and libraries. It is documented in https://source.android.com/security/enhancements/enhancements41.html

10.5.2 Separation and Sandboxing

The TOE’s Android platform takes advantage of the Linux user-based protection as a means of identifying and isolating application resources. The Android system assigns a unique user ID (UID) to each Android application and runs it as that user in a separate process. This approach is different from other operating systems (including the traditional Linux configuration), where multiple applications run with the same user permissions.

This sets up a kernel-level Application Sandbox. The kernel enforces security between applications and the system at the process level through standard Linux facilities, such as user and group IDs that are assigned to applications. By default, applications cannot interact with each other and applications have limited access to the operating system. If application A tries to do something malicious like read application B's data or dial the phone without permission (which is a separate application), then the operating system protects against this because application A does not have the appropriate user privileges. The sandbox is simple, auditable, and based on decades-old UNIX-style user separation of processes and file permissions.

Since the Application Sandbox is in the kernel, this security model extends to native code and to operating system applications. All of the software above the kernel in including operating system libraries, application framework, application runtime, and all applications run within the Application Sandbox. On some platforms, developers are constrained to a specific development framework, set of APIs, or language in order to enforce security. On Android, there are no restrictions on how an application can be written that are required to enforce security; in this respect, native code is just as secure as interpreted code.

In some operating systems, memory corruption errors generally lead to completely compromising the security of the device. This is not the case in Android due to all applications and their resources being sandboxed at the OS level. A memory corruption error will only allow arbitrary code execution in the context of that particular application, with the permissions established by the operating system.

10.5.3 Hardware Protection of Keystore

The keys in the Keystore are relying on the hardware backer TEE for its protection and access is controlled for the hardware-backed keys. Access controls are specified during key generation and enforced for the lifetime of the key. Keys can be restricted to be usable only after the user has authenticated, and only for specified purposes or with specified cryptographic parameters. For more information, please see the Implementer's Reference.

The Android Keystore API and the underlying Keymaster Hardware Abstraction Layer (HAL) provides a set of cryptographic primitives to allow the implementation of protocols using access-controlled, hardware-backed keys. In addition to expanding the range of cryptographic primitives, Keystore in Android provides the following:

- A usage control scheme to allow key usage to be limited, to mitigate the risk of security compromise due to misuse of keys

- An access control scheme to enable restriction of keys to specified users, clients, and a defined time range

HAL implementations must not perform any sensitive operations in user space, or even in kernel space. Sensitive operations are delegated to a secure processor reached through some kernel interface. Access to the Keystore is provided by the Keymaster API, the Hardware Abstraction Layer (HAL). Section 10.1 is showing how access is provided to the Keystore and what operations are available

The TOE is also using the FIPS 140-2 validated Qualcomm QTI Crypto Engine Core for protection of certain keys and for some cryptographic operations.

10.5.4 Full Disk Encryption

Android full disk encryption is based on dm-crypt, which is a kernel feature that works at the block device layer. The encrypted key is stored in the crypto metadata. Hardware backing is implemented by using Trusted Execution Environment's (TEE) signing capability. In order to make the key resilient against off-box attacks, this algorithm by signing the resultant key with a stored TEE key. The resultant signature is then turned into an appropriate length key by one more application of scrypt. This key is then used to encrypt and decrypt the master key. The TOE does not provide any way to export plaintext DEKs or KEKs (including all keys stored in the keystore) as the TOE chains all KEKs to the HEK/REK. The AES in XTS mode is used for the internal storage. For encryption of SD cards (external storage) AES in CBC mode is used.

10.5.5 Secure Boot and Self-test

The Android operating system of the TOE supports verified boot through the optional device-mapper-verity (dm-verity) kernel feature, which provides transparent integrity checking of block devices. dm-verity helps prevent persistent rootkits that can hold onto root privileges and compromise devices. This feature helps Android be sure when booting a device it is in the same state as when it was last used. The dm-verity feature lets you look at a block device, the underlying storage layer of the file system, and determine if it matches its expected configuration. It does this using a cryptographic hash tree. For every block (typically 4k), there is a SHA256 hash.

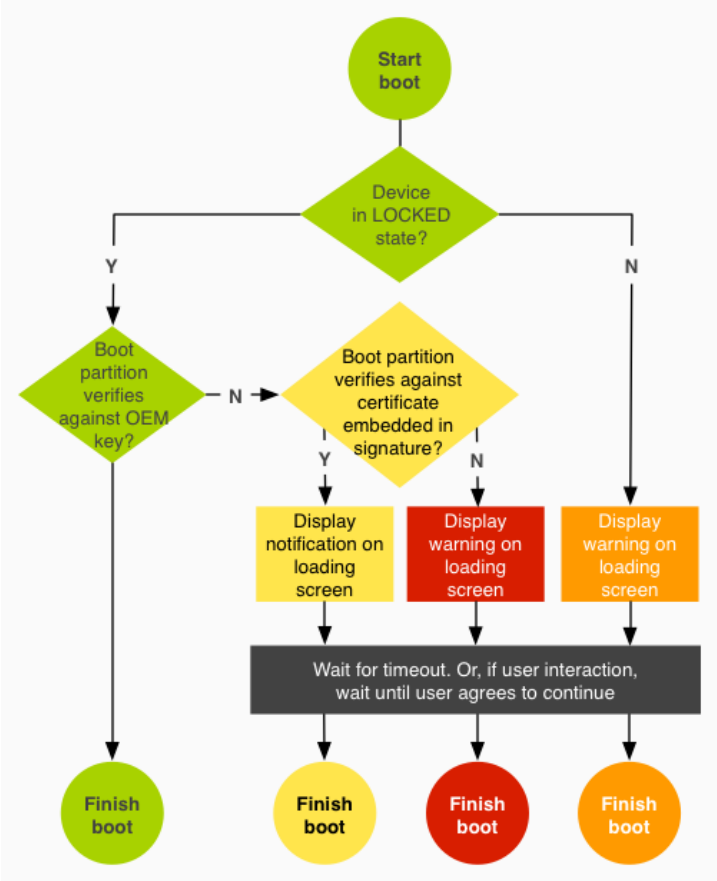

Figure 5 - Verified boot flow

The boot chain consists of three major steps. First the boot loader is verified using the hardware root of trust. Then the boot loader verifies the boot partition. Then the kernel verifies the system partition. This is described in Android documentation http://source.android.com/security/verifiedboot/verified-boot.html.

When the self-test of the verified boot fails, the TOE transitions to a non-operational mode. The user may attempt to power-cycle the TOE to see if the failure condition persists, and if it does persist, the user may attempt to boot to the recovery mode/kernel to wipe data and perform a factory reset in order to recover the device.

The TOE requires a reliable time for the TLS certificate validation and key store applications. These TOE components obtain time from the TOE using system API calls. An application cannot modify the system time, as mobile applications need the android “SET_TIME” permission to do so. Likewise, only a process with root privileges can directly modify the system time using system-level APIs. The TOE uses the Cellular Carrier time (obtained through the Carrier’s network time server) as a trusted source; however, the user can also manually set the time through the TOE’s user interface.

Time: check if NTP or carriers network time is used

- User can enable/disable to set automatic network time (NITZ and NTP) to system clock via settings menu. Default enabled.
- NITZ from carrier network is used when it’s available
- If the phone has not received a NITZ update within 24 hours it fetches NTP time

The time functions is used as a security functions for:

- Validation of application signature
- Certificate validation such as TLS, VPN, Secure Mail

Note: The system clock is not used in certificate validation during secure boot. Secure boot must work even when time is lost, data is lost and without access to a network. For example fresh out of factory or after

completely lost all battery including backup battery. Secure boot don't trust anything that comes from Linux side.

10.5.6 Software Firmware Control and Updates

The user can obtain the version of the TOE software/firmware (Model number Android version, Baseband version, Kernel version and Build number). This is done using the menus of the user interface "System Settings->Security->About phone".

Trusted updates are performed using Firmware Over the Air (FOTA) functionality as illustrated and described below.

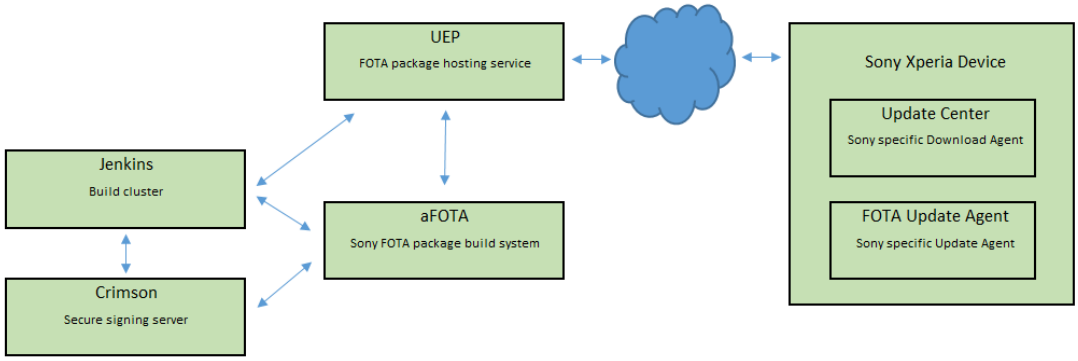

Figure 6 - The FOTA trusted update process

The aFOTA server is responsible for creation of FOTA update packages for system firmware for Sony Xperia devices. FOTA package is published via UEP for Update Center to download via external networks. After download Update Center reboots into software update mode, where Sony FOTA Update Agent (UA) is able to update all necessary partitions to update system software. This could include modem CPU SW, depending on platform configuration. Partitions is updated as a binary pattern, included any image signatures and DM verity hashes. FOTA update package might include instruction to update values in Trim Area. FOTA package is securely signed in backend-systems at Sony. A FOTA package is only applied if the signature of the FOTA package is valid. The TOE will also check the new image to ensure that it is not older than the current image, and if so, the TOE not perform the update the TOE software. Internal Firmware for NFC and similar chipsets is updated after the new system software is started. UA do not support update of any security related keys. All checks of digital signatures are done via Sony Security library.

The Sony FOTA Update Agent (UA) is executed from both normal Android Kernel and the FOTA kernel. The FOTA UA executable is built-in in the kernel romfs. Both Android kernel and FOTA kernel is verified by boot loader. The UA access to the platform is restricted by SE Linux rules and other protective mechanisms. The Sony FOTA kernel is using the standard 'recovery' domain as Google recovery kernel. UA do not have access to any security related keys. UA have an internal state machine to recover for power loss or system crash to be able to continue from previous state. When state is recovered the integrity and signature of FOTA package is always checked again. Images can be completely overwritten or updated via a delta. Before FOTA update starts the current system software is matched against the FOTA package information making sure it matches the device. Applications is not installed separately by UA. The Sony Update Center (UC) may download and install Applications. This is done using the Android Package Installer.

ALC_TSU_EXT: Sony Mobile works with suppliers and other partners to ensure that products and services receive security patches. Everyone is encouraged to report security and privacy issues concerning Sony Mobile products and services by emailing softwaresecurity@sonymobile.com (more details can be found on https://www.sonymobile.com/global-en/software-security/). Sony Mobile's security and privacy team will work together with the submitter, involving partners if necessary, to resolve issues in our products and services. Sony Mobile aims to deploy available patches as soon as possible. Xperia smartphones are kept updated within normal and regular software maintenance, both directly to open-market devices and via our carrier partners, so timings can vary by region and/or operator.

10.6 TOE Access

An unlocked TOE will transit into a locked state after a certain time of user inactivity. The time interval is configurable by the user and administrator (from 15 seconds to 10 minutes). The user or administrator may also explicitly lock the TOE by pressing the on/off button or by the administrator using an MDM system. In a locked

state the TOE will be able to display time and date, notifications, missed calls, text messages, signal strength and battery life. The level of information shown on the locked device is configurable by the user and the administrator. The TOE can be configured to display a user-specified message on the locked screen.

The TOE can be configured to display a user defined lock message when the device is locked. Such a message can also be specified by the administrator over the MDM interface when the device is locked.

The TOE can connect to wireless networks (WiFi), but is restricted only to connect to those networks that selected by the user or administrator using an MDM. The list of networks is specified by the SSID for these networks. The administrator (using the TOE's MDM APIs) can disable USB mass storage mode.

10.7 Trusted Path and Trusted Channels

The TOE provide mutually authenticated and encrypted channels using 802.11-2012, 802.1X, EAP-TLS, TLS to provide a communication channel between itself and another trusted IT product.

10.8 Mapping to the Security Functional Requirements

The following table provides a mapping of the SFRs defined in chapter 5 of this ST to the functions implemented by the TOE, referring to the previous sections of the TOE Summary Specification where additional information is required.

SFR	Comment
FCS_CKM.1(1)	RSA and ECC key generation are functions of the key master module of the TOE. The module can generate RSA key pairs with modulus size of 2048 that meets the following: FIPS PUB 186-4, “Digital Signature Standard (DSS)”. ECC key pairs can be generated for NIST curves P-256, P-384 and P-521 that meet the following: FIPS PUB 186-4.
FCS_CKM.1(2)	The PRF-384 key generation is implemented as defined in IEEE 802.11-2012, "Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications", section 11.6.1.2. It is implemented as part of the WPA implementation and is used for the key generation of AES keys when the CCMP cipher (defined in section 11.4.3.1 of IEEE 802.11-2012 is used. The Broadcom chipset BCM4359 used by the TOE is NIST validated with NIST ID 3678. The Random Bit Generator used is the one of the Qualcomm platform. Compliance to the requirements of FCS_RBG_EXT.1 are addressed in the Entropy Assessment Report from Qualcomm. The TOE uses PRF384 for WPA2 derivation of 128-bit AES Temporal Key and also employs its OpenSSL AES-256 DRBG when generating random values use in the EAP-TLS and 802.11 4-way handshake.
FCS_CKM.2.1(1)	The RSA and ECC based key establishment are functions of the key master module. The following list of key establishments schemes are supported: The RSA-based key establishment schemes meets the NIST Special Publication 800-56B, “Recommendation for Pair-Wise Key Establishment Schemes Using Integer Factorization Cryptography”; The Elliptic curve-based key establishment schemes meets the NIST Special Publication 800-56A, “Recommendation for Pair-Wise Key Establishment Schemes Using Discrete Logarithm Cryptography”. When setting up a TLS session, the TOE acts as a receiver, any decryption errors which occur during the key establishment are presented to the user at a highly abstracted level, such as a failure to connect.
FCS_CKM.2.1(2)	The WLAN AES Key Wrap in an EAPOL-Key frame is implemented as defined in NIST SP 800-38F and IEEE 802.11-2012, "Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications", sections 11.6.2 and 11.6.3. Unwrap of the GTK is using AES Key Wrap in an EAPOL-Key frame. Upon receiving an EAPOL frame, the TOE will check the frame to ensure that it is a proper EAPOL message and then decrypt the GTK using the KEK, thus ensuring that it does not expose the Group Temporal Key (GTK).
FCS_CKM_EXT.1	The TOE supports a device unique Root Encryption Key (REK) that is generated and installed in the production environment. The availability of a trusted execution environment (TEE) in a system on a chip (SoC) provides the TOE with a hardware-backed, strong security services to the Android OS, to platform services, and even to third-party apps. The TOE uses this hardware protected keystore for storing the REK. It does not provide any way to export plaintext REK. A FIPS 140-2 validated security module (NIST ID 2614) ensures that the key is protection from being exported or derived. The REK is deterministically derived from the Primary Hardware Key generated and during the Qualcomm chip production. The TOE generates the fuse value during manufacturing using its hardware DRBG. It is imported and burnt into the Qualcomm chip during chip manufacturing. The TEE does not allow direct access to the

	REK but provides access to a derived HEK (Hardware Encryption Key), which is derived from the REK through a KDF function for encryption and decryption.
FCS_CKM_EXT.2	All the DEKs are generated using the RBG of the FIPS validated hardware module and they are AES keys with 256 bit key size.
FCS_CKM_EXT.3	The TOE can generate the KEKs in two different ways. It can be generated using a password based key derivation function (PBKDF) or using a RGB as defined in FCS_RBG_EXT.1. The KEKs are always AES 256-bit keys. Applications such as the BackupManagerService and other none TOE applications may use a KEK generated by PBKDF while most applications are using the KEK of the TrustZone and Keystore. The PBKDF functionality uses 10000 Hash rounds and a 512-bit salt stored in same file as password hash. The key size is 256-bits. Note: none of the TSFs rely on the KEKs generated using PBKDF. The filesystem encryption uses scrypt, which generates a KEK used for software based filesystem encryption. However this KEK is unused since TOE is configured for HW based filesystem encryption where key comes from hardware RNG. So the number of hash rounds is no applicable. Using scrypt parameters used are N=15, r=3, P=1, and key size 256-bits. For more information see FCS_RBG_EXT.1.
FCS_CKM_EXT.4	Cryptographic keys that are no longer needed are destroyed. The exceptions to this are public keys (that protect the boot chain and software updates) and the REK, which are never cleared. However, these are stored in the secure hardware module preventing the key to leave that module. There are several different ways keys are destroyed: • Encrypted keys are “destroyed” by destroying the encryption key with which they were encrypted. • Keys stored in volatile memory (RAM) are destroyed by a single direct overwrite of zeros, following by a read-verify. • Keys stored in non-volatile EEPROM are destroyed by a single direct overwrite consisting of a pseudo random pattern using the RBG function (FCS_RBG_EXT.1) followed a read-verify. • Keys stored in non-volatile flash memory are destroyed by a single direct overwrite consisting of zeros followed by a read-verify. • For non-volatile memory other than EEPROM and flash, the destruction shall be executed by overwriting three or more times with a random pattern that is changed before each write. See *Table 2* in section 10.1.1 for a description of the different types of keys.
FCS_CKM_EXT.5	The TOE will perform a wipe of keys (and user data) that is triggered either by a certain number of failed authentication attempts (FIA_AFL_EXT.1) or by the user (or administrator) actively doing a wipe (FMT_SMF_EXT.1). For volatile memory, the wipe is done by a single direct overwrite consisting of zeroes following by a read-verify. For non-volatile EEPROM, the destruction is executed by a single direct overwrite consisting of a pseudo random pattern using the TSF’s RBG, followed a read-verify. For non-volatile flash memory, the destruction shall be executed by a single direct overwrite consisting of zeros followed by a read-verify. For non-volatile memory other than EEPROM and flash, the destruction shall be executed by overwriting three or more times with

	a random pattern that is changed before each write. The TOE will perform a power cycle at the end of the wipe operation.
FCS_CKM_EXT.6	The salt generated is 256 bits generated in accordance with NIST Special Publication 800-90A using Hash_DRBG (any). Salt is used for the password protection of encrypted file systems. Salt is available from /dev/random which obtain its random source from the hardware secure module.
FCS_CKM_EXT.7	The ECDH public/private key pairs are generated every time a pairing is done. Initially, each device generates its own Elliptic Curve Diffie-Hellman (ECDH) public-private key pair. This key pair needs to be generated only once per device and may be computed in advance of pairing. A device may, at any time, choose to discard its public-private key pair and generate a new one, although there is not a requirement to do so. In particular the implementation does not permit the use of static ECDH key pairs.
FCS_COP.1(1)	The TOE performs encryption/decryption in accordance with a specified cryptographic algorithms • AES-CBC (as defined in FIPS PUB 197, and NIST SP 800-38A) mode, • AES-CCMP (as defined in FIPS PUB 197, NIST SP 800-38C and IEEE 802.11-2012), and • AES Key Wrap (KW) (as defined in NIST SP 800-38F), • AES-GCM (as defined in NIST SP 800-38D), • AES-XTS (as defined in NIST SP 800-38E) mode Using the cryptographic key sizes 128-bit and 256-bit.
FCS_COP.1(2)	The TOE performs cryptographic hashing in accordance with specified cryptographic algorithm SHA-1 and SHA-256 and message digest sizes 160 and 256 bits that meets: FIPS Pub 180-4. The HMAC SHA-1 and SHA-256 is used as part of FCS_COP.1(4).
FCS_COP.1(3)	The TOE performs cryptographic signature generation and verification in accordance with: • 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-256, P-384 and P-521 that meet the following: FIPS PUB 186-4, “Digital Signature Standard (DSS)”, Section 5.
FCS_COP.1(4)	The TOE performs keyed-hash message authentication in accordance with HMAC-SHA-1 and HMAC-SHA-256, and cryptographic key sizes 160 and 256 message digest sizes (MAC length) 160 and 256 bits that meet the following: FIPS Pub 198-1, "The Keyed-Hash Message Authentication Code, and FIPS Pub 180-4, “Secure Hash Standard. The block sizes are 512 for SHA-1 and SHA-256..
FCS_COP.1(5)	The TOE performs password-based key derivation functions (PBKDF) in accordance with PBKDF2 using the algorithms HMAC-SHA-1 and SHA-256 with 32768 iterations and output cryptographic key sizes 256 that meet the following: NIST SP 800-132. While PBKDF2 is available, it is not used by any TSF, as described under FCS_CKM_EXT.3 The PBKDF provided by the TOE is described here: http://android-developers.blogspot.se/2013/02/using- cryptography-to-store-credentials.html
FCS_HTTPS_EXT.1	The TOE implements HTTPS protocol compliant with RFC 2818 using TLS (as described in FCS_TLSC_EXT.2).

FCS_IV_EXT.1	The TOE generates initialization vectors in accordance with [MDPPF] Table 14: References and IV Requirements for NIST-approved Cipher Modes. The TOE generates IVs within the hardware module for data storage encryption and for key storage encryption. The TOE uses AES-XTS mode for data encryption and AES-CBC for key storage, both using 256 bit keys.
FCS_RBG_EXT.1	The TOE provides a deterministic random bit generator as part of the secure hardware module. The hardware random bit generation is not exposed as an API, but internally used for generation of keys, Initialization Vectors, random padding and other elements of secure protocols that require randomness. It generates entropy in accordance with NIST Special Publication 800-90A using Hash_DRBG (any) with 256 bits security strength (in according to NIST SP 800-57).
FCS_SRV_EXT.1	The TOE provides cryptographic services to applications as a dynamically-loadable library used by the secure hardware module performing the keystore services.
FCS_STG_EXT.1	The TOE is using the FIPS 140-2 validated Qualcomm QTI Crypto Engine Core as a trusted execution environment for the protection of keys and for cryptographic operations. Access to this keystore is provided by the Keymaster API, the Hardware Abstraction Layer (HAL).
FCS_TLSC_EXT.1	The TOE implements TLS 1.0 and TLS 1.1 (RFC 4346), and TLS 1.2 (RFC 5246) 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 The TSF verifies that the server certificate presented for EAP-TLS (as part of WPA2) chains to one of the specified CAs. The server certificate is validated using the Android package java.security.cert methods for validating certificates and certification paths (certificate chains establishing a trust chain from a certificate to a trust anchor). It is documented in http://developer.android.com/reference/java/security/cert/package-summary.html.
FCS_TLSC_EXT.2	The TOE implements TLS 1.2 (RFC 5246) 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_DHE_RSA_WITH_AES_128_CBC_SHA as defined in RFC 5246 • TLS_DHE_RSA_WITH_AES_256_CBC_SHA as defined in RFC 5246 • TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA as defined in RFC 4492 • TLS_ECDHE_RSA_WITH_AES_256_CBC_SHA as defined in RFC 4492 • TLS_ECDHE_ECDSA_WITH_AES_128_CBC_SHA as defined in RFC 4492 • TLS_ECDHE_ECDSA_WITH_AES_256_CBC_SHA as defined in RFC 4492 • 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

	The TOE verifies that the presented identifier matches the reference identifier according to RFC 6125. The TOE supports mutual authentication using X.509v3 certificates. The TOE provides mobile app the use of TLS versions 1.0, 1.1, and 1.2, including all ciphersuites described above. The TOE supports Common Name (CN) and Subject Alternative Name (SAN) (DNS and IP address) as reference identifiers. The TOE supports client (mutual) authentication. The Android core libraries of the TOE supports certificate pinning. Pinned domains will receive a certificate validation failure if the certificate does not chain to a set of expected certificates. This protects against possible compromise of Certificate Authorities. The TOE presents the signature_algorithms extension in the Client Hello with the supported_signature_algorithms value containing the following hash algorithms: SHA256, SHA384, SHA512 and no other hash algorithms. They are supported by default. The TOE supports by default the evaluated elliptic NIST curves secp256r1 and secp384r1.
FDP_ACF_EXT.1	As described in section 10.2 User Data Protection, an Android application can only access a limited range of system resources. These restrictions are implemented in one of three different ways. Some capabilities are restricted by an intentional lack of APIs to the sensitive functionality (e.g. there is no Android API for directly manipulating the SIM card). In some instances, the separation of roles provides a security measure, as with the per-application isolation of storage. In other instances, the sensitive APIs are intended for use by trusted applications and protected through the Android permission mechanism. For more detailed information see section 10.2.
FDP_DAR_EXT.1	As described in section 3.4.1.2 and 10.5.4, the TOE offer full encryption for all user data in the internal storage, as well as any external SD card. With the exception of the boot image of the TOE that is integrity protected as part of security boot, TSF data is signed or encrypted and user data is subject to data encryption.
FDP_IFC_EXT.1	The TOE provides an IPsec VPN Client that ensures that all traffic (other than explicitly excluded) to establish the VPN connection flows through the VPN. The TOE routes all packets through the iptables firewall (default module in Android) when the VPN is active and there is no difference in the routing of IP traffic when using different supported baseband protocols. The VPN support has two parts: 1. Part 1 is a IPsec stack in Linux kernel. The IPsec stack integrated in the Linux kernel since 2.6 (NETKEY) was originally based on the KAME stack (at least in regards to the API). The source code is part of the kernel repository, where the main components are found in the net/xfrm folder, including the implementation of the Netlink/XFRM configuration interface. 2. The 2nd part is a user-space client that will configure the keying process. In Android case we use automatic keying provided by a user space IKE daemon racoon (ipsec-tools 0.7.3 http://ipsec-tools.sourceforge.net/), that way we do not have to manually install SAs and policies and get ephemeral encryption/integrity keys established via Diffie-Hellman during IKE. For VPN encryption the TOE uses the OpenSSL library provided as part of Android. Android supports network security using VPN: 1. Always-on VPN – The VPN can be configured so that applications don’t have access to the network until a VPN connection is

	established, which prevents applications from sending data across other networks. 2. Per User VPN – On multiuser devices, VPNs are applied per Android user, so all network traffic is routed through a VPN without affecting other users on the device. 3. Per Profile VPN – VPNs are applied per Work Profile, which allows an IT administrator to ensure that only their enterprise network traffic goes through the enterprise- Work Profile VPN—not the user's personal network traffic. 4. Per Application VPN – Android provides support to facilitate VPN connections on allowed applications or prevents VPN connections on disallowed applications.
FDP_STG_EXT.1	The TOE has a build in Trusted Anchor Database containing the built-in certificates. The user can disable certificates and add additional certificates using the user menu. The MDM can also disable or add certificates using the MDM API. The built-in certificates are protected, as they are part of the TSF's read only system partition, while the TOE protects user-loaded certificates by storing them with appropriate permissions to prevent modification by mobile applications. The TOE also stores the user-loaded certificates in the user's Keystore along with the user data, while that is hardware protected.
FDP_UPC_EXT.1	The TOE provides the following protocols for user data transfer: TLS, HTTPS, Bluetooth DR/EDR and Bluetooth LE. The following APIs are available for: • TLS – javax.net.ssl.SSLContext: http://developer.android.com/reference/javax/net/ssl/SSLContext.html • HTTPS – javax.net.ssl.HttpsURLConnection: http://developer.android.com/reference/javax/net/ssl/HttpsURLConnection.html • Bluetooth – android.bluetooth: http://developer.android.com/reference/android/bluetooth/package-summary.html
FIA_AFL_EXT.1	As described in section 10.3 Identification and Authentication, the TOE maintains a counter of failed authentication attempts. After a certain (configurable to be between 1-100) number of consecutive failed authentications the TSF will perform a factory reset, which means that it will wipe all data and application from the TOE. The counter is written into non-volatile memory and will remain over power-cycles and will only be reset upon a successful authentication.
FIA_BLT_EXT.1	As described in section 10.3 Identification and Authentication, the Bluetooth different types of authentication are supported, and how user confirmation is necessary.
FIA_PAE_EXT.1	The TOE conforms to IEEE Standard 802.1X for a Port Access Entity (PAE) in the "Supplicant" role for Wireless connections.
FIA_PMG_EXT.1	The TOE can be configured for a password policy that supports passwords consisting of any combination of upper and lower case letters, numbers, and any special character available on the keyboard. The minimum size of such passwords can also be defined via a Configuration Profile.
FIA_TRT_EXT.1	The TOE enforces a delay of at least 5 seconds between incorrect authentication attempts.
FIA_UAU.7	The TOE will obscure passwords entered by the user. By default, display the most recently entered character of the password briefly or until the

	user enters the next character in the password is entered. Characters entered are displayed as a dot when obscured.
FIA_UAU_EXT.1	The TOE provides encryption of data and keys stored on the device to prevent unauthorized access to encrypted data. Access to encrypted data is only provided once the user has been successfully authenticated.
FIA_UAU_EXT.2	The TOE allows certain actions to be performed when in a locked state. The allowed actions are identified in the SFR FIA_UAU_EXT.2.
FIA_UAU_EXT.3	The TOE requires the user to authenticate in order to unlock and to be able to authenticate keystore, change password, change unlock method, allow boot-loader to be unlocked, factory reset from settings menu.
FIA_X509_EXT.1	The TOE validates X.509 certificates using the certificate path validation algorithm defined in RFC 5280. Certificate paths must terminate in the Trust Anchor Database for a certificate to be regarded as trusted. The extendedKeyUsage field in the certificate is validated to ensure that the key is used in accordance with its usage specification. CA certificates are only accepted if the basicConstraints extension exists and the CA flag in this extension is set. The validation also verified include checking the revocation status of the certificate using a Certificate Revocation List (CRL) as specified in RFC 5759.
FIA_X509_EXT.2	The TOE uses X.509v3 certificates as defined by RFC 5280 to support authentication for EAP-TLS exchanges and IPsec, TLS, HTTPS and code signing for system software updates, code signing for mobile applications, code signing for integrity verification. Please, see the guidance for the management of certificates. The connection will be rejected when, during a connection establishment, a certificate validity check fails, i.e. certificate cannot be verified (including validity, certification path, and revocation through CRLs.
FIA_X509_EXT.3	The TOE's provides apps the `java.security.cert` Java API class of methods for validating certificates and certification paths. More information is available here: http://developer.android.com/reference/java/security/cert/package-summary.html
FMT_MOF_EXT.1	In FMT_SMF_EXT.1 indicates the management functions and the entity (user or administrator) that can perform the function. A TOE that is enrolled grants the MDM exclusive management rights to certain functions. This restricts the user from performing certain management functions. If a user fails to comply with the policies, the TOE will be unenrolled.
FMT_SMF_EXT.1	In FMT_SMF_EXT.1 indicates the management functions that can be performed divided into the following categories: • Management functions available (in general) are listed under FMT_SMF_EXT.1 • Management functions that are provided to the user and not the administrator are listed under "User", indicated in [MDFPP] as FMT_MOF_EXT.1.1 • Management functions provided to the administrator (using an MDM agent API) are listed under "Administrator" • Management functions for which the administrator may set a policy that restricts the user from performing that function are listed in "MDM policy", indicated in [MDFPP] as FMT_MOF_EXT.1.2.
FMT_SMF_EXT.2	When unenrolled, the TOE removes all MDM policies. The TOE also provides the capability to the MDM Agent to wipe all protected data and to remove any installed enterprise applications.

FPT_AEX_EXT.1	The TOE provides address space randomization as part of the Linux kernel. This is described in section 10.5 Protection of the TSF.
FPT_AEX_EXT.2	The TOE relies on the memory management of the Linux kernel that utilizes the MMU to enforce 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. Furthermore, the TOE relies on the hardware-based No eXecute (NX) to prevent code execution on the stack and heap.
FPT_AEX_EXT.3	As described in section 10.5 Protection of the TSF the TOE's provides the ProPolice mechanisms to prevent stack buffer overruns (-fstack-protector) in addition to taking advantage of hardware-based hardware-based No eXecute (NX) to prevent code execution on the stack and heap.
FPT_AEX_EXT.4	The TOE provides a number of different mechanisms to protect the TOE from unauthorized modifications. In addition to the detection of unauthorized modifications provided by secure boot, the TOE also relies on process separation and Linux mechanisms, such as assignment of unique UID and sandboxing. This is described in section 10.5 Protection of the TSF. The TOE protects itself from USSD and MMI code by preventing its execution when in a locked state. The TOE does not support any auxiliary boot modes.
FPT_KST_EXT.1	Outside of the secure hardware module the TOE does not store any plain text material in non-volatile memory. It also provides full disk encryption of all user and TSF data and all keys are in addition wrapped with a KEK. As part of the boot process the TOE must decrypt the encrypted partition as well as the KEK in order to access the encryption keys. However, certain TSF data, such as non-confidential parameters are not encrypted, since it would not add any additional security by encrypting them, for example configurations in trim area.
FPT_KST_EXT.2	The TOE uses a hardware protected keystore for storing the keys. It does not provide any way to export plaintext keys. The FIPS 140-2 validated security module (NIST ID 2423) ensures that keys are protected from being exported. Access to the Keystore is provided by the Keymaster API, the Hardware Abstraction Layer (HAL). The key material is handled within the keystore only and thereby prevents any unencrypted key material to be exported. See section 10.1 Cryptographic Support.
FPT_KST_EXT.3	The TOE does not provide any way to export plaintext DEKs or KEKs (including all keys stored in the keystore) as the TOE chains all KEKs to the HEK/REK as described in section 10.5.
FPT_NOT_EXT.1	The TOE provides self-test that is performed during boot. When the verified boot fails, the TOE transitions to a non-operational mode. The user may attempt to power-cycle the TOE to see if the failure condition persists, and if it does persist, the user may attempt to boot to the recovery mode/kernel to wipe data and perform a factory reset in order to recover the device.
FPT_STM.1	TOE uses time from system clock, which can use time from NITZ or NTP depending on user setting. Applications also have a direct API to get NTP time directly via SNTP Client API. The system clock is used for password lifetime check, user inactivity locking and certificate validations. Note: FOTA does not use time when validating update images. For more information see section 10.5
FPT_TST_EXT.1	The TOE uses a FIPS 104-2 validated (NIST ID 2614) hardware module for cryptographic operations. It runs a suite of self-tests during initial start-up as required by FIPS 140-2 for all cryptographic functionality.

FPT_TST_EXT.2	Each step of the boot-up and the software update processes contains components that are cryptographically signed to ensure integrity of the TSF components, as a basis for trusting the security functionality it provides. Details on the boot mechanism is provided in section 10.5.5
FPT_TUD_EXT.1	The TOE provides users the ability to query the current version of the TOE firmware/software; the current version of the hardware model of the device; and the current version of installed mobile applications, as described in section 10.5.
FPT_TUD_EXT.2	The TOE can be updated using the Firmware over the Air (FOTA) as described in more detail in section 10.5.6. Each step of the software update processes contains components that are cryptographically signed by Sony to ensure integrity of the TOE update components. Also all application must be digitally signed and the signatures are validated before the TOE will install the application.
FTA_SSL_EXT.1	The TOE transits into a locked state either after a certain amount of user inactivity or directly initiated by the user or by the MDM. When going into a locked state the display is overwritten. This is further described in section 10.6 TOE Access.
FTA_TAB.1	The TOE displays an advisory warning message regarding unauthorized use of the TOE.
FTA_WSE_EXT.1	The TOE allows the user and the administrator (using an MDM) to specify the wireless networks (SSIDs) to which the TSF may connect.
FTP_ITC_EXT.1	The TOE trusted communication channels between itself and other trusted IT products through the use of 802.11-2012, 802.1X, EAP-TLS, TLS and HTTPS protocols. It is always the TOE (or any of its application) that initiates such a communication via a trusted channel. The TOE provides its application access to trusted channels using an API provided by the TOE.

11 Abbreviations, Terminology and References

11.1 Abbreviations and Terminology

AES Advanced Encryption Standard

API Application Programming Interface

DEK Data Encryption Key

EDR Enhanced Data Rate

An optional part of the Bluetooth specification that provides a faster data rate (speed) and possibly improved battery life. Both devices need to support EDR, in which case EDR is used automatically.

FEK File Encryption Key

FOTA Firmware Over The Air

A feature where users can update their handset firmware over the carrier network. It removes the need of special cables, computers or third-party programs.

GPS Global Positioning System

Global Positioning System that enables GPS receivers to determine their current location, time and velocity. The GPS satellites are maintained by the United States Air Force.

GTK Group Temporal Key

KEK Key Encryption Key

MAC Mandatory Access Control

MDM Mobile Device Management

MITM Man in the Middle

MMS Multimedia Message Service

OTA Over The Air

In this context this term refers to downloading or uploading content or software (such as downloading ringtones, uploading images, etc.).

PIN Personal Identification Number

PKI Public Key Infrastructure

REK Root Encryption Key

SMS Short Message Service

TEE Trusted Execution Environment, a secure area of the main processor of a smart phone.

TOE Target of Evaluation

VPN Virtual Private Network

WLAN Wireless Local Area Network

11.2 References

[CC] Common Criteria for Information Technology Security Evaluation

Part 1: Introduction and general model, September 2012, Version 3.1, Revision 4, CCMB-2012-09-001

Part 2: Security functional components, September 2012, Version 3.1, Revision 4, CCMB-2012-09-002

Part 3: Security assurance components, September 2012, Version 3.1, Revision 4, CCMB-2012-09-003

Common Methodology for Information Technology Security Evaluation, Evaluation Methodology, September 2012, Version 3.1, Revision 4, CCMB-2012-09-004

[MDFPP] Protection Profile for Mobile Device Fundamentals, Version 2.0, 17 September 2014

[TD0034] Revision of Test 5 in FCS_TLSC_EXT.1.1 & EXT.2.1 reqs in MDF PP V2.0, MDM PP V2.0, MDM Agent PP V2.0

[TD0044] Update to FMT_SMF_EXT.1

[TD0057] Update to TD0047 for Non Wear Leveled Flash Memory

[TD0058] MDFPP v2.0 FMT_SMF_EXT.1, function 15

[TD0059] FCS_SRV_EXT.1 & CAVS

[TD0060] FDP_IFC_EXT.1 & FMT_SMF_EXT.1 Function 3

[TD0064] Whitelisting SSIDs (FMT_SMF_EXT.1, function 6) in MDF PP v2.0