Apple Inc.

Apple iOS 10.2 PP_MD_V3.0, EP_MDM_AGENT_V3.0, & PP_WLAN_CLI_EP Security Target

Version 2.0
2017-07-27

VID: 10782

Prepared for:
Apple Inc.
1 Infinite Loop
Cupertino, CA 9501400
www.apple.com

Prepared by:
atsec information security Corp.
9130 Jollyville Road, Suite 260
Austin, TX 78759
www.atsec.com

Table of Contents

Table of Figures

Table of Tables

Revision History

Version	Date	Change
1.0	2017-07-19	Initial Release
2.0	2017-07-27	Addressing validator comments

1 Security Target Introduction

This document is the Common Criteria (CC) Security Target (ST) for the Apple iOS 10.2 operating system on various hardware platforms, listed in section 1.4, to be evaluated as a Mobile Device in compliance with:

- The Mobile Device Fundamentals Protection Profile Version 3.0, dated 10 June, 2016 [PP_MD_V3.0],
- The Extended Package for Mobile Device Management Agents Version 3.0 [EP_MDM_AGENT_V3.0], dated 21 November, 2016, and
- The General Purpose Operating Systems Protection Profile/ Mobile Device Fundamentals Protection Profile Extended Package (EP) Wireless Local Area Network (WLAN) Clients, dated 8 February, 2016 [PP_WLAN_CLI_EP_V1.0].

1.1 Security Target Reference

ST Title: Apple iOS 10.2 PP_MD_V3.0, EP_MDM_AGENT_V3.0, & PP_WLAN_CLI_EP Security Target

ST Version: Version 2.0

ST Date: 2017-07-27

1.2 TOE Reference

Target of Evaluation (TOE) Identification:

- Apple iOS 10.2 on iPhone and iPad devices.
  Please refer to section 1.4 for device specific information.
- The TOE guidance documentation as detailed in section 1.5.3.

TOE Developer: Apple Inc.

Evaluation Sponsor: Apple Inc.

1.3 TOE Overview

The TOE is the iOS 10.2 operating system, that runs on iPad and iPhone devices, with the underlying hardware platform. The operating system manages the device hardware, provides mobile device agent functionality, and provides the technologies required to implement native apps. iOS 10.2 provides a built-in Mobile Device Management (MDM) framework application programmer interface (API), giving management features that may be utilized by external MDM solutions, allowing enterprises to use profiles to control some of the device settings.

iOS 10.2 provides a consistent set of capabilities allowing the supervision of enrolled devices. This includes the preparation of devices for deployment, the subsequent management of the devices, and the termination of management.

The TOE provides cryptographic services for the encryption of data-at rest, for secure communication channels, for protection of configuration profiles, and for use by applications.

User data protection is provided by encrypting the user data, restricting access by applications and by restricting access until the user has been successfully authenticated.

User identification and authentication is provided by a user defined passphrase where the minimum length of the passphrase, passphrase rules, as well as the maximum number of consecutive failed authentication attempts can be configured by an administrator.

Security management capabilities are provided to users via the user interface of the device and to administrators through the installation of configuration profiles on the device. This installation can be done using the Apple Configurator tool or by using an MDM system.

The TOE protects itself by having its own code and data protected from unauthorized access (using hardware provided memory protection features), by encrypting user and TOE Security Functionality (TSF) data using TSF protected keys and encryption/decryption functions, by self-tests, by ensuring the integrity and authenticity of TSF updates and downloaded applications, and by locking the TOE upon user request or after a defined time of user inactivity.

In addition, the TOE implements a number of cryptographic protocols that can be used to establish a trusted channel to other IT entities.

The MDM Agent provides secure alerts to the MDM Server indicating status events.

1.4 TOE Description

The TOE is a mobile operating system with its underlying hardware. The TOE is intended to be used as a communication solution providing mobile staff connectivity to enterprise data.

The TOE provides an interface allowing the enterprise to supervise devices under their control.

At the highest level, the operating system part of the TOE acts as an intermediary between the underlying hardware and the apps operating on the TOE. Apps do not talk to the underlying hardware directly. Instead, they communicate with the hardware through a set of well-defined system interfaces. These interfaces make it easy to write apps that work consistently on devices having different hardware capabilities.

The TOE needs to be configured by an authorized administrator to operate in compliance with the requirements defined in this [ST]. The evaluated configuration for this includes:

- the requirement to define a password for user authentication,
- the specification of a password policy defining criteria on the minimum length and complexity of a password,
- the specification of the maximum number of consecutive failed attempts to enter the password,
- the specification of the session locking policy,
- the specification of the audio and video collection devices allowed,
- the specification of the virtual private network (VPN) connection,
- the specification of the wireless networks allowed, and
- the requirement of the certificates in the trust anchor database.

The following table lists the devices that are covered by this evaluation.

Device Name	Model Number	Proc-essor	Wi-Fi	Cellular	Bluetooth
iPhone 5s	A1533 (GSM)	A7	802.11/a/b/g/n/ac	See table 2	4.0
	A1533 (CDMA)		802.11/a/b/g/n/ac	See table 2	4.0
	A1453		802.11/a/b/g/n/ac	See table 2	4.0
	A1457		802.11/a/b/g/n/ac	See table 2	4.0
	A1530		802.11/a/b/g/n/ac	See table 2	4.0
iPhone 6 Plus/ iPhone 6	A1549/A1522 (GSM)	A8	802.11/a/b/g/n/ac	See table 2	4.0
	A1549/A1522 (CDMA)		802.11/a/b/g/n/ac	See table 2	4.0
	A1586/A1524		802.11/a/b/g/n/ac	See table 2	4.0
iPhone 6S Plus iPhone 6S	A1634/A1633 (US)	A9	802.11/a/b/g/n/ac	See table 2	4.2
	A1687/A1688 (Global)		802.11/a/b/g/n/ac	See table 2	4.2
iPhone 7 Plus/	A1784/A1778 (GSM)	A10	802.11/a/b/g/n/ac	See table 2	4.2
iPhone 7	A1661/A1660 (CDMA)		802.11/a/b/g/n/ac	See table 2	4.2
iPhone SE	A1662 (US)	A9	802.11/a/b/g/n/ac	See table 2	4.2
	A1723 (Global)				
iPad mini 2	A1489 (Wi-Fi only)	A7	802.11a/b/g/n	-	4.0
	A1490 (Wi-Fi + cellular)		802.11a/b/g/n	See table 2	4.0
	A1491 (Wi-Fi + cellular)		802.11a/b/g/n	See table 2	4.0
iPad mini 3	A1599 (Wi-Fi only)	A7	802.11a/b/g/n	-	4.0
	A1600 (Wi-Fi + cellular)		802.11a/b/g/n	See table 2	4.0
	A1601 (Wi-Fi + cellular)		802.11a/b/g/n	See table 2	4.0
iPad mini 4	A1538 (Wi-Fi only)	A8	802.11a/b/g/n	-	4.2
	A1550 (Wi-Fi + cellular)		802.11a/b/g/n	See table 2	4.2
iPad Air	A1474 (Wi-Fi only)	A7	802.11a/b/g/n	-	4.0
	A1475 (Wi-Fi + cellular)		802.11a/b/g/n	See table 2	4.0
	A1476 (Wi-Fi + cellular)		802.11a/b/g/n	See table 2	4.0

Device Name	Model Number	Proc-essor	Wi-Fi	Cellular	Bluetooth
iPad Air 2	A1566 (Wi-Fi only) A1567 (Wi-Fi + cellular)	A8X	802.11a/b/g/n/ac 802.11a/b/g/n/ac	- See table 2	4.2 4.2
iPad Pro 12.9”	A1584 (Wi-Fi only) A1652 (Wi-Fi + cellular)	A9X	802.11/a/b/g/n/ac 802.11/a/b/g/n/ac	- See table 2	4.2 4.2
iPad Pro 9.7”	A1673 (Wi-Fi only) A1674 (Wi-Fi + cellular)	A9X	802.11/a/b/g/n/ac 802.11 /a/b/g/n/ac	- See table 2	4.2 4.2
iPad	A1822 (Wi-Fi only) A1823 (Wi-Fi + cellular)	A9	802.11/a/b/g/n/ac 802.11/a/b/g/n/ac	- See table 2	4.2 4.2

Table 1: Devices Covered by the Evaluation

The following table lists the cellular protocols supported by each model.

Device Name	Model Number	Cellular

iPhone 5s	A1533 (GSM)	UMTS/HSPA+/DC-HSDPA (850, 900, 1700/2100, 1900, 2100 MHz);
		GSM/EDGE (850, 900, 1800, 1900 MHz);
		LTE (Bands 1, 2, 3, 4, 5, 8, 13, 17, 19, 20, 25)

	A1533 (CDMA)	CDMA EV-DO Rev. A and Rev. B (800, 1700/2100, 1900, 2100 MHz);
		UMTS/HSPA+/DC-HSDPA (850, 900, 1700/2100, 1900, 2100 MHz);
		GSM/EDGE (850, 900, 1800, 1900 MHz);
		LTE (Bands 1, 2, 3, 4, 5, 8, 13, 17, 19, 20, 25)

	A1453	CDMA EV-DO Rev. A and Rev. B (800, 1700/2100, 1900, 2100 MHz);
		UMTS/HSPA+/DC-HSDPA (850, 900, 1700/2100, 1900, 2100 MHz);
		GSM/EDGE (850, 900, 1800, 1900 MHz);
		LTE (Bands 1, 2, 3, 4, 5, 8, 13, 17, 18, 19, 20, 25, 26)

	A1457	UMTS/HSPA+/DC-HSDPA (850, 900, 1900, 2100 MHz);
		GSM/EDGE (850, 900, 1800, 1900 MHz);
		LTE (Bands 1, 2, 3, 5, 7, 8, 20)

	A1530	UMTS/HSPA+/DC-HSDPA (850, 900, 1900, 2100 MHz);
		GSM/EDGE (850, 900, 1800, 1900 MHz);
		FDD-LTE (Bands 1, 2, 3, 5, 7, 8, 20);
		TD-LTE (Bands 38, 39, 40)

iPhone 6 Plus/ iPhone 6	A1549/A1522 (GSM)	UMTS/HSPA+/DC-HSDPA (850, 900, 1700/2100, 1900, 2100 MHz) GSM/EDGE (850, 900, 1800, 1900 MHz) LTE (Bands 1, 2, 3, 4, 5, 7, 8, 13, 17, 18, 19, 20, 25, 26, 28, 29)
	A1549/A1522 (CDMA)	CDMA EV-DO Rev. A and Rev. B (800, 1700/2100, 1900, 2100 MHz) UMTS/HSPA+/DC-HSDPA (850, 900, 1700/2100, 1900, 2100 MHz) GSM/EDGE (850, 900, 1800, 1900 MHz) LTE (Bands 1, 2, 3, 4, 5, 7, 8, 13, 17, 18, 19, 20, 25, 26, 28, 29)

Device Name	Model Number	Cellular
	A1586/A1524	CDMA EV-DO Rev. A and Rev. B (800, 1700/2100, 1900, 2100 MHz) UMTS/HSPA+/DC-HSDPA (850, 900, 1700/2100, 1900, 2100 MHz) TD-SCDMA 1900 (F), 2000 (A) GSM/EDGE (850, 900, 1800, 1900 MHz) FDD-LTE (Bands 1, 2, 3, 4, 5, 7, 8, 13, 17, 18, 19, 20, 25, 26, 28, 29) TD-LTE (Bands 38, 39, 40, 41)
iPhone 6S Plus/ iPhone 6S	A1634/A1633 (US)	CDMA EV-DO Rev. A (800, 1700/2100, 1900, 2100 MHz) UMTS/HSPA+/DC-HSDPA (850, 900, 1700/2100, 1900, 2100 MHz) TD-SCDMA 1900 (F), 2000 (A) GSM/EDGE (850, 900, 1800, 1900 MHz) LTE (Bands 1, 2, 3, 4, 5, 7, 8, 12, 13, 17, 18, 19, 20, 25, 26, 27, 28, 29, 30) TD-LTE (Bands 38, 39, 40, 41)
	A1687/A1688 (Global)	CDMA EV-DO Rev. A (800, 1700/2100, 1900, 2100 MHz) UMTS/HSPA+/DC-HSDPA (850, 900, 1700/2100, 1900, 2100 MHz) TD-SCDMA 1900 (F), 2000 (A) GSM/EDGE (850, 900, 1800, 1900 MHz) LTE (Bands 1, 2, 3, 4, 5, 7, 8, 12, 13, 17, 18, 19, 20, 25, 26, 27, 28, 29) TD-LTE (Bands 38, 39, 40, 41)
iPhone 7 Plus/ iPhone 7	A1784/A1778 (GSM)	UMTS/HSPA+/DC-HSDPA (850, 900, 1700/2100, 1900, 2100 MHz) GSM/EDGE (850, 900, 1800, 1900 MHz) FDD-LTE (Bands 1, 2, 3, 4, 5, 7, 8, 12, 13, 17, 18, 19, 20, 25, 26, 27, 28, 29, 30) TD-LTE (Bands 38, 39, 40, 41)
	A1661/A1660 (CDMA)	CDMA EV-DO Rev. A (800, 1900, 2100 MHz) UMTS/HSPA+/DC-HSDPA (850, 900, 1700/2100, 1900, 2100 MHz) TD-SCDMA 1900 (F), 2000 (A) GSM/EDGE (850, 900, 1800, 1900 MHz) FDD-LTE (Bands 1, 2, 3, 4, 5, 7, 8, 12, 13, 17, 18, 19, 20, 25, 26, 27, 28, 29, 30) TD-LTE (Bands 38, 39, 40, 41)

Device Name	Model Number	Cellular
iPhone SE	A1662 (US)	CDMA EV-DO Rev. A (800, 1700/2100, 1900, 2100 MHz) UMTS/HSPA+/DC-HSDPA (850, 900, 1700/2100, 1900, 2100 MHz) GSM/EDGE (850, 900, 1800, 1900 MHz) LTE (Bands 1, 2, 3, 4, 5, 8, 12, 13, 17, 18, 19, 20, 25, 26, 29)
	A1723 (Global)	CDMA EV-DO Rev. A (800, 1700/2100, 1900, 2100 MHz) UMTS/HSPA+/DC-HSDPA (850, 900, 1700/2100, 1900, 2100 MHz) TD-SCDMA 1900 (F), 2000 (A) GSM/EDGE (850, 900, 1800, 1900 MHz) LTE (Bands 1, 2, 3, 4, 5, 7, 8, 12, 17, 18, 19, 20, 25, 26, 28) TD-LTE (Bands 38, 39, 40, 41)
iPad mini 2	A1490	UMTS/HSPA/HSPA+/DC-HSDPA (850, 900, 1700/2100, 1900, 2100 MHz); GSM/EDGE (850, 900, 1800, 1900 MHz) CDMA EV-DO Rev. A and Rev. B (800, 1900 MHz) LTE (Bands 1, 2, 3, 4, 5, 7, 8, 13, 17, 18, 19, 20, 25, 26)
	A1491	UMTS (WCDMA)/HSPA+/ DC-HSDPA (850, 900, 1900, 2100 MHz), GSM/EDGE (850, 900, 1800, 1900 MHz), TD-SCDMA (1900 (F), 2000 (A)) LTE (Bands 1, 2, 3, 5, 7, 8, 18, 19, 20) TD-LTE (Bands 38, 39)
iPad mini 3	A1600	UMTS/HSPA/HSPA+/DC‑HSDPA (850, 900, 1700/2100, 1900, 2100 MHz); GSM/EDGE (850, 900, 1800, 1900 MHz) CDMA EV-DO Rev. A and Rev. B (800, 1900 MHz) LTE (Bands 1, 2, 3, 4, 5, 7, 8, 13, 17, 18, 19, 20, 25, 26)
	A1601	UMTS/HSPA/HSPA+/DC‑HSDPA (850, 900, 1700/2100, 1900, 2100 MHz); GSM/EDGE (850, 900, 1800, 1900 MHz) CDMA EV-DO Rev. A (800, 1900 MHz) TD-SCDMA (1900 (F), 2000 (A)) LTE (Bands 1, 2, 3, 4, 5, 7, 8, 18, 19, 20) TD-LTE (Bands 38, 39, 40)

Device Name	Model Number	Cellular
iPad mini 4	A 1550	UMTS/HSPA/HSPA+/DC-HSDPA (850, 900, 1700/2100, 1900, 2100 MHz); GSM/EDGE (850, 900, 1800, 1900 MHz) CDMA EV-DO Rev. A and Rev. B (800, 1900 MHz) LTE (Bands 1, 2, 3, 4, 5, 7, 8, 13, 17, 18, 19, 20, 25, 26, 28, 29, 38, 39, 40, 41)
iPad Air	A1475	UMTS/HSPA/HSPA+/DC-HSDPA (850, 900, 1700/2100, 1900, 2100 MHz); GSM/EDGE (850, 900, 1800, 1900 MHz) CDMA EV-DO Rev. A and Rev. B (800, 1900 MHz) LTE (Bands 1, 2, 3, 4, 5, 7, 8, 13, 17, 18, 19, 20, 25, 26)
	A1476	UMTS (WCDMA)/HSPA+/ DC-HSDPA (850, 900, 1900, 2100 MHz), GSM/EDGE (850, 900, 1800, 1900 MHz), TD-SCDMA (1900 (F), 2000 (A)) LTE (Bands 1, 2, 3, 4, 5, 7, 8, 13, 17, 18, 19, 20, 25, 26) TD-LTE (Bands 38, 39)
iPad Air 2	A1567	GSM/EDGE (850, 900, 1800, 1900 MHz), UMTS/HSPA/HSPA+/DC-HSDPA (850, 900, 1700/2100, 1900, 2100 MHz), CDMA EV-DO Rev. A and Rev. B (800, 1900 MHz), TD-SCDMA LTE (Bands 1, 2, 3, 4, 5, 7, 8, 13, 17,18, 19, 20, 25, 26, 28, 29) TD-LTE (Bands 38, 39, 40,41)
iPad Pro 12.9”	A1652	CDMA EV-DO Rev. A and Rev. B (800, 1900 MHz) UMTS/HSPA/HSPA+/DC-HSDPA (850, 900, 1700/2100, 1900, 2100 MHz); GSM/EDGE (850, 900, 1800, 1900 MHz) LTE (Bands 1, 2, 3, 4, 5, 7, 8, 13, 17, 18, 19, 20, 25, 26, 28, 29, 38, 39, 40, 41)
iPad Pro 9.7”	A1674	CDMA EV-DO Rev. A and Rev. B (800, 1900 MHz) UMTS/HSPA/HSPA+/DC-HSDPA (850, 900, 1700/2100, 1900, 2100 MHz); GSM/EDGE (850, 900, 1800, 1900 MHz) LTE Advanced (Bands 1, 2, 3, 4, 5, 7, 8, 12, 13, 17, 18, 19, 20, 25, 26, 27, 28, 29, 30, 38, 39, 40, 41)
iPad	A1823	CDMA EV-DO Rev. A and Rev. B UMTS/HSPA/HSPA+/DC-HSDPA (850, 900, 1700/2100, 1900, 2100 MHz); GSM/EDGE (850, 900, 1800, 1900 MHz) LTE (Bands 1, 2, 3, 4, 5, 7, 8, 12, 13, 17, 18, 19, 20, 25, 26, 28, 29, 38, 39, 40, 41)

Table 2: Cellular Protocols Supported

1.5 TOE Architecture

The implementation of TOE architecture can be viewed as a set of layers, which are shown in Figure 1. Lower layers contain fundamental services and technologies. Higher-level layers build upon the lower layers and provide more sophisticated services and technologies.

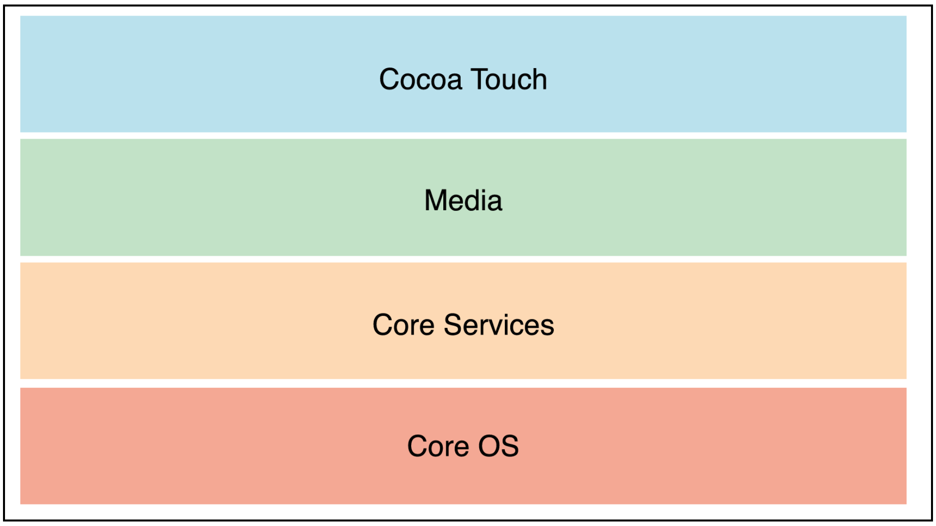


Figure 1: Layers of iOS

The individual layers provide the following services.

The **Cocoa Touch layer** contains key frameworks for building iOS apps. These frameworks define the appearance of applications (apps). They also provide the basic app infrastructure and support for key technologies such as multitasking, touch-based input, push notifications, and many high-level system services. When designing apps, one should investigate the technologies in this layer first to see if they meet the needs.

The **Media layer** contains the graphics, audio, and video technologies you use to implement multimedia experiences in apps. The technologies in this layer make it easy to build apps that look and sound great.

The **Core Services layer** contains fundamental system services for apps. Key among these services are the Core Foundation and Foundation frameworks, which define the basic types that all apps use. This layer also contains individual technologies to support features such as location, iCloud, social media, and networking.

This layer also implements data protection functions that allow apps that work with sensitive user data to take advantage of the built-in encryption available on some devices. When an app designates a specific file as protected, the system stores that file on disk in an encrypted format. While the device is locked, the contents of the file are inaccessible to both the app and to any potential intruders. However, when the device is unlocked by the user, a decryption key is created to allow the app to access the file. Other levels of data protection are also available.

The **Core OS layer** contains the low-level features that most other technologies are built upon. Even if an app does not use these technologies directly, they are most likely being used by other frameworks. And in situations where an app needs to explicitly deal with security or communicating with an external hardware accessory, it does so by using the frameworks in this layer.

Security related frameworks provided by this layer are:

- the Generic Security Services Framework, providing services as specified in Request for Comment (RFC) 2743 (Generic Security Service Application

Program Interface Version 2, Update 1) and RFC 4401 (Pseudo Random Function);

- the Local Authentication Framework;
- the Network Extension Framework, providing support for configuring and controlling VPN tunnels;
- the Security Framework, providing services to manage and store certificates, public and private keys, and trust policies (this framework also provides the Common Crypto library for symmetric encryption and hash-based message authentication codes); and
- the System Framework, providing the kernel environment, drivers, and low-level UNIX interfaces (the kernel manages the virtual memory system, threads, file system, network, and inter-process communication and is therefore responsible for separating apps from each other and controlling the use of low-level resources).

The TOE may be managed by an MDM solution that enables an enterprise to control and administer the TOE instances that are enrolled in the MDM solution.

1.5.1 Physical Boundaries

The TOE is a Mobile Device which is composed of a hardware platform and its system software. It provides wireless connectivity and includes software for VPN connection, for access to the protected enterprise network, enterprise data and applications, and for communicating with other Mobile Devices. The software for the VPN connection is evaluated separately.

The TOE does not include the user applications that run on top of the operating system, but does include controls that limit application behavior. The TOE may be used as a mobile device within an enterprise environment where the configuration of the device is managed through an evaluated MDM solution.

1.5.2 Security Functions provided by the TOE

The TOE provides the security functionality required by [PP_MD_V3.0], [EP_MDM_AGENT_V3.0] and [PP_WLAN_CLI_EP_V1.0].

1.5.2.1 Cryptographic Support

The TOE provides cryptographic services via the following two cryptographic modules.

- Apple iOS CoreCrypto Kernel Module v7
- Apple iOS CoreCrypto Module v7

The iOS CoreCrypto Kernel Module is an iOS kernel extension optimized for library use within the iOS kernel. Once the module is loaded into the iOS kernel its cryptographic functions are made available to iOS Kernel services only.

The following figure shows the boundary of the iOS CoreCrypto Kernel Module within the TOE.

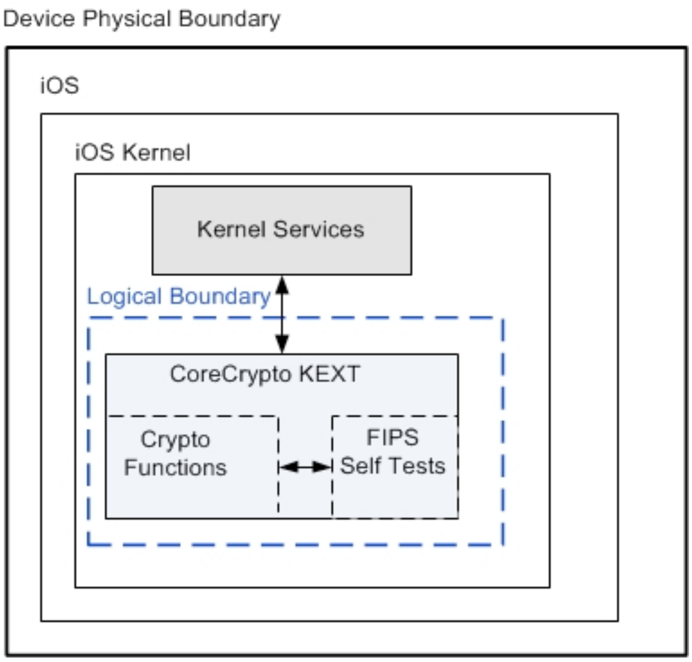


Figure 2: Logical Block Diagram of the iOS CoreCrypto Kernel Module

The iOS CoreCrypto Module is designed for library use within the iOS user space. It is implemented as an iOS dynamically loadable library. The dynamically loadable library is loaded into the iOS application and its cryptographic functions are made available to the application. A second instance of this module is used within the secure enclave to provide cryptographic services there.

The following figure shows the boundary of the iOS CoreCrypto Module within the TOE.

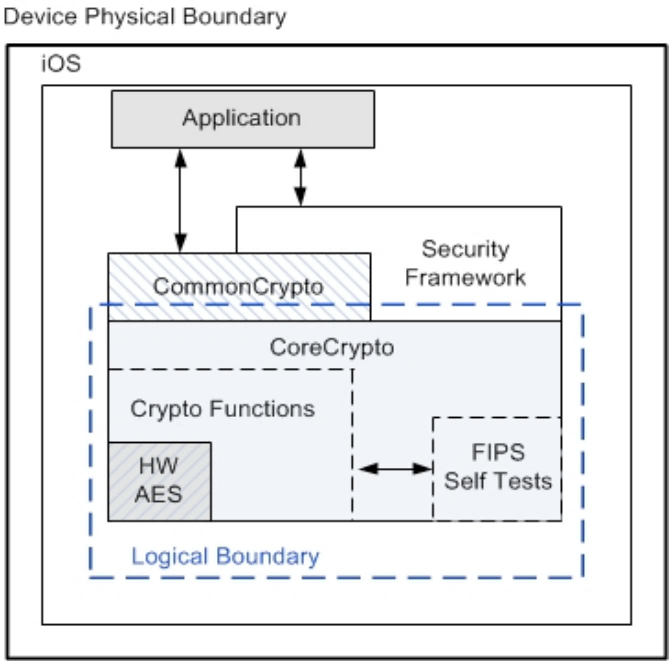


Figure 3: Logical Block Diagram of the iOS CoreCrypto Module.

The cryptographic functions provided include symmetric key generation, encryption and decryption using the Triple-DES and Advanced Encryption Standard (AES) algorithms, asymmetric key generation and key establishment, cryptographic hashing, and keyed-hash message authentication.

Note that the Wi-Fi chip, performs the AES functions.

The functions listed below are used to implement the security protocols supported as well as for the encryption of data-at-rest.

- **Random Number Generation; Symmetric Key Generation**
    - SP 800-90A Deterministic Random Bit Generator (DRBG)
        - CTR_DRBG with AES 128 core, with derivation function and without prediction resistance
        - CTR_DRBG with AES 256 core without derivation function and without prediction resistance
- **Symmetric Encryption and Decryption**
    - FIPS 197 AES, SP 800-38 A, SP 800-38 C, SP 800-38 DSP 800-38 F
        - Key sizes: 128/256 bits
          Block chaining modes: CBC, CCM, GCM, KW
- **Digital Signature and Asymmetric Key Generation**
    - FIPS186-4 RSA, PKCS #1.5
        - KEY(gen)(2048/3072), SigGenPKCS1.5 (2048/3072), SigVerPKCS1.5 (1024/2048/3072)
    - FIPS 186-4 ECDSA, ANSI X9.62
        - PKG: curves P-256, P-384
        - PKV: curves P-256, P-384
        - SIG(gen): curves P-256, P-384
        - SIG(ver): curves P-256, P-384
- **Message Digest**
    - FIPS 180-4 SHS
        - SHA-1, SHA-2(256, 384, 512)
- **Keyed Hash**
    - FIPS 198 HMAC
        - SHA-1, SHA-2(256, 384, 512)
- **PBKDF**
    - SP 800-132
        - Password based key derivation using HMAC with SHA-1 or SHA-2 as pseudorandom function
- **EC Diffie-Hellman**
    - Curve25519

1.5.2.2 User Data Protection

User data in files is protected using cryptographic functions, ensuring this data remains protected even if the device gets lost or is stolen. Critical data, like passwords used by applications or application defined cryptographic keys, can be stored in the key chain, which provides additional protection. Password protection and encryption ensure that data-at-rest remains protected even in the case of the device being lost or stolen.

Data can also be protected such that only the application that owns the data can access it.

1.5.2.3 Identification and Authentication

Except for making emergency calls, users need to authenticate using a password. This password can be configured for a minimum length, for dedicated password policies and for a maximum life time. When entered, passwords are obscured and the frequency of entering passwords is limited as well as the number of consecutive failed attempts of entering the password. The TOE also enters a locked state after a (configurable) time of user inactivity and the user is required to enter his password to unlock the TOE.

External entities connecting to the TOE via a secure protocol (Extensible Authentication Protocol Transport Layer Security (EAP-TLS), Transport Layer Security (TLS), IPsec) can be authenticated using X.509 certificates.

1.5.2.4 Security Management

The security functions listed in Table 4: Management Functions, can be managed either by the user or by an authorized administrator through an MDM system. This table identifies the functions that can be managed and indicates, if the management can be performed by the user, by the authorized administrator, or both.

1.5.2.5 Protection of the TSF

Some of the functions the TOE implements to protect the TSF and TSF data are as follows.

- Protection of cryptographic keys—keys used for TOE internal key wrapping and for the protection of data-at-rest are not exportable. There are provisions for fast and secure wiping of key material.
- Use of memory protection and processor states to separate applications and protect the TSF from unauthorized access to TSF resources—in addition, each device includes a separate system called the "secure enclave" which is the only system that can use the Root Encryption Key (REK). The secure enclave is a separate CPU that executes a stand-alone operating system and has separate memory.
- Digital signature protection of the TSF image—all updates to the TSF need to be digitally signed.
- Software/firmware integrity self-test upon start-up—the TOE will not go operational when this test fails.
- Digital signature verification for applications
- Access to defined TSF data and TSF services only when the TOE is unlocked

1.5.2.6 TOE Access

The TSF provides functions to lock the TOE upon request and after an administrator-configurable time of inactivity.

Access to the TOE via a wireless network is controlled by user/administrator defined policy.

1.5.2.7 Trusted Path/Channels

The TOE supports the use of the following cryptographic protocols that define a trusted channel between itself and another trusted IT product.

- IEEE 802.11-2012
- IEEE 802.1X
- EAP-TLS
- TLS
- IPsec (addressed in a separate evaluation)

1.5.2.8 Audit

The TOE provides the ability for responses to be sent from the MDM Device Agent to the MDM Server. These responses are configurable by the organization using a scripting language given in the Over-the-Air Profile Delivery and Configuration document.

1.5.3 TOE Documentation

[ACONFIG] Apple Configurator 2 Help
http://help.apple.com/configurator/mac/2.2//

[CC_GUIDE] Apple iOS 10.2 Common Criteria Guide.
https://www.niap-ccevs.org/st/st_vid10725-agd.pdf

[CKTSPG] Certificate, Key, and Trust Services Programming Reference.
https://developer.apple.com/library/ios/documentation/Security/Conceptual/CertKeyTrustProgGuide/01introduction/introduction.html

[CKTSREF] Certificate, Key, and Trust Services Reference.
https://developer.apple.com/library/ios/documentation/Security/Reference/certifkeytrustservices/index.html

[CRYPTOGUIDE] Cryptographic Services Guide
https://developer.apple.com/library/ios/documentation/Security/Conceptual/cryptoservices/Introduction/Introduction.html

[DEP_Guide] Apple Deployment Programs Device Enrollment Program Guide
https://www.apple.com/ae/business/docs/DEP_Guide.pdf

[HTTPSTN2232] Technical Note TN 2232: HTTPS Server Trust Evaluation
https://developer.apple.com/library/ios/technotes/tn2232/_index.html

[iPhone_UG] iPhone User Guide for iOS 9.3 (Sept 16, 2015)
https://itunes.apple.com/us/book/iphone-user-guide-for-ios-9.3/id1035373510?mt=11

[iPad_UG] iPad User Guide for iOS 9.3 (Sept 16, 2015)
https://itunes.apple.com/us/book/ipad-user-guide-for-ios-9.3/id1035374126?mt=11

[IPLKEYREF] Information Property List Key Reference
https://developer.apple.com/library/ios/documentation/General/Reference/InfoPlistKeyReference/Introduction/Introduction.html

[iOS_CFG] Configuration Profile Reference (March 21, 2016)
https://developer.apple.com/library/ios/featuredarticles/iPhoneConfigurationProfileRef/Introduction/Introduction.html

[iOSDeployRef] iOS Deployment Reference
https://itunes.apple.com/us/book/ios-deployment-reference/id917468024?mt=11

[iOS_MDM] Mobile Device Management Protocol Reference (January 20, 2016)
https://developer.apple.com/go/?id=mobile-device-management-protocol-reference

[iOSMP] iOS Manual Pages
https://developer.apple.com/library/ios/documentation/System/Conceptual/ManPages_iPhoneOS/

[KEYCHANGEPG] Keychain Services Programming Guide
https://developer.apple.com/library/ios/technotes/tn2232/_index.html

[PM_Help] Profile Manager Help
https://help.apple.com/profilemanager/mac/5.1/#/apdB8464D5D-F4C9-4848-9AD7-4B188ED6E130r

[SECFWREF] Security Framework Reference
https://developer.apple.com/library/ios/documentation/Security/Reference/SecurityFrameworkReference/index.html

1.5.4 Technical Decisions

The following technical decisions were found to be applicable to the TOE:

[TD0214] FCS_CKM.1 & FCS_CKM_EXT.5 – Selectable Audit Events
https://www.niap-ccevs.org/Documents_and_Guidance/view_td.cfm?td_id=219

[TD0194] Update to audit of FTP_ITC_EXT.1/WLAN
https://www.niap-ccevs.org/Documents_and_Guidance/view_td.cfm?td_id=198

[TD0180] Update to audit of FDP_DAR_EXT.1
https://www.niap-ccevs.org/Documents_and_Guidance/view_td.cfm?td_id=184

[TD0174] Optional Ciphersuites for TLS
https://www.niap-ccevs.org/Documents_and_Guidance/view_td.cfm?td_id=178

[TD0163] Update to FCS_TLSC_EXT.1.1 Test 5.4 and FCS_TLSS_EXT.1.1
https://www.niap-ccevs.org/Documents_and_Guidance/view_td.cfm?td_id=167

[TD0158] FMT_SMF_EXT.1
https://www.niap-ccevs.org/Documents_and_Guidance/view_td.cfm?td_id=162

[TD0147] FIA_UAU.5.2
https://www.niap-ccevs.org/Documents_and_Guidance/view_td.cfm?td_id=151

[TD0146] FCS_STG_EXT.3.1 - Integrity
https://www.niap-ccevs.org/Documents_and_Guidance/view_td.cfm?td_id=150

[TD0145] FCS_CKM_EXT.3.1 – Security strength of KEKs
https://www.niap-ccevs.org/Documents_and_Guidance/view_td.cfm?td_id=149

[TD0120] FMT_SMF_EXT.1, Functions 2 & 5, Users and/or Administrators Configuration
https://www.niap-ccevs.org/Documents_and_Guidance/view_td.cfm?td_id=123

[TD0118] FAU_GEN.1 Application of Audit Requirements Update
https://www.niap-ccevs.org/Documents_and_Guidance/view_td.cfm?td_id=121

[TD0103] Access Control Policy Prohibiting Apps Write/Exe Permissions
https://www.niap-ccevs.org/Documents_and_Guidance/view_td.cfm?td_id=8106

The following technical decisions were considered but were inapplicable the TOE.

[TD0210] FIA_BMG_EXT.1.2 - SAFAR Update
https://www.niap-ccevs.org/Documents_and_Guidance/view_td.cfm?td_id=215

[TD0148] Hardware-protected hash in FPT_TST_EXT.2.1
https://www.niap-ccevs.org/Documents_and_Guidance/view_td.cfm?td_id=152

[TD0079] RBG Cryptographic Transitions per NIST SP 800-131A Revision 1
https://www.niap-ccevs.org/Documents_and_Guidance/view_td.cfm?td_id=82

[TD0042] Removal of Low-level Crypto Failure Audit from PPs
https://www.niap-ccevs.org/Documents_and_Guidance/view_td.cfm?td_id=45

1.5.5 Other References

[BT] Specification of the Bluetooth System 4.0.0218
https://www.bluetooth.com/specifications/adopted-specifications

[PP_MD_V3.0] U.S. Government Approved Protection Profile - Protection Profile for Mobile Device Fundamentals, Version 3.0
https://www.niap-ccevs.org/Profile/Info.cfm?id=353

[EP_MDM_AGENT_V3.0] U.S. Government Approved Protection Profile - Extended Package for Mobile Device Management Agents Version 3.0
https://www.niap-ccevs.org/Profile/Info.cfm?id=360

[PP_WLAN_CLI_V1.0] Extended Package for WLAN Client Version 1.0
https://www.niap-ccevs.org/Profile/Info.cfm?id=386

[iOS_SEC] iOS Security, iOS 10 (March 2017)
https://www.apple.com/business/docs/iOS_Security_Guide.pdf

[iOS_TEC] iOS Technology Overview
https://developer.apple.com/library/ios/documentation/Miscellaneous/Conceptual/iPhoneOSTechOverview/Introduction/Introduction.html

[FIPS_CORECRYPTO_IOS_US_SECPOL_3.2] – FIPS Security Policy
http://csrc.nist.gov/groups/STM/cmvp/documents/140-1/140val-all.htm#2827

2 Conformance Claims

2.1 CC Conformance

This [ST] is conformant to the following CC documents.

- Common Criteria for Information Technology Security Evaluations Part 1, Version 3.1, Revision 4, September 2012,
- Common Criteria for Information Technology Security Evaluations Part 2, Version 3.1, Revision 4, September 2012: Part 2 extended, and
- Common Criteria for Information Technology Security Evaluations Part 3, Version 3.1, Revision 4, September 2012: Part 3 extended.

2.2 Protection Profile (PP) Conformance

This [ST] is conformant to the following PPs.

- Protection Profile for Mobile Device Fundamentals, Version 3.0, dated 10 June, 2016 [PP_MD_V3.0], and
- Extended Package for Mobile Device Management Agents Version 3.0, dated 21 November, 2016 [EP_MDM_AGENT_V3.0].
- General Purpose Operating Systems Protection Profile/ Mobile Device Fundamentals Protection Profile Extended Package (EP) Wireless Local Area Network (WLAN) Clients, dated 8 February, 2016 [PP_WLAN_CLI_EP_V1.0]

2.3 Conformance Rationale

This [ST] provides exact compliance with version 3.0 of the Mobile Device Fundamentals Protection Profile [PP_MD_V3.0] with the Extended Package for MDM Agents [EP_MDM_AGENT_V3.0], and for the WLAN client [PP_WLAN_CLI_EP_V1.0]. The security problem definition, security objectives and security requirements in this [ST] are all taken from the PP and the Extended Packages (EP) performing only operations defined there.

The requirements in the PPs are assumed to represent a complete set of requirements that serve to address any interdependencies. Given that all of the appropriate functional requirements given in the PPs have been copied into this [ST], the dependency analysis for the requirements is assumed to be already performed by the PP authors and is not reproduced in this document.

3 Security Problem Definition

The security problem definition has been taken from [PP_MD_V3.0], [EP_MDM_AGENT_V3.0] and [PP_WLAN_CLI_EP_V1.0]. It is reproduced here for the convenience of the reader.

3.1 Threats

T.EAVESDROP Network Eavesdropping (PP_MD_V3.0)

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_EAVESDROP Network Eavesdropping (EP_MDM_AGENT_V3.0)

Unauthorized entities may intercept communications between the MDM and mobile devices to monitor, gain access to, disclose, or alter remote management commands. Unauthorized entities may intercept unprotected wireless communications between the mobile device and the Enterprise to monitor, gain access to, disclose, or alter TOE data.

T.NETWORK Network Attack (PP_MD_V3.0)

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.NETWORK_ATTACK Network Attack (EP_MDM_AGENT_V3.0)

An attacker may masquerade as MDM Server and attempt to compromise the integrity of the mobile device by sending malicious management commands.

T.PHYSICAL Physical Access (PP_MD_V3.0)

An attacker, with physical access, may attempt to access user data on the Mobile Device including credentials. These physical access threats may involve attacks, which attempt to access the device through external hardware ports, impersonate the user authentication mechanisms, through its user interface, and also through direct and possibly destructive access to its storage media.

Note: Defending against device re-use after physical compromise is out of scope for this protection profile.

T.PHYSICAL_ACCESS Physical Access (EP_MDM_AGENT_V3.0)

The mobile device may be lost or stolen, and an unauthorized individual may attempt to access user data. Although these attacks are primarily directed against the mobile device platform, the MDM Agent configures features, which address this threat.

T.FLAWAPP Malicious or Flawed Application (PP_MD_V3.0)

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, and 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.MALICIOUS_APPS Malicious and Flawed Application (EP_MDM_AGENT_V3.0)

Malicious or flawed application threats exist because apps loaded onto a mobile device may include malicious or exploitable code. An administrator of the MDM or mobile device user may inadvertently import malicious code, or an attacker may insert malicious code into the TOE, resulting in the compromise of TOE or TOE data.

T.PERSISTENT Persistent Presence (PP_MD_V3.0)

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.

T.BACKUP (EP_MDM_AGENT_V3.0)

An attacker may try to target backups of data or credentials and exfiltrate data. Since the backup is stored on either a personal computer or end user’s backup repository, it’s not likely enterprise would detect compromise.

T.TSF_FAILURE (PP_WLAN_CLI_EP_V1.0)

Security mechanisms of the TOE generally build up from a primitive set of mechanisms (e.g., memory management, privileged modes of process execution) to more complex sets of mechanisms. Failure of the primitive mechanisms could lead to a compromise in more complex mechanisms, resulting in a compromise of the TSF.

T.UNAUTHORIZED ACCESS (PP_WLAN_CLI_EP_V1.0)

A user may gain unauthorized access to the TOE data and TOE executable code. A malicious user, process, or external IT entity may masquerade as an authorized entity in order to gain unauthorized access to data or TOE resources. A malicious user, process, or external IT entity may misrepresent itself as the TOE to obtain identification and authentication data.

T.UNDETECTED ACTIONS (PP_WLAN_CLI_EP_V1.0)

Malicious remote users or external IT entities may take actions that adversely affect the security of the TOE. These actions may remain undetected and thus their effects cannot be effectively mitigated.

3.2 Assumptions

A.CONFIG (PP_MD_V3.0)

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 (PP_MD_V3.0)

It is assumed that the mobile user will immediately notify the administrator if the Mobile Device is lost or stolen.

A.PRECAUTION (PP_MD_V3.0)

It is assumed that the mobile user exercises precautions to reduce the risk of loss or theft of the Mobile Device.

A.CONNNECTIVITY (EP_MDM_AGENT_V3.0)

The TOE relies on network connectivity to carry out its management activities. The TOE will robustly handle instances when connectivity is unavailable or unreliable.

A.MOBILE_DEVICE_PLATFORM (EP_MDM_AGENT_V3.0)

The MDM Agent relies upon Mobile platforms and hardware evaluated against the MDFPP and assured to provide policy enforcement as well as cryptographic services and data protection. The Mobile platform provides trusted updates and software integrity verification of the MDM Agent.

A.PROPER_ADMIN (EP_MDM_AGENT_V3.0)

One or more competent, trusted personnel who are not careless, willfully negligent, or hostile, are assigned and authorized as the TOE Administrators, and do so using and abiding by guidance documentation.

A.PROPER_USER (EP_MDM_AGENT_V3.0)

Mobile device users are not willfully negligent or hostile, and use the device within compliance of a reasonable Enterprise security policy.

A.NO_TOE_BYPASS (PP_WLAN_CLI_EP_V1.0)

Information cannot flow between the wireless client and the internal wired network without passing through the TOE.

A.TRUSTED_ADMIN (PP_WLAN_CLI_EP_V1.0)

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

3.3 Organizational Security Policies

An organizational security policy (OSP) is a set of rules, practices, and procedures imposed by an organization to address its security needs. The following OSPs must be enforced by the TOE or its operational environment.

P.ADMIN (EP_MDM_AGENT_V3.0)

The configuration of the mobile device security functions must adhere to the Enterprise security policy.

P.DEVICE_ENROLL (EP_MDM_AGENT_V3.0)

A mobile device must be enrolled for a specific user by the administrator of the MDM prior to being used in the Enterprise network by the user.

P.NOTIFY (EP_MDM_AGENT_V3.0)

The mobile user must immediately notify the administrator if a mobile device is lost or stolen so that the administrator may apply remediation actions via the MDM system.

P.ACCOUNTABILITY (EP_MDM_AGENT_V3.0)

Personnel operating the TOE shall be accountable for their actions within the TOE.

4 Security Objectives

The security objectives have been taken from [PP_MD_V3.0] ,[EP_MDM_AGENT_V3.0], and [PP_WLAN_CLI_EP_V1.0]. They are reproduced here for the convenience of the reader.

4.1 Security Objectives for the TOE

O.COMMS Protected Communications (PP_MD_V3.0)

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 (PP_MD_V3.0)

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 (PP_MD_V3.0)

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 (PP_MD_V3.0)

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 (PP_MD_V3.0)

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.

O.PRIVACY End User Privacy and Device Functionality (PP_MD_V3.0)

In a BYOD environment (use cases 3 and 4), a personally-owned mobile device is used for both personal activities and enterprise data. Enterprise management solutions may have the technical capability to monitor and enforce security policies on the device. However, the privacy of the personal activities and data must be ensured. In addition, since there are limited controls that the enterprise can enforce on the personal side, separation of personal and enterprise data is needed. This will protect against the T.FLAWAPP and T.PERSISTENT threats.

O. APPLY_POLICY (EP_MDM_AGENT_V3.0)

The TOE must facilitate configuration and enforcement of enterprise security policies on mobile devices via interaction with the mobile OS and the MDM Server. This will include the initial enrollment of the device into management, through its lifecycle including policy updates and through its possible unenrollment from management services.

O.ACCOUNTABILITY (EP_MDM_AGENT_V3.0)

The TOE must provide logging facilities which record management actions undertaken by its administrators.

O. DATA_PROTECTION_TRANSIT (EP_MDM_AGENT_V3.0)

Data exchanged between the MDM Server and the MDM Agent must be protected from being monitored, accessed, or altered.

O.AUTH_COMM (PP_WLAN_CLI_EP_V1.0)

The TOE will provide a means to ensure that it is communicating with an authorized Access Point and not some other entity pretending to be an authorized Access Point, and will provide assurance to the Access Point of its identity.

O.CRYPTOGRAPHIC_FUNCTIONS (PP_WLAN_CLI_EP_V1.0)

The TOE shall provide or use cryptographic functions (i.e., encryption/decryption and digital signature operations) to maintain the confidentiality and allow for detection of modification of data that are transmitted outside the TOE and its host environment.

O.SYSTEM_MONITORING (PP_WLAN_CLI_EP_V1.0)

The TOE will provide the capability to generate audit data.

O.TOE_ADMINISTRATION (PP_WLAN_CLI_EP_V1.0)

The TOE will provide mechanisms to allow administrators to be able to configure the TOE.

O.TSF_SELF_TEST (PP_WLAN_CLI_EP_V1.0)

The TOE will provide the capability to test some subset of its security functionality to ensure it is operating properly.

O.WIRELESS_ACCESS_POINT_CONNECTION (PP_WLAN_CLI_EP_V1.0)

The TOE will provide the capability to restrict the wireless access points to which it will connect.

4.2 Security Objectives for the TOE Environment

OE.CONFIG (PP_MD_V3.0)

TOE administrators will configure the Mobile Device security functions correctly to create the intended security policy

OE.NOTIFY (PP_MD_V3.0)

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

OE.PRECAUTION (PP_MD_V3.0)

The Mobile User exercises precautions to reduce the risk of loss or theft of the Mobile Device.

OE.IT_ENTERPRISE (EP_MDM_AGENT_V3.0)

The Enterprise IT infrastructure provides security for a network that is available to the TOE and mobile devices that prevents unauthorized access.

OE.MOBILE_DEVICE_PLATFORM (EP_MDM_AGENT_V3.0)

The MDM Agent relies upon the trustworthy Mobile platform and hardware to provide policy enforcement as well as cryptographic services and data protection. The Mobile platform provides trusted updates and software integrity verification of the MDM Agent.

OE.DATA_PROPER_ADMIN (EP_MDM_AGENT_V3.0)

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

OE.DATA_PROPER_USER (EP_MDM_AGENT_V3.0)

Users of the mobile device are trained to securely use the mobile device and apply all guidance in a trusted manner.

OE.WIRELESS_NETWORK (EP_MDM_AGENT_V3.0)

A wireless network will be available to the mobile devices.

OE.NO_TOE_BYPASS (PP_WLAN_CLI_EP_V1.0)

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

OE.TRUSTED_ADMIN (PP_WLAN_CLI_EP_V1.0)

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

5 Extended Components Definition

The Security Target draws upon the extended components implicitly defined in the [PP_MD_V3.0], [PP_MD_AGENT_V3.0] and [PP_WLAN_CLI_EP_V1.0].

6 Security Functional Requirements

This chapter describes the Security Functional Requirements (SFRs) for the TOE. The SFRs have been taken from [PP_MD_V3.0], [PP_MD_AGENT_V3.0] and [PP_WLAN_CLI_EP_V1.0] with selections, assignments and potential refinements being applied.

For each SFR, the source of the SFR is indicated in parentheses, thus:

- (MDF) – The SFR can be found in [PP_MD_V3.0],
- (AGENT) – The SFR can be found in [PP_MD_AGENT_V3.0]
- (WLAN) – The SFR can be found in [PP_WLAN_CLI_EP_V1.0]

Selections and assignment operations performed as required by [PP_MD_V3.0], [PP_MD_AGENT_V3.0] and [PP_WLAN_CLI_EP_V1.0] are marked in **bold**.

Refinements are marked indicated in ***bold, italics and underlined***. Note that this [ST] does not identify selections, assignments or refinements already applied in [PP_MD_V3.0], [PP_MD_AGENT_V3.0] and [PP_WLAN_CLI_EP_V1.0].

6.1 Security Audit (FAU)

Agent Alerts (FAU_ALT)

FAU_ALT_EXT.2 Extended: Agent Alerts

FAU_ALT_EXT.2.1(AGENT)

The MDM Agent shall provide an alert via the trusted channel to the MDM Server in the event of any of the following:

- successful application of policies to a mobile device;
- **receiving** periodic reachability events;
- **no other events**

FAU_ALT_EXT.2.2(AGENT)

The MDM Agent shall queue alerts if the trusted channel is not available.

Audit Data Generation (FAU_GEN)

FAU_GEN.1(1) Audit Data Generation

FAU_GEN.1.1(1)(MDF)

The TSF shall be able to generate an audit record of the following auditable events:

1) Start-up and shutdown of the audit functions;
2) All auditable events for the [not selected] level of audit;
3) All administrative actions;
4) Start-up and shutdown of the Rich OS;
5) Insertion or removal of removable media;
6) Specifically defined auditable events in **Table 3: Audit** ;
7) **No other auditable events derived from this profile.**
8) **No additional auditable events.**

FAU_GEN.1.2(1)(MDF)

The TSF shall record within each audit record at least the following information:

1) Date and time of the event;
2) type of event;
3) subject identity;
4) the outcome (success or failure) of the event;
5) additional information in **Table 3: Audit ;**
6) **no additional information.**

FAU_GEN.1(2) Audit Data Generation

FAU_GEN.1.1(2)(AGENT)

The MDM Agent shall be able to generate an MDM Agent audit record of the following auditable events:

- startup and shutdown of the MDM Agent,
- change in MDM policy,

- any modification commanded by the MDM Server,
- specifically defined auditable events listed in Table 3: Audit ,
- **no other events.**

FAU_GEN.1.2(2)(AGENT)

The **TSF** shall record within each MDM Agent audit record at least the following information:

- date and time of the event,
- type of event,
- subject identity,
- (if relevant) the outcome (success or failure) of the event,
- additional information in **Table 3: Audit ;**
- **no other relevant audit information**.

Requirement	Auditable Events	Additional Audit Record Contents
FAU_ALT_EXT.2(AGENT)	Type of alert.	No additional information
FAU_SEL.1(AGENT)	All modifications to the audit configuration that occur while the audit collection functions are operating.	No additional information
FCS_CKM_EXT.1(MDF)	**None.**	No additional information.
FCS_CKM_EXT.5(MDF)	**None.**	No additional information
FCS_CKM.1(MDF)	**None.**	No additional information
FCS_TLSC_EXT.1(AGENT)	Failure to establish a TLS session	Reason for failure.
	Failure to verify presented identifier	Presented identifier and reference identifier.
	Establishment/termination of a TLS session.	Non-TOE endpoint of connection.
FCS_TLSC_EXT.1(WLAN)	Failure to establish an EAP-TLS session.	Reason for failure. Non-TOE endpoint connection.
	Establishment/termination of an EAP-TLS session.	
FCS_STG_EXT.1(MDF)	Import or key destruction	Identity of key. Role and identity of requestor.
	No other events	
FCS_STG_EXT.3(MDF)	Failure to verify integrity of stored key.	Identity of key being verified.
FDP_DAR_EXT.1(MDF)	Failure to encrypt/decrypt data.	No additional information
FDP_DAR_EXT.2(MDF)	Failure to encrypt/decrypt data.	No additional information
FDP_IFC_EXT.1(MDF)	None.	No additional information

Requirement	Auditable Events	Additional Audit Record Contents
FDP_STG_EXT.1(MDF)	Addition or removal of certificate from Trust Anchor Database	Subject name of certificate
FIA_ENR_EXT.2(AGENT)	Enrollment in management	Reference identifier of MDM Server
FIA_X509_EXT.1(MDF)	Failure to validate x.509v3 certificate	Reason for failure of validation
FMT_UNR_EXT.1(AGENT)	Attempt to unenroll.	No additional information
FMT_POL_EXT.2(AGENT)	Failure of policy validation.	Reason for failure of validation.
FMT_SMF_EXT.3(AGENT)	Success or failure of function.	No additional information
FPT_NOT_EXT.1(MDF)	**None.**	**No additional information.**
FPT_TST_EXT.1(MDF)	Initiation of self-test	**None.**
	Failure of self-test	
FPT_TST_EXT.2(MDF)	Start-up of TOE	No additional information
	None.	
FPT_TST_EXT.1(WLAN)	Execution of this set of TSF self-test.	No additional information
	None.	
FTA_WSE_EXT.1(WLAN)	All attempts to connect to access points.	Identity of access point being connected to as well as success and failures (including reason for failure)
FTP_ITC_EXT.1/(AGENT) FTP_ITT_EXT.1(AGENT)	Initiation and termination of trusted channel.	Trusted channel protocol. Non-TOE endpoint of connection.
FTP_ITC_EXT.1(WLAN)	All attempts to establish a trusted channel.	Identification of the non-TOE endpoint of the channel.

Table 3: Audit Events

Security Audit Event Selection (FAU_SEL)

FAU_SEL.1(2) Security Audit Event Selection

FAU_SEL.1.1(2)(AGENT)

The TSF shall be able to select the set of events to be audited from the set of all auditable events based on the following attributes:

- event type;
- success of auditable security events;
- failure of auditable security events; and
- none.

Security Audit Event Storage (FAU_STG)

FAU_STG.1 Audit Storage Protection

FAU_STG.1.1(MDF)

The TSF shall protect the stored audit records in the audit trail from unauthorized deletion.

FAU_STG.1.2(MDF)

The TSF shall be able to prevent unauthorized modifications to the stored audit records in the audit trail.

FAU_STG.4 Prevention of Audit Data Loss

FAU_STG.4.1(MDF)

The TSF shall overwrite the oldest stored audit records if the audit trail is full.

6.2 Cryptographic Support (FCS)

Cryptographic Key Management (FCS_CKM)

FCS_CKM.1(1) Cryptographic Key Generation

FCS_CKM.1.1(1)(MDF)

The TSF shall generate asymmetric cryptographic keys in accordance with a specified cryptographic key generation algorithm

- RSA schemes using cryptographic key sizes of 2048-bit or greater that meet FIPS PUB 186-4, "Digital Signature Standard (DSS)", Appendix B.3;
- **ECC schemes using "NIST curves" P-384 and P-256 that meet the following: FIPS PUB 186-4, "Digital Signature Standard (DSS)", Appendix B.4, Curve25519 schemes that meet the following: [RFC7748]**

FCS_CKM.1(2) Cryptographic Key Generation

FCS_CKM.1.1(2)(WLAN)

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

FCS_CKM_EXT.1 Extended Cryptographic Key Support (REK)

FCS_CKM_EXT.1.1(MDF)

The TSF shall support **mutable hardware** REK(s) with a **symmetric** key of strength **256 bits**.

FCS_CKM_EXT.1.2(MDF)

Each REK shall be hardware-isolated from Rich OS on the TSF in runtime.

FCS_CKM_EXT.1.3(MDF)

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

FCS_CKM_EXT.1.4(MDF)

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

Application Note: FCS_CKM_EXT.1.4 is included as required by Annex C.1 of the Protection Profile.

FCS_CKM.2(1) Cryptographic Key Establishment

FCS_CKM.2.1(1)(MDF)

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**.”

FCS_CKM.2(2) Cryptographic Key Establishment (While device is locked)

FCS_CKM.2.1(2)(MDF)

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.”

FCS_CKM.2(3) WLAN Cryptographic Key Distribution (GTK)

FCS_CKM.2.1(3)(WLAN)

The TSF shall decrypt Group Temporal Key in accordance with AES Key Wrap in an EAPOL-Key frame that meets the following: RFC 3394 for AES Key Wrap, 802.11-2012 for the packet format and timing considerations and does not expose the cryptographic keys.

FCS_CKM_EXT.2 Cryptographic Key Random Generation

FCS_CKM_EXT.2.1(MDF)

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

FCS_CKM_EXT.3 Extended: Cryptographic Key Generation

FCS_CKM_EXT.3.1(MDF)

The TSF shall use **symmetric KEKs of 256-bit security strength** corresponding to at least the security strength of the keys encrypted by the KEK.

FCS_CKM_EXT.3.2(MDF)

The TSF shall generate all KEKs using one or more of the following methods:

a) derive the KEK from a Password Authentication Factor using PBKDF and
b) generate the KEK using an RBG that meets this profile (as specified in FCS_RBG_EXT.1).

c) Combine the KEK from other KEKs in a way that preserves the effective entropy of each factor by **concatenating the keys and use a KDF (as described in SP 800-56c), encrypting one key with another**.

Application Note: The random number generator on the main device is used.

Application Note: The selection for item C was modified in accordance with the TRRT response to Technical Query #237

FCS_CKM_EXT.4 Extended: Key Destruction

FCS_CKM_EXT.4.1(MDF)

The TSF shall destroy cryptographic keys in accordance with the specified cryptographic key destruction methods:

- by clearing the KEK encrypting the target key,
- in accordance with the following rules:
- For volatile memory, the destruction shall be executed by a single direct overwrite **consisting of zeroes**.
- For non-volatile EEPROM, the destruction shall be executed by a single direct overwrite consisting of a pseudo random pattern using the TSF's RBG (as specified in FCS_RBG_EXT.1), followed by a read-verify.
- For non-volatile flash memory that is not wear-leveled, the destruction shall be executed by a block erase **that erases the reference to memory that stores data as well as the data itself.**
- For non-volatile flash memory that is wear-leveled, the destruction shall be executed **by a block erase**.
- For non-volatile memory other than EEPROM and flash, the destruction shall be executed by overwriting three or more times with a random pattern that is changed before each write.

FCS_CKM_EXT.4.2(MDF)

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

FCS_CKM_EXT.5 Extended: TSF Wipe

FCS_CKM_EXT.5.1(MDF)

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(MDF)

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

FCS_CKM_EXT.6 Extended: Salt Generation

FCS_CKM_EXT.6.1(MDF)

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

Application Note: the salt is generated using the random number generator implemented in the secure enclave, which like the one implemented in the main *device, satisfies the requirements of FCS_RBG_EXT.1. A proprietary Entropy Assessment Report (EAR) has been provided to NIAP that gives details of both random number generators.*

Cryptographic Operations (FCS_COP)

FCS_COP.1(1): Confidentiality Algorithms

FCS_COP.1.1(1)(MDF)

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-CCM (as defined in NIST SP 800-38C),**

and cryptographic key sizes 128-bit key sizes and **256-bit key sizes**.

FCS_COP.1(2): Hashing Algorithms

FCS_COP.1.1(2)(MDF)

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

FCS_COP.1(3): Signature Algorithms

FCS_COP.1.1(3)(MDF)

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 and P-384 that meet the following: FIPS PUB 186-4, “Digital Signature Standard (DSS)”, Section 5.**

FCS_COP.1(4) Keyed Hash Algorithms

FCS_COP.1.1(4)(MDF)

The TSF shall perform keyed-hash message authentication in accordance with a specified cryptographic algorithm HMAC-SHA-1 and **HMAC-SHA-256, HMAC-SHA-384, HMAC-SHA-512** and cryptographic key sizes **128 to 256 bit** and message digest sizes 160 and **256, 384, 512** bits that meet the following: FIPS Pub 198-1, "The Keyed-Hash Message Authentication Code, and FIPS Pub 180-4, “Secure Hash Standard".

FCS_COP.1(5) Password-Based Key Derivation Functions

FCS_COP.1.1(5)(MDF)

The TSF shall perform Password-based Key Derivation Functions in accordance with a specified cryptographic algorithm HMAC-**SHA-1**, with ***a minimum of 10,000*** iterations, and output cryptographic key sizes **256** that meet the following: NIST SP 800-132.

Application note: The number of iterations is calibrated to take at least 80 milliseconds and is a minimum of 10,000. The number of iterations may be greater in some devices.

HTTPS Protocol (FCS_HTTPS)

FCS_HTTPS_EXT.1 Extended: HTTPS Protocol

FCS_HTTPS_EXT.1.1(MDF)

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

FCS_HTTPS_EXT.1.2(MDF)

The TSF shall implement HTTPS using TLS (FCS_TLSC_EXT.1).

FCS_HTTPS_EXT.1.3(MDF)

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

Initialization Vector Generation (FCS_IV)

FCS_IV_EXT.1 Extended: Initialization Vector Generation

FCS_IV_EXT.1.1(MDF)

The TSF shall generate IVs in accordance with Table 15 ***defined in MDFPP***: References and IV Requirements for NIST-approved Cipher Modes.

Random Bit Generation (FCS_RBG)

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

FCS_RBG_EXT.1.1(MDF)

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

FCS_RBG_EXT.1.2(MDF)

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(MDF)

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

Cryptographic Algorithm Services (FCS_SRV)

FCS_SRV_EXT.1 Extended: Cryptographic Algorithm Services

FCS_SRV_EXT.1.1(MDF)

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

- All mandatory and **selected algorithms with the exception of ECC over curve 25519-based algorithms** in FCS_CKM.2(2)
- The following algorithms in FCS_COP.1(1): AES-CBC
- 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)

Cryptographic Key Storage (FCS_STG)

FCS_STG_EXT.1 Extended: Secure Key Storage

FCS_STG_EXT.1.1(MDF)

The TSF shall provide **software-based** secure key storage for asymmetric private keys and **symmetric keys, persistent secrets**.

FCS_STG_EXT.1.2(MDF)

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

FCS_STG_EXT.1.3(MDF)

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

FCS_STG_EXT.1.4(MDF)

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(MDF)

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

FCS_STG_EXT.2 Extended: Encrypted Cryptographic Key Storage

FCS_STG_EXT.2.1(MDF)

The TSF shall encrypt all DEKs and KEKs and **all software-based key storage** by KEKs that are

1) **Protected by the REK with**
   a. **encryption by a KEK chaining to a REK,**
2) **Protected by the REK and the password with**

b. encryption by a KEK chaining to a REK and the password-derived KEK.

FCS_STG_EXT.2.2(MDF)

DEKs and KEKs and **all software-based key storage** shall be encrypted using one of the following methods: **AES in the Key Wrap (KW) mode, CTR.**

FCS_STG_EXT.3 Extended: Integrity of Encrypted Key Storage

FCS_STG_EXT.3.1(MDF)

The TSF shall protect the integrity of any encrypted DEKs and KEKs and **no other keys** by **an immediate application of the key for decrypting the protected data followed by a successful verification of the decrypted data with a previously known information.**

FCS_STG_EXT.3.2(MDF)

The TSF shall verify the integrity of the **MAC** of the stored key prior to use of the key.

FCS_STG_EXT.4 Extended:

FCS_STG_EXT.4.1(AGENT)

The MDM Agent shall store persistent secrets and private keys when not in use in platform-provided key storage.

TLS Client Protocol (FCS_TLSC)

FCS_TLSC_EXT.1 Extended: TLS Protocol

FCS_TLSC_EXT.1.1(MDF)

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 PP_MD_V3.0:***
  - **TLS_RSA_WITH_AES_256_CBC_SHA as defined in RFC 5246**
  - **TLS_RSA_WITH_AES_128_CBC_SHA256 as defined in RFC 5246**
  - **TLS_RSA_WITH_AES_256_CBC_SHA256 as defined in RFC 5246**
  - **TLS_DHE_RSA_WITH_AES_128_CBC_SHA256 as defined in RFC 5246**
  - **TLS_DHE_RSA_WITH_AES_256_CBC_SHA256 as defined in RFC 5246**
  - **TLS_ECDHE_ECDSA_WITH_AES_128_CBC_SHA256 as defined in RFC 5289**
  - **TLS_ECDHE_ECDSA_WITH_AES_256_CBC_SHA384 as defined in RFC 5289**
  - **TLS_ECDHE_ECDSA_WITH_AES_128_GCM_SHA256 as defined in RFC 5289**
  - **TLS_ECDHE_ECDSA_WITH_AES_128_GCM_SHA384 as defined in RFC 5289**

FCS_TLSC_EXT.1.2(MDF)

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

FCS_TLSC_EXT.1.3(MDF)

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

FCS_TLSC_EXT.1.4(MDF)

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

FCS_TLSC_EXT.1/WLAN Extensible Authentication Protocol-Transport Layer Security (EAP-TLS)

FCS_TLSC_EXT.1.1(WLAN)

The TSF shall implement TLS 1.0 and **TLS 1.2 (RFC 5246)** in support of the EAP-TLS protocol as specified in RFC 5216 supporting the following ciphersuites:

- Mandatory Ciphersuites in accordance with RFC 5246:
    - TLS_RSA_WITH_AES_128_CBC_SHA as defined in RFC 5246
- Optional Ciphersuites:
    - **TLS_RSA_WITH_AES_256_CBC_SHA as defined in RFC 5246**

FCS_TLSC_EXT.1.2(WLAN)

The TSF shall generate random values used in the EAP-TLS exchange using the RBG specified in FCS_RBG_EXT.1.

FCS_TLSC_EXT.1.3(WLAN)

The TSF shall use X509 v3 certificates as specified in FIA_X509_EXT.1

FCS_TLSC_EXT.1.4(WLAN)

The TSF shall verify that the server certificate presented includes the Server Authentication purpose (id-kp 1 with OID 1.3.6.1.5.5.7.3.1) in the extendedKeyUsage field.

FCS_TLSC_EXT.1.5(WLAN)

The TSF shall allow an authorized administrator to configure the list of CAs that are allowed to sign authentication server certificates that are accepted by the TOE.

FCS_TLSC_EXT.1.6/(WLAN)

The TSF shall allow an authorized administrator to configure the list of algorithm suites that may be proposed and accepted during the EAP-TLS exchanges.

6.3 User Data Protection (FDP)

Access Control (FDP_ACF)

FDP_ACF_EXT.1 Extended: Security Access Control

FDP_ACF_EXT.1.1(MDF)

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

FDP_ACF_EXT.1.2(MDF)

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

Data-At-Rest Protection (FDP_DAR)

FDP_DAR_EXT.1 Extended: Protected Data Encryption

FDP_DAR_EXT.1.1(MDF)

Encryption shall cover all protected data.

FDP_DAR_EXT.1.2(MDF)

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

FDP_DAR_EXT.2 Extended: Sensitive Data Encryption

FDP_DAR_EXT.2.1(MDF)

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

FDP_DAR_EXT.2.2(MDF)

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

FDP_DAR_EXT.2.3(MDF)

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

FDP_DAR_EXT.2.4(MDF)

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

Subset Information Flow Control - VPN (FDP_IFC)

FDP_IFC_EXT.1 Extended: Subset Information Flow Control

FDP_IFC_EXT.1.1(MDF)

The TSF shall **provide a VPN client which can protect all IP traffic using IPSec** with the exception of IP traffic required to establish the VPN connection.

Application Note: The VPN client shall be validated in a separate evaluation.

Certificate Data Storage (FDP_STG)

FDP_STG_EXT.1 Extended: User Data Storage

FDP_STG_EXT.1.1(MDF)

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

Inter-TSF User Data Protected Channel (FDP_UPC)

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

FDP_UPC_EXT.1.1(MDF)

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(MDF)

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

6.4 Identification and Authentication (FIA)

Authentication Failures (FIA_AFL)

FIA_AFL_EXT.1 Authentication failure handling

FIA_AFL_EXT.1.1(MDF)

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

FIA_AFL_EXT.1.2(MDF)

The TSF shall detect when a configurable positive integer within **2 to 10** of **non-unique** unsuccessful authentication attempts occur related to last successful authentication for each authentication mechanism.

FIA_AFL_EXT.1.3(MDF)

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

FIA_AFL_EXT.1.4(MDF)

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

FIA_AFL_EXT.1.5(MDF)

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

FIA_AFL_EXT.1.6(MDF)

The TSF shall increment the number of unsuccessful authentication attempts prior to notifying the user that the authentication was unsuccessful.

Bluetooth Authorization and Authentication (FIA_BLT)

FIA_BLT_EXT.1 Extended: Bluetooth User Authorization

FIA_BLT_EXT.1.1(MDF)

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

FIA_BLT_EXT.2 Extended: Bluetooth Mutual Authentication

FIA_BLT_EXT.2.1(MDF)

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

FIA_BLT_EXT.3 Extended: Rejection of Duplicate Bluetooth Connections

FIA_BLT_EXT.3.1(MDF)

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

FIA_BLT_EXT.4 Extended: Secure Simple Pairing

FIA_BLT_EXT.4.1(MDF)

The TOE shall support Bluetooth Secure Simple Pairing, both in the host and the controller. Furthermore, Secure Simple Pairing shall be used during the pairing process if the remote device also supports it.

Enrollment of Mobile Device into Management (FIA_ENR)

FIA_ENR_EXT.2 Extended: Enrollment of Mobile Device into Management

FIA_ENR_EXT.2.1(AGENT)

The MDM Agent shall record the reference identifier of the MDM Server during the enrollment process.

Port Access Entity Authentication (FIA_PAE)

FIA_PAE_EXT.1 Extended: PAE Authentication

FIA_PAE_EXT.1.1(WLAN)

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

Password Management (FIA_PMG)

FIA_PMG_EXT.1 Extended: Password Management

FIA_PMG_EXT.1.1(MDF)

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.

Authentication Throttling (FIA_TRT)

FIA_TRT_EXT.1 Extended: Authentication Throttling

FIA_TRT_EXT.1.1(MDF)

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.

User Authentication (FIA_UAU)

FIA_UAU.5 Multiple Authentication Mechanisms

FIA_UAU.5.1(MDF)

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

FIA_UAU.5.2(MDF)

The TSF shall authenticate any user's claimed identity according to the **validation of the user's password.**

FIA_UAU.6(1) Re-Authentication

FIA_UAU.6.1(1)(MDF)

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

FIA_UAU.6(2) Re-Authentication

FIA_UAU.6.1(2)(MDF)

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

FIA_UAU.7 Protected authentication feedback

FIA_UAU.7.1(MDF)

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

FIA_UAU_EXT.1 Extended: Authentication for Cryptographic Operation

FIA_UAU_EXT.1.1(MDF)

The TSF shall require the user to present the Password Authentication Factor prior to decryption of protected data and encrypted DEKs, KEKs and **all software-based key storage** at startup.

FIA_UAU_EXT.2 Extended: Timing of Authentication

FIA_UAU_EXT.2.1(MDF)

The TSF shall allow answering calls, make emergency calls, and use the camera (unless its use is generally disallowed) on behalf of the user to be performed before the user is authenticated.

FIA_UAU_EXT.2.2(MDF)

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

X509 Certificates (FIA_X509)

FIA_X509_EXT.1 Extended: Validation of certificates

FIA_X509_EXT.1.1(MDF)

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 **the Online Certificate Status Protocol (OCSP) as specified in RFC 2560**.
- 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(MDF)

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

FIA_X509_EXT.1.2(MDF)

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.

X509 Certificate Authentication

FIA_X509_EXT.2 Extended: X509 certificate authentication

FIA_X509_EXT.2.1(MDF)

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

FIA_X509_EXT.2.2(MDF)

When the TSF cannot establish a connection to determine the revocation status of a certificate, the TSF shall allow the administrator to choose whether to accept the certificate in these cases, allow the user to choose whether to accept the certificate in these cases.

FIA_X509_EXT.2/WLAN Extended: X509 certificate authentication (EAP-TLS)

FIA_X509_EXT.2.1(WLAN)

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

FIA_X509_EXT.2.2(WLAN)

When the TSF cannot establish a connection to determine the validity of a certificate, the TSF shall allow the administrator to choose whether to accept the certificate in these cases, allow the user to choose whether to accept the certificate in these cases.

Request Validation of Certificates

FIA_X509_EXT.3 Extended: Request Validation of certificates

FIA_X509_EXT.3.1(MDF)

The TSF shall provide a certificate validation service to applications.

FIA_X509_EXT.3.2(MDF)

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

6.5 Security Management (FMT)

Management of Functions in TSF (FMT_MOF)

FMT_MOF_EXT.1 Extended: Management of security functions behavior

FMT_MOF_EXT.1.1(MDF)

The TSF shall restrict the ability to perform the functions in column ***3*** of Table ***4 defined in MDPP*** to the user.

FMT_MOF_EXT.1.2(MDF)

The TSF shall restrict the ability to perform the functions in column ***5*** of Table ***4 defined in MDFPP*** to the administrator when the device is enrolled and according to the administrator-configured policy.

Trusted Policy Update

FMT_POL_EXT.2 Trusted Policy Update

FMT_POL_EXT.2.1(AGENT)

The MDM Agent shall only accept policies and policy updates digitally signed by the Enterprise.

FMT_POL_EXT.2.2(AGENT)

The MDM Agent shall not install policies if the policy signing certificate is deemed invalid.

Specification of Management Functions (FMT_SMF)

FMT_SMF_EXT.1 Extended: Specification of Management Functions

FMT_SMF_EXT.1.1(MDF)

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

Management Function	User	Administrator	Administrator, when enrolled in MDM
Function 1: Configure password policy: a. minimum password length b. minimum password complexity c. maximum password lifetime	-	X	X

Management Function	**User**	**Administrator**	**Administrator, when enrolled in MDM**
Function 2: Configure session locking policy: a. screen-lock enabled/disabled b. screen lock timeout c. number of authentication failures	-	X	X
Function 3: Enable/disable the VPN protection: a. across device b. **no other method**	-	X	X
Function 4 Enable/disable **Bluetooth, Wi-Fi, cellular radio**	X	-	-
Function 5: Enable/disable **camera**: a. across device b. **no other method**	-	X	X
Enable/disable camera on a per-app basis	X	-	-
Function 5: **Enable/disable microphone on a per-app basis**	X	-	-
Function 6: Transition to the locked state	-	X	-
Function 7: TSF wipe of protected data	-	X	-
Function 8: Configure application installation policy by c. **denying installation of applications**	-	X	X
Function 9: Import keys/secrets into the secure key storage	-	X	-
Function 10: Destroy imported keys/secrets and **no other keys/secrets** in the secure key storage	-	X	-
Function 11: Import X.509v3 certificates in the Trust Anchor Database	-	X	X

Management Function	User	Administrator	Administrator, when enrolled in MDM
Function 12: Remove imported X509v3 certificates and **no other X509v3 certificates** in the Trust Anchor Database	X	-	-
Function 13: Enroll the TOE in management	X	-	-
Function 14: Remove applications	-	X	X
Function 15: Update system software	-	X	-
Function 16: Install applications	-	X	X
Function 17: Remove Enterprise applications	-	X	-
Function 18: Configure the Bluetooth trusted channel: a. disable/enable the Discoverable mode (for BR/EDR) b. change the Bluetooth device name **c. specify minimum level of security for each pairing.**	X	-	-
Function 19: Enable/disable display notifications in the locked state of: **all notifications**	X	X	-
Function 22: enable/disable location services: a. across device b. **on a per-app basis**	X	X	-
Function 28: Wipe Enterprise data	-	X	X
Function 30: Configure whether to establish a trusted channel or disallow establishment if the TSF cannot establish a connection to determine the validity of a certificate	-	X	X
Function 33: Configure **certificate** used to validate digital signature on application	-	X	X
Function 36: Configure the unlock banner	-	X	X

Table 4: Management Functions

Application note: Most of the administrator management functions are implemented by the specification and installation of Configuration Profiles. Also for the enforcement of other functions, such as the password policy, the installation of Configuration Profiles with dedicated values for some of the payload keys is required.

Application note: With respect to function 3, the TOE allows for both demanding VPN across the device or defining a VPN on a per-App basis. NIAP TD0060 allows both options.

Application note: Function 24 has not been included in the table since the TOE does not support a developer mode.

Application note: Function 20 has not been included in the table since the TOE has data-at-rest protection natively enabled.

Application note: Function 21 has not been included in the table since the TOE does not support removable media. Backups can be made using iTunes, and backup encryption can be made mandatory.

Application note: Function 27 has not been included in the table since the TOE (in its evaluated configuration) does not support bypass of local user authentication.

Application note: Function 32 has not been included in the table since audit review is not implemented on TOE devices.

FMT_SMF_EXT.1/WLAN Specification of Management Functions

FMT_SMF_EXT.1.1(WLAN)

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

- configure security policy for each wireless network:
  - **specify the CA(s) from which the TSF will accept WLAN authentication server certificates(s)**
  - security type
  - authentication protocol
  - client credentials to be used for authentication

FMT_SMF_EXT.2 Extended: Specification of Remediation Actions

FMT_SMF_EXT.2.1(MDF)

The TSF shall offer wipe of all data associated with profiles under management upon unenrollment and when issuing a remote wipe command.

FMT_SMF_EXT.3 Extended: Specification of management Functions

FMT_SMF_EXT.3.1(AGENT)

The MDM Agent shall be capable of interacting with the platform to perform the following functions:

- administrator-provided management functions in MDF PP;
- administrator-provided management functions in MDM PP;
- Import the certificates to be used for authentication of MDM Agent communications
- **no additional functions**

FMT_SMF_EXT.3.2(AGENT)

The MDM Agent shall be capable of performing the following functions:

- Enroll in management;
- Configure whether users can unenroll the agent from management
- **no other functions**

User Unenrollment Prevention

FMT_UNR_EXT.1 Extended: User Unenrollment Prevention

FMT_UNR_EXT.1.1(AGENT)

The MDM Agent shall provide a mechanism to enforce the following behavior upon an attempt to unenroll the mobile device from management: **prevent the unenrollment from occurring.**

6.6 Protection of the TSF (FPT)

Anti-Exploitation Services (FPT_AEX)

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

FPT_AEX_EXT.1.1(MDF)

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

FPT_AEX_EXT.1.2(MDF)

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

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

FPT_AEX_EXT.2.1(MDF)

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

FPT_AEX_EXT.2.2(MDF)

The TSF shall prevent write and execute permissions from being simultaneously granted to any page of physical memory **with no exceptions**.

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

FPT_AEX_EXT.3.1(MDF)

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

FPT_AEX_EXT.4 Extended: Domain Isolation

FPT_AEX_EXT.4.1(MDF)

The TSF shall protect itself from modification by untrusted subjects.

FPT_AEX_EXT.4.2(MDF)

The TSF shall enforce isolation of address space between applications.

JTAG Disablement (FPT_JTA)

FPT_JTA_EXT.1 Extended: JTAG Disablement

FPT_JTA_EXT.1.1(MDF)

The TSF shall disable access through hardware to JTAG.

Key Storage (FPT_KST)

FPT_KST_EXT.1 Extended: Key Storage

FPT_KST_EXT.1.1(MDF)

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

FPT_KST_EXT.2 Extended: No Key Transmission

FPT_KST_EXT.2.1(MDF)

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

FPT_KST_EXT.3 Extended: No Plaintext Key Export

FPT_KST_EXT.3.1(MDF)

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

Self-Test Notification (FPT_NOT)

FPT_NOT_EXT.1 Extended: Self-Test Notification

FPT_NOT_EXT.1.1(MDF)

The TSF shall transition to non-operational mode and **no other action** when the following types of failures occur:

- failures of the self-test(s)
- TSF software integrity verification failures
- **no other failures**.

Reliable Time Stamps (FPT_STM)

FPT_STM.1 Reliable time stamps

FPT_STM.1.1(MDF)

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

TSF Functionality Testing (FPT_TST)

FPT_TST_EXT.1 Extended: TSF Cryptographic Functionality Testing

FPT_TST_EXT.1.1(MDF)

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.

FPT_TST_EXT.1/WLAN TSF Cryptographic Functionality Testing (Wireless LAN)

FPT_TST_EXT.1.1(WLAN)

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

FPT_TST_EXT.1.2(WLAN)

The **TOE** shall provide the capability to verify the integrity of stored TSF executable code when it is loaded for execution through the use of the TSF-provided cryptographic services.

TSF Integrity Testing

FPT_TST_EXT.2 Extended: TSF Integrity Testing

FPT_TST_EXT.2.1(MDF)

The TSF shall verify the integrity of the bootchain up through the Application Processor OS kernel, **and baseband firmware** stored in mutable media prior to its execution through the use of **a digital signature using a hardware-protected asymmetric key**.

FPT_TST_EXT.2.2(MDF)

The TSF shall not execute code if the code signing certificate is deemed invalid.

Application Note: FPT_TST_EXT.2.2 has been included as required by Annex C.4 of the Protection Profile.

Trusted Update (FPT_TUD)

FPT_TUD_EXT.1 Extended: Trusted Update: TSF version query

FPT_TUD_EXT.1.1(MDF)

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

FPT_TUD_EXT.1.2(MDF)

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(MDF)

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

Trusted Update Verification (FPT_TUD_EXT)

FPT_TUD_EXT.2 Extended: Trusted Update Verification

FPT_TUD_EXT.2.1(MDF)

The TSF shall verify software updates to the Application Processor system software and **no other processor system software** using a digital signature by the manufacturer prior to installing those updates.

FPT_TUD_EXT.2.2(MDF)

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

FPT_TUD_EXT.2.3(MDF)

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(MDF)

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

FPT_TUD_EXT.2.5(MDF)

The TSF shall by default only install mobile applications cryptographically verified by **a built-in X.509v3 certificate**.

FPT_TUD_EXT.2.6(MDF)

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

6.7 TOE Access (FTA)

Session Locking (FTA_SSL)

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

FTA_SSL_EXT.1.1(MDF)

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

FTA_SSL_EXT.1.2(MDF)

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

FTA_SSL_EXT.1.3(MDF)

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

a) clearing or overwriting display devices, obscuring the previous contents;
b) zeroize the decrypted class key for the NSFileProtectionComplete class.

Wireless Network Access (FTA_WSE)

FTA_WSE_EXT.1 Wireless Network Access

FTA_WSE_EXT.1.1(WLAN)

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

6.8 Trusted Path/Channels (FTP)

Trusted Channel Communication (FTP_ITC)

FTP_ITC_EXT.1(1) Extended: Trusted channel Communication

FTP_ITC_EXT.1.2(1)(MDF)

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

FTP_ITC_EXT.1.3(1)(MDF)

The TSF shall initiate communication via the trusted channel for wireless access point connections, administrative communication, configured enterprise connections, and **OTA updates.**

FTP_ITC_EXT.1(2) Extended: Trusted channel Communication

FTP_ITC_EXT.1.1(2)(AGENT)

The TSF shall use **HTTPS** 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(2)(AGENT)

The TSF shall permit the TSF and the MDM Server and **no other IT entities** to initiate communication via the trusted channel.

FTP_ITC_EXT.1.3(2)(AGENT)

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

FTP_ITC_EXT.1(3) Extended: Trusted channel Communication

FTP_ITC_EXT.1.1(3)(WLAN)

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

FTP_ITC_EXT.1.2(3)(WLAN)

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

6.9 Objective Requirements

Default TOE Access Banners (FTA_TAB)

FTA_TAB.1 Default TOE Access Banners

FTA_TAB.1.1(MDF)

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

6.10 Security Function Requirements Rationale

The requirements in the PP are assumed to represent a complete set of requirements that serve to address any interdependencies. Given that all of the appropriate functional requirements given in the PPs have been copied into this [ST], the dependency analysis for the requirements is assumed to be already performed by the PP authors and is not reproduced in this document.

7 Security Assurance Requirements

The Security Assurance Requirements (SARs) for the TOE are defined in [PP_MD_V3.0]. They consist of the assurance components of Evaluation Assurance Level (EAL1) as defined in part 3 of the CC augmented by ASE_SPD.1 and ALC_TSU_EXT.1, which is defined in [PP_MD_V3.0]. These security assurance requirements are also applicable to the EP [EP_MDM_AGENT_V3.0].

The assurance components included in [PP_MD_V3.0] are:

- ASE_CCL.1
- ASE_ECD.1
- ASE_INT.1
- ASE_OBJ.1
- ASE_REQ.1
- ASE_SPD.1
- ASE_TSS.1
- ADV_FSP.1
- AGD_OPE.1
- AGD_PRE.1
- ALC_CMC.1
- ALC_CMS.1
- ALC_TSU_EXT.1
- ATE_IND.1
- AVA_VAN.1

7.1 Security Target Evaluation (ASE)

7.1.1 Conformance Claims (ASE_CCL.1)

ASE_CCL.1.1D

The developer shall provide a conformance claim.

ASE_CCL.1.2D

The developer shall provide a conformance claim rationale.

ASE_CCL.1.1C

The conformance claim shall contain a CC conformance claim that identifies the version of the CC to which the ST and the TOE claim conformance.

ASE_CCL.1.2C

The CC conformance claim shall describe the conformance of the ST to CC Part 2 as either CC Part 2 conformant or CC Part 2 extended.

ASE_CCL.1.3C

The CC conformance claim shall describe the conformance of the ST to CC Part 3 as either CC Part 3 conformant or CC Part 3 extended.

ASE_CCL.1.4C

The CC conformance claim shall be consistent with the extended components definition.

ASE_CCL.1.5C

The conformance claim shall identify all PPs and security requirement packages to which the ST claims conformance.

ASE_CCL.1.6C

The conformance claim shall describe any conformance of the ST to a package as either package-conformant or package-augmented.

ASE_CCL.1.7C

The conformance claim rationale shall demonstrate that the TOE type is consistent with the TOE type in the PPs for which conformance is being claimed.

ASE_CCL.1.8C

The conformance claim rationale shall demonstrate that the statement of the security problem definition is consistent with the statement of the security problem definition in the PPs for which conformance is being claimed.

ASE_CCL.1.9C

The conformance claim rationale shall demonstrate that the statement of security objectives is consistent with the statement of security objectives in the PPs for which conformance is being claimed.

ASE_CCL.1.10C

The conformance claim rationale shall demonstrate that the statement of security requirements is consistent with the statement of security requirements in the PPs for which conformance is being claimed.

ASE_CCL.1.1E

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

7.1.2 Extended Components Definition (ASE_ECD.1)

ASE_ECD.1.1D

The developer shall provide a statement of security requirements.

ASE_ECD.1.2D

The developer shall provide an extended components definition.

ASE_ECD.1.1C

The statement of security requirements shall identify all extended security requirements.

ASE_ECD.1.2C

The extended components definition shall define an extended component for each extended security requirement.

ASE_ECD.1.3C

The extended components definition shall describe how each extended component is related to the existing CC components, families, and classes.

ASE_ECD.1.4C

The extended components definition shall use the existing CC components, families, classes, and methodology as a model for presentation.

ASE_ECD.1.5C

The extended components shall consist of measurable and objective elements such that conformance or nonconformance to these elements can be demonstrated.

ASE_ECD.1.1E

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

ASE_ECD.1.2E

The evaluator shall confirm that no extended component can be clearly expressed using existing components.

7.1.3 ST Introduction (ASE_INT.1)

ASE_INT.1.1D

The developer shall provide an ST introduction.

ASE_INT.1.1C

The ST introduction shall contain an ST reference, a TOE reference, a TOE overview and a TOE description.

ASE_INT.1.2C

The ST reference shall uniquely identify the ST.

ASE_INT.1.3C

The TOE reference shall identify the TOE.

ASE_INT.1.4C

The TOE overview shall summarise the usage and major security features of the TOE.

ASE_INT.1.5C

The TOE overview shall identify the TOE type.

ASE_INT.1.6C

The TOE overview shall identify any non-TOE hardware/software/firmware required by the TOE.

ASE_INT.1.7C

The TOE description shall describe the physical scope of the TOE.

ASE_INT.1.8C

The TOE description shall describe the logical scope of the TOE.

ASE_INT.1.1E

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

ASE_INT.1.2E

The evaluator shall confirm that the TOE reference, the TOE overview, and the TOE description are consistent with each other.

7.1.4 Security Objectives for the Operational Environment (ASE_OBJ.1)

ASE_OBJ.1.1D

The developer shall provide a statement of security objectives.

ASE_OBJ.1.1C

The statement of security objectives shall describe the security objectives for the operational environment.

ASE_OBJ.1.1E

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

7.1.5 Stated Security Requirements (ASE_REQ.1)

ASE_REQ.1.1D

The developer shall provide a statement of security requirements.

ASE_REQ.1.2D

The developer shall provide a security requirements rationale.

ASE_REQ.1.1C

The statement of security requirements shall describe the SFRs and the SARs.

ASE_REQ.1.2C

All subjects, objects, operations, security attributes, external entities and other terms that are used in the SFRs and the SARs shall be defined.

ASE_REQ.1.3C

The statement of security requirements shall identify all operations on the security requirements.

ASE_REQ.1.4C

All operations shall be performed correctly.

ASE_REQ.1.5C

Each dependency of the security requirements shall either be satisfied, or the security requirements rationale shall justify the dependency not being satisfied.

ASE_REQ.1.6C

The statement of security requirements shall be internally consistent.

ASE_REQ.1.1E

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

7.1.6 Security Problem Definition (ASE_SPD.1)

ASE_SPD.1.1D

The developer shall provide a security problem definition.

ASE_SPD.1.1C

The security problem definition shall describe the threats.

ASE_SPD.1.2C

All threats shall be described in terms of a threat agent, an asset, and an adverse action.

ASE_SPD.1.3C

The security problem definition shall describe the OSPs.

ASE_SPD.1.4C

The security problem definition shall describe the assumptions about the operational environment of the TOE.

ASE_SPD.1.1E

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

7.1.7 TOE Summary Specification (ASE_TSS.1)

ASE_TSS.1.1D

The developer shall provide a TOE summary specification.

ASE_TSS.1.1C

The TOE summary specification shall describe how the TOE meets each SFR.

ASE_TSS.1.1E

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

ASE_TSS.1.2E

The evaluator shall confirm that the TOE summary specification is consistent with the TOE overview and the TOE description.

7.2 Development (ADV)

7.2.1 Basic Functional Specification (ADV_FSP.1)

ADV_FSP.1.1d

The developer shall provide a functional specification.

ADV_FSP.1.2d

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

ADV_FSP.1.1c

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

ADV_FSP.1.2c

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

ADV_FSP.1.3c

The functional specification shall provide rationale for the implicit categorisation of interfaces as SFR-non-interfering.

ADV_FSP.1.4c

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

ADV_FSP.1.1e

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

ADV_FSP.1.2e

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

7.3 Guidance documents (AGD)

7.3.1 Operational User Guidance (AGD_OPE.1)

AGD_OPE.1.1d

The developer shall provide operational user guidance.

AGD_OPE.1.1c

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

AGD_OPE.1.2c

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

AGD_OPE.1.3c

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

AGD_OPE.1.4c

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

AGD_OPE.1.5c

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

AGD_OPE.1.6c

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

AGD_OPE.1.7c

The operational user guidance shall be clear and reasonable.

AGD_OPE.1.1e

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

7.3.2 Preparative Procedures (AGD_PRE.1)

AGD_PRE.1.1d

The developer shall provide the TOE including its preparative procedures.

AGD_PRE.1.1c

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

AGD_PRE.1.2c

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

AGD_PRE.1.1e

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

AGD_PRE.1.2e

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

7.4 Life-cycle support (ALC)

7.4.1 Labelling of the TOE (ALC_CMC.1)

ALC_CMC.1.1d

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

ALC_CMC.1.1c

The TOE shall be labelled with its unique reference.

ALC_CMC.1.1e

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

7.4.2 TOE CM Coverage (ALC_CMS.1)

ALC_CMS.2.1d

The developer shall provide a configuration list for the TOE.

ALC_CMS.2.1c

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

ALC_CMS.2.2c

The configuration list shall uniquely identify the configuration items.

ALC_CMS.2.1e

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

7.4.3 Timely Security Updates (ALC_TSU_EXT.1)

ALC_TSU_EXT.1.1d

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

ALC_TSU_EXT.1.1c

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

ALC_TSU_EXT.1.2c

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

ALC_TSU_EXT.1.3c

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

ALC_TSU_EXT.1.4c

The description shall include where users can seek information about the availability of new updates including details (e.g. CVE identifiers) of the specific public vulnerabilities correct by each update.

ALC_TSU_EXT.1.1e

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

7.5 Tests (ATE)

7.5.1 Independent Testing - Conformance (ATE_IND.1)

ATE_IND.1.1d

The developer shall provide the TOE for testing.

ATE_IND.1.1c

The TOE shall be suitable for testing.

ATE_IND.1.1e

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

ATE_IND.1.2e

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

7.6 Vulnerability assessment (AVA)

7.6.1 Vulnerability Survey (AVA_VAN.1)

AVA_VAN.1.1d

The developer shall provide the TOE for testing.

AVA_VAN.1.1c

The TOE shall be suitable for testing.

AVA_VAN.1.1e

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

AVA_VAN.1.2e

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

AVA_VAN.1.3e

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

8 TOE Summary Specification (TSS)

This chapter describes the relevant aspects of how the security functional requirements are implemented in the security functionality provided by the TOE. This chapter is structured in accordance with the structuring of the security functional requirements in chapter 6 of this document, which in turn has been taken from the structure of the description of the security functional requirements in [PP_MD_V3.0], [PP_MD_AGENT_3.0] and [PP_WLAN__CLI_EP_V1.0].

8.1 Hardware Protection Functions

8.1.1 The Secure Enclave

The Secure Enclave is a coprocessor fabricated in the Apple A7, A8, A8X, A9, A9x and A10 processors. It utilizes its own secure boot and personalized software update separate from the application processor. It provides all cryptographic operations for Data Protection key management and maintains the integrity of Data Protection even if the kernel has been compromised.

The Secure Enclave uses encrypted memory and includes a hardware random number generator. Its microkernel is based on the L4 family, a second-generation microkernel generally used to implement UNIX-like operating systems, with modifications by Apple. Communication between the Secure Enclave and the application processor is isolated to an interrupt-driven mailbox and shared memory data buffers. Note that only a small dedicated amount of memory used for communication between the secure enclave and the main system is shared. The main system has no access to other memory areas of the secure enclave and no keys or key material may be exported.

Each Secure Enclave is provisioned during fabrication with its own Unique ID (UID) that is not accessible to other parts of the system and is not known to Apple. When the device starts up, an ephemeral key is created, entangled with its UID, and used to encrypt the Secure Enclave’s portion of the device’s memory space.

Additionally, data that is saved to the file system by the Secure Enclave is encrypted with a key entangled with the UID and an anti-replay counter.

The UID also serves as the REK for the whole device.

In addition to the UID also the Group Key (GID) and Apple's root certificate are provisioned during manufacturing. The GID is only unique per device type and is used in the secure software update process. Apple's root certificate is used to verify the integrity and authenticity of software during the secure boot process and for updates of the system software.

The Secure Enclave has its own physical noise source and random number generator which is used for generating the salt value for the password-based key generation function (PBKDF) 2, which uses AES with the device UID as the key for the pseudorandom function (PRF). The counter is calculated such that it takes at least 80 milliseconds to execute the function. While the counter value therefore may vary per device type, it is larger than 10,000 for each device type. The salt (128 bit) is regenerated every time the password changes. The salt value is stored AES encrypted with the UID in the system keybag.

Other salt values used for functions in iOS are generated using the True Random Number Generator (TRNG) of the application processor. This includes nonces used in the generation of DSA signatures as well as nonces required for the Wi-Fi and TLS protocol.

8.1.2 Memory Protection

iOS uses the read and write protection for memory pages provided by the advanced Reduced Instruction Set (RISC) machine (ARM) processor for separating applications from the kernel and to provide a sandbox for each application.

Further protection is provided by iOS using ARM's Execute Never (XN) feature, which marks memory pages as non-executable. Memory pages marked as both writable and executable can be used only by apps under tightly controlled conditions: the kernel checks for the presence of the Apple-only dynamic code-signing entitlement. Even then, only a single mmap call can be made to request an executable and writable page, which is given a randomized address.

8.2 Cryptographic Support

8.2.1 Overview of Key Management

Each TOE comes with a unique 256-bit AES key called the UID. This key is stored in the Secure Enclave and is not accessible by the "regular" processor.

Even the software in the Secure Enclave cannot read the UID. It can only request encryption and decryption operations performed by a dedicated AES engine accessible only from the Secure Enclave.

The UID itself is generated outside of the device in the production environment using a protected system with a random number generator that complies with the requirements of NIST Special Publication 800-90A and is seeded appropriately. (FCS_RBG_EXT.1.1 and FCS_RBG_EXT.1.2 for the random number generator used to generate the REK).

The UID is used to derive two other keys, called "key 0x89B" and "key 0x835" in accordance with the requirements of NIST SP 800-38F. Both keys are derived during the first boot by encrypting defined constants with the UID. Those keys then are used to wrap two other keys: the "EMF key" (the file system master key, wrapped by "key 0x89B") and the "DKey" (the device key, wrapped by "key 0x835"). Both are stored in block 0 of the flash memory, which is also called the "effaceable storage". This area of flash memory can be wiped very quickly. Both keys are generated using the random number generator of the secure enclave (post-processed by the CTR_DRBG) when iOS is first installed or after the device has been wiped.

All keys are generated using an internal entropy source, post-processed by a deterministic random number generator (DRNG) (CTR_DRBG). System entropy is generated from timing variations during boot, and additionally from interrupt timing once the device has booted. Keys generated inside the Secure Enclave use its true hardware random number generator based on multiple ring oscillators post processed with CTR_DRBG.

The EMF key is used as a master key used for the encryption of file system metadata. The EMF key is generated using the random number generator of the secure enclave (post-processed by the CTR_DRBG) when iOS is first installed or after the device has been wiped. Also all class keys are generated in the secure enclave and passed to the iOS kernel in wrapped form only.

The Dkey is used within the key hierarchy to directly wrap the class keys that can be used when the device is locked. For class keys that can only be used when the device is unlocked the class keys are wrapped with the XOR of the Dkey and the passcode key.

Every time a file on the data partition is created, a new 256-bit AES key (the "per-file" key) is created using the hardware random number generator of the secure enclave post-processed by CTR_DBRG. Files are encrypted using this key with AES in cipher block chaining (CBC) mode where the initialization vector (IV) is calculated with the block offset into the file, encrypted with the secure hash algorithm (SHA) -1 hash of the per-file key. (FCS_RBG_EXT.1 for the data encryption keys).

Each per-file key is wrapped (in the secure enclave) with the class key of the file's class and then stored in the metadata of the file. Key wrapping uses AES key wrapping per RFC 3394.

Class keys themselves are wrapped either with device key only (for the class NSFileProtectionNone) or are wrapped with a key derived from the device key and the passcode key using XOR. This key wrapping is also performed within the secure enclave.

Each file belongs to one of the following classes with its associated class key.

NSFileProtectionComplete

The class key is protected with a key derived from the user passcode and the device UID. Shortly after the user locks a device (10 seconds, if the "Require Password" setting is 'Immediately'), the decrypted class key is erased, rendering all data in this class inaccessible until the user enters the passcode again.

NSFileProtectionCompleteUnlessOpen

Some files may need to be written while the device is locked. A good example of this is a mail attachment downloading in the background. This behavior is achieved by using asymmetric elliptic curve cryptography (ECDH over Curve25519). iOS implements this by generating a device-wide asymmetric key pair and then protects the private key of this pair by encrypting it with the class key for the NSFileProtectionCompleteUnlessOpen class. Note that this class key can only be unwrapped when the device is unlocked since it requires the passcode to be entered which then is used in the key derivation function (KDF) for that generates the key encryption key (KEK) for this class key as described above. The device-wide asymmetric key pair is generated within the secure enclave.

When receiving data to be protected when the device is in the locked state, the application can create a file with the file attribute NSFileProtectionCompleteUnlessOpen. In this case iOS generates another asymmetric key pair within the secure enclave (per file object used to store the data). The device-wide public key and the file object private key are then used to generate a shared secret (using one-pass DH (Diffie-Hellman) as described in NIST SP 800-56A). The KDF is Concatenation Key Derivation Function (Approved Alternative 1) as described in 5.8.1 of NIST SP 800-56A. AlgorithmID is omitted. PartyUInfo and PartyVInfo are the ephemeral and static public keys, respectively. SHA-256 is used as the hashing function. The key generated in that fashion is used as the symmetric key to encrypt the data. The object private key and the shared secret are cleared when the file is closed and only the object public key is stored with the file object.

To read the file, the per file object shared secret is regenerated using the device-wide private key and the per file object public key.

Unwrapping of the device-wide private key can only be performed when the correct passcode has been entered, since the device-wide private key is wrapped with a key that can only be unwrapped with a class key that itself can only be unwrapped when the passcode is available. The guidance given in section D.3.3 of MDFPP Version 3 allows a key agreement scheme to be used. The key agreement scheme implemented uses using elliptic curve Diffie Hellman (ECDH) over Curve25519. When the correct passcode has been entered, the files with sensitive data received while the device was in the locked state get their per-file key re-wrapped with the NSFileProtectionCompleetUnlessOpenclass key. It is up to the application to check when the device is unlocked and then cause iOS to re-wrap the file encryption key with the class key for the NSFileProtectionComplete class by changing the file's NSFileProtectionKey attribute to NSFileProtectionComplete.

Protected Until First User Authentication

This class behaves in the same way as Complete Protection, except that the decrypted class key is not removed from memory when the device is locked. The protection in this class has similar properties to desktop full-volume encryption, and protects data from attacks that involve a reboot. This is the default class for all third-party app data not otherwise assigned to a Data Protection class.

NSFileProtectionNone

This class key is wrapped only with the device key, and is kept in Effaceable Storage. Since all the keys needed to decrypt files in this class are stored on the device, the encryption only affords the benefit of fast remote wipe. If a file is not assigned a Data Protection class, it is still stored in encrypted form (as is all data on an iOS device).

Keychain data is protected using a class structure similar to the one used for files. Those classes have behaviors equivalent to the file Data Protection classes but use distinct keys.

In addition, there are keychain classes with the additional extension "ThisDeviceOnly". Class keys for those classes are wrapped with a key that is also derived from the Device Key which, when copied from a device during backup and restored on a different device will make them useless.

The keys for both file and keychain Data Protection classes are collected and managed in keybags. iOS uses the following four keybags: system, backup, escrow, and iCloudBackup. The keys stored in the System keybag and some keys stored in the Escrow keybag, which are used for device update and by MDM, are relevant for functions defined in [PP_MD_V3.0].

The system keybag is where the wrapped class keys used in normal operation of the device are stored. For example, when a passcode is entered, the NSFileProtectionComplete key is loaded from the system keybag and unwrapped. It is a binary plist stored in the No Protection class, but whose contents are encrypted with a key held in Effaceable Storage. In order to give forward security to keybags, this key is wiped and regenerated each time a user changes their passcode.

The AppleKeyStore kernel extension manages the system keybag, and can be queried regarding a device's lock state. It reports that the device is unlocked only if all the class keys in the system keybag are accessible, and have been unwrapped successfully.

The following table summarizes the storage for keys in persistent storage.

Key / Persistent Secret	Purpose	Storage (for all devices)
UID	REK for device Key entanglement	Secure enclave
Salt (128 bit)	Additional input to one way functions	AES encrypted in the system keybag
key 0x89B	Wrapping of EMF key	Block 0 of the flash memory. (Effaceable storage.) Secure enclave
key 0x835	Wrapping of DKey	Block 0 of the flash memory. (Effaceable storage.)

		Secure enclave
EMF key	A master key used for the encryption of file system metadata	Stored in wrapped form in persistent storage
NSFileProtectionCompleteUnlessOpen device-wide asymmetric key pair	Writing files while the device is locked	Stored in wrapped form in Persistent storage
CompleteUntilFirstUserAuthentication		Stored in wrapped form in persistent storage
NSFileProtectionCompleteUnlessOpen	Writing files while the device is locked : KDF static public keys	Stored in wrapped form in persistent storage
AfterFirstUnlock		Stored in wrapped form in persistent storage
AfterFirstUnlockThisDeviceOnly		Stored in wrapped form in persistent storage
WhenUnlocked		Stored in wrapped form in persistent storage
WhenUnlockedThisDeviceOnly		Stored in wrapped form in persistent storage
Dkey		Stored in wrapped form in persistent storage
NSFileProtectionNone		Stored in wrapped form in persistent storage
NSFileProtectionComplete class key	User device lock	Stored in wrapped form in persistent storage
Individual keys for files and keychains		Stored in wrapped form in persistent storage

Table 5: Summary of keys and persistent secrets in iOS 10.2

8.2.2 Storage of Persistent Secrets and Private Keys by the Agent

The MD (Mobile Device) Agent calls the Apple iOS API on the device in order to store keys and persistent secrets in the keychain; which are therefore stored in wrapped form in persistent storage, as described above.

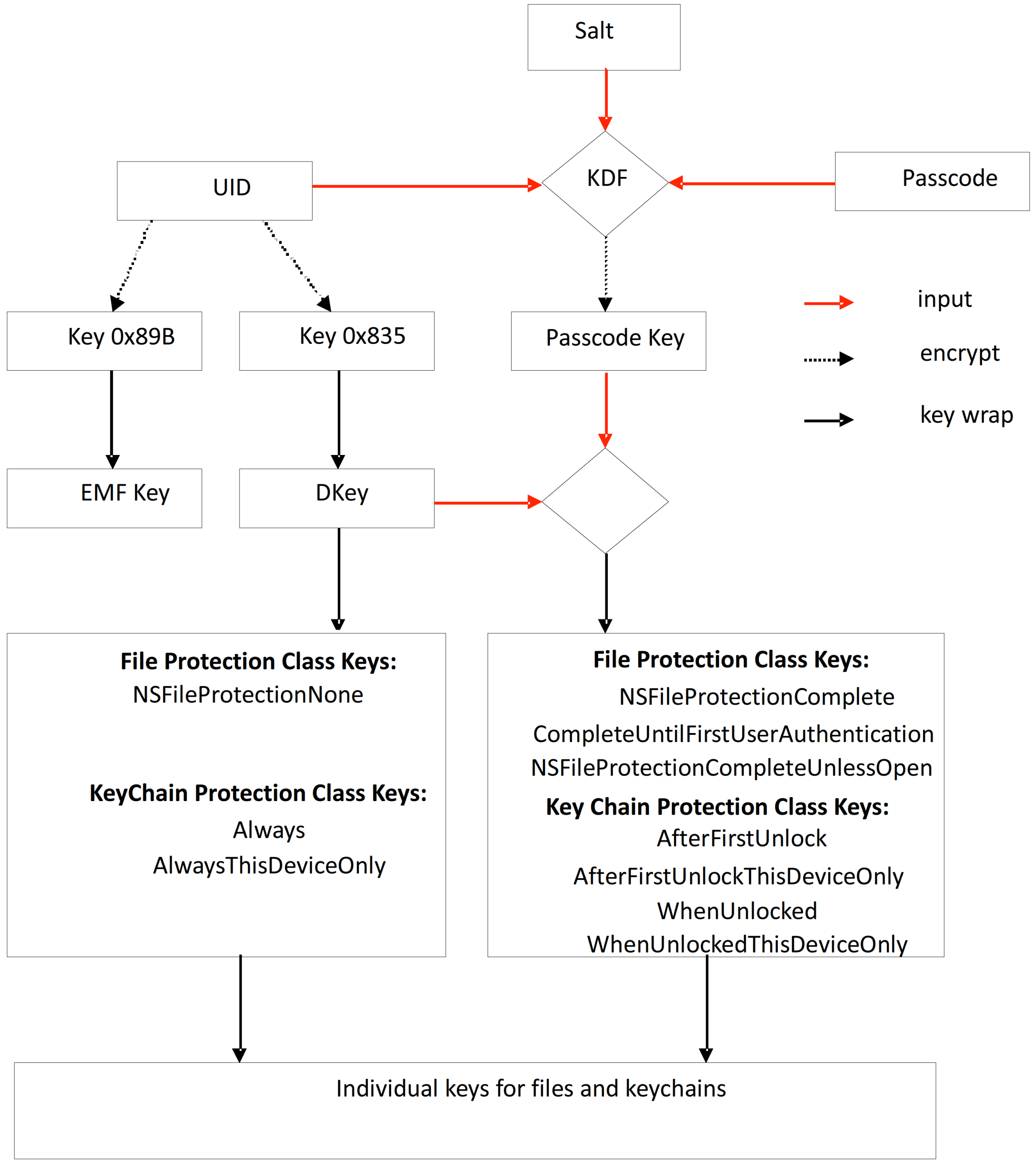
The following table summarizes the keys and persistent secrets stored for the Agent. They are used on all devices listed in this [ST].

Key / Persistent Secret	**Purpose**	**Storage** **(for all devices)**
TLS keys	Protecting MDM Protocol communications with the MDM Server	Stored on the device in wrapped form in persistent storage
Device Push Token	The device push token is received when registering with the Apple Push Notification Service (APNS) in order to have an unambiguous	The token is not stored on the device, but sent to the MDM server. The MDM server stores it to be able to contact the device.

Key / Persistent Secret	Purpose	Storage (for all devices)
	identifier in APNS.	
UDID	Unique Device ID	Stored in wrapped form in persistent storage
PushMagic	The magic string that must be included in the push notification message. This value is generated by the device.	Stored in wrapped form in persistent storage
Device identity certificate	The device presents its identity certificate for authentication when it connects to the check-in server.	Stored in wrapped form in persistent storage
Certificate Payload	https://developer.apple.com/library/ios/featuredarticles/iPhoneConfigurationProfileRef/Introduction/Introduction.html#//apple_ref/doc/uid/TP40010206-CH1-SW248	Stored in wrapped form in persistent storage
Profile encryption key	A profile can be encrypted so that it can only be decrypted using a private key previously installed on a device.	Stored in wrapped form in persistent storage
Guid	Volume Purchase Program (VPP) Account Protection A random UUID should be standard 8-4-4-4-12 formatted UUID string and must be unique for each installation of your product	Stored in wrapped form in persistent storage

Table 6: Summary of keys and persistent secrets used by the Agent

The following figure provides an overview on the key management hierarchy implemented in iOS.



Salt
UID
KDF
Passcode
Key 0x89B
Key 0x835
Passcode Key
input
encrypt
key wrap
EMF Key
DKey
File Protection Class Keys:
NSFileProtectionNone
KeyChain Protection Class Keys:
Always
AlwaysThisDeviceOnly
File Protection Class Keys:
NSFileProtectionComplete
CompleteUntilFirstUserAuthentication
NSFileProtectionCompleteUnlessOpen
Key Chain Protection Class Keys:
AfterFirstUnlock
AfterFirstUnlockThisDeviceOnly
WhenUnlocked
WhenUnlockedThisDeviceOnly
Individual keys for files and keychains

Figure 4: Key Hierarchy in iOS

The Data Protection API can be used by applications to define the class a new file belongs to by using the NSFileProtectionKey attribute and setting its value to one of the classes described above. When the device is locked, a new file can only be created in the classes NSFileProtectionNone and NSFileProtectionCompleteUnlessOpen.

Note that the UID is not accessible by any software and that the two keys 'Key 0x89B' and 'Key 0x835' are both derived by encrypting defined values (identical for all devices) with the UID. Both are stored in the Secure Enclave. All other keys shown in the figure are stored in wrapped form in persistent storage and unwrapped when needed.

To summarize:

- All keys are managed and maintained in the Secure Enclave Processor, a dedicated execution environment with its own operating system that is completely separated from iOS. Both can interact with each other using a mailbox system detailed in section 8.1.1.
- All file system items and all key chain items are stored in encrypted form only.
- File system metadata is encrypted using the EMF key.
- Files and key chain items are encrypted with individual keys. Those keys are wrapped with the class key of the class, the file, or the keychain to which the item belongs.
- Files and key chain items belonging to the classes 'NSFileProtectionNone' (files) and 'Always' or 'AlwaysThisDeviceOnly' are encrypted with keys that are wrapped with the Dkey only. Those items can be accessed (decrypted) before the user is authenticated. For all other classes the passcode key (which is derived from the user's passcode) is used in the generation of the wrapping key used for those classes and therefore decrypting those items is only possible when the user has correctly entered his passphrase.
- All decryption errors are handled in compliance with NIST Special Publication 800-56B.
- When a wipe command is issued, protected data is wiped by erasing the top level KEKs. Since all data-at-rest is encrypted with one of those keys, the device is wiped.

iOS performs the following activities to protect the keys used for file encryption.

Every time the TOE is booted, the TOE does the following.

- An ephemeral AES key (256 bit) is created in the secure enclave using the random number generator of the secure enclave.
- The (wrapped) Dkey and (wrapped) EMF key (both 256 bit keys) are loaded by the iOS kernel from the effaceable storage and sent to the secure enclave.
- The secure enclave unwraps the Dkey and the EMF key.
- The secure enclave wraps the Dkey with the newly generated ephemeral key.
- The secure enclave stores the ephemeral key in the storage controller. This area is not accessible by the iOS kernel.

When iOS accesses a file the following operations are performed.

- The iOS kernel first extracts the file metadata (which are encrypted with the EMF key) and sends them to the secure enclave.
- The secure enclave decrypts the file metadata and sends it back to the iOS kernel.
- The iOS kernel determines which class key to use and sends the class key (which is wrapped with the Dkey, or with the XOR of the Dkey and the Passcode Key) and the file key (which is wrapped with the class key) to the secure enclave.
- The secure enclave unwraps the file key and re-wraps it with the ephemeral key and sends this wrapped key back to the iOS kernel.
- The iOS kernel sends the file access request (read or write) together with the wrapped file key to the storage controller.
- The storage controller uses its internal implementation of AES, decrypts the file key and then decrypts (when the operation is read) or encrypts (when the operation is write) the data during its transfer from/to the flash memory.

The following summarize the storage location for key material.

- The UID is stored in the firmware of the secure enclave in a section not accessible by any program in the secure enclave or the main processor. The processor in the secure enclave can only be used to encrypt and decrypt data (using AES256) using the UID as a key.
- Keys 0x89B and 0x835 are stored in the secure enclave.

- The EMF key, DKey and the class keys are stored in the effaceable area, all in wrapped form only. As explained, they are never available in clear in the main processor system.
- File keys and keychain item keys are stored in non-volatile memory, but in wrapped form only. As explained they are never available in the clear in the main processor system.
- The system and the applications can store private keys in keychain items. They are protected by the encryption of the keychain item.
- Symmetric keys used for TLS, HTTPS, or Wi-Fi sessions are held in RAM only. They are generated and managed using one of the two libraries, Apple iOS CoreCrypto Kernel Module v7.0 and Apple iOS CoreCrypto Module v7.0, or by the AES implementation within the Wi-Fi chip. The functions of those libraries, such as memset(0), also perform the clearing of those keys after use.

8.2.3 CAVS Certificates

The following Cryptographic Algorithm Validation System (CAVS) certificates apply to iOS 10.2. The User and Kernel Space tables have been reproduced from [FIPS_CORECRYPTO_IOS_US_SECPOL_3.2].

User Space

Cryptographic Function	Algorithm	Modes/Options	Algorithm Certificate Number
Random Number Generation; Symmetric Key Generation	[SP 800-90] DRBG	CTR_DRBG (AES Optimized Assembler Implementation) Key Size: 128 bit key size	1264, 1265, 1266, 1267, 1268, 1269, 1270, 1271, 1272, 1273, 1274, 1275
Symmetric Encryption and Decryption	[FIPS 197] AES SP 800-38 A SP 800-38 D SP 800-38 E SP 800-38 F	Optimized Assembler Implementation: Key Sizes: 128/256 bits Modes: CBC, CCM, CTR, GCM, KW	4156, 4157, 4158, 4159, 4160, 4161, 4162, 4163, 4164, 4166, 4169, 4170
Digital Signature and Asymmetric Key Generation	[FIPS186-4] RSA PKCS #1.5	Generic Software Implementation: KEY(gen)(2048/3072) SigGenPKCS1.5 (2048/3072) SigVerPKCS1.5 (1024/2048/3072)	2264, 2265, 2266, 2267, 2268, 2269, 2270, 2271, 2272, 2273, 2274, 2299
	[FIPS 186-4] ECDSA ANSI X9.62	Generic Software Implementation: PKG: curves P-256, P-384 PKV: curves P-256, P-384 SIG(gen): curves P-256, P-384 using (SHA-224, SHA-256, SHA384, SHA512) SIG(ver): curves P-256, P-384 using (SHA-1, SHA-224, SHA-256, SHA384, SHA512)	957, 958, 959, 960, 961, 962, 963, 964, 965, 966, 967, 997

Cryptographic Function	Algorithm	Modes/Options	Algorithm Certificate Number
Message Digest	[FIPS 180-4] SHS	Optimized Assembler Implementation using VNG: SHA-1, SHA-2 (256, 384, 512)	3431, 3432, 3433, 3434, 3435, 3436, 3437, 3438, 3439, 3440, 3441, 3442
Keyed Hash	[FIPS 198] HMAC	Optimized Assembler Implementation using VNG: SHA-1, SHA-2 (256, 384, 512) Key Size: at least 112 bits	2733, 2734, 2735, 2736, 2737, 2738, 2739, 2740, 2741, 2742, 2743, 2744
KAS	[SP800-56A] EC Diffie-Hellman Implementation follows SP800-56A for primitive only	Generic Software Implementation: 6.2.2.2 One-Pass Diffie-Hellman, C(1e, 1s, ECC CDH) Curves:P-256, P384	CVL: 959, 960, 961, 962, 963, 964, 965, 966, 967, 968, 969, 1010
Key Derivation	[SP 800-132] PBKDF	Password Based Key Derivation using HMAC with SHA-1 or SHA-2	Vendor Affirmed
RSA Key Wrapping	SP800-56B	KTS-OAEP	Vendor Affirmed

Table 7: User Space CAVS Certificates

Kernel Space

Cryptographic Function	Algorithm	Modes/Options	Algorithm Certificate Number
Symmetric Encryption and Decryption	[FIPS 197] AES SP800-38 A SP800-38 E SP800-38 F	Optimized Assembler Implementation: Key sizes: 128/256 bits Modes: CBC, KW	4255, 4256, 4257, 4258, 4259, 4260
Message Digest	[FIPS 180-4] SHS	Optimized Assembler Implementation using VNG: SHA-1, SHA-2 (256, 384, 512)	3557, 3558, 3559, 3560, 3561, 3562

Cryptographic Function	Algorithm	Modes/Options	Algorithm Certificate Number
Keyed Hash	[FIPS 198] HMAC	Optimized Assembler Implementation: KS<BS, KS=BS, KS>BS SHA-1, SHA-2 (256, 384, 512)	2854, 2855, 2856, 2857, 2858, 2859

Table 8: Kernel Space CAVS Certificates

Wi-Fi:

iOS Device Name	*Broadcom Wi-Fi Chip Core*	*AES CAVS Certificate Number*	*CCM*	*ECB*	*CBC*	*CTR*
iPhone 6	4345	4035	4035	4035	4035	4035
iPhone 6 Plus	4345	4035	4035	4035	4035	4035
iPhone 6S	4350	4035	4035	4035	4035	4035
iPhone 6S Plus	4350	4035	4035	4035	4035	4035
iPhone SE	43452	4035	4035	4035	4035	4035
iPhone 7	4355	3678	3678	3678	3678	3678
iPhone 7 Plus	4355	3678	3678	3678	3678	3678
iPad mini 4	4350	4035	4035	4035	4035	4035
iPad Air 2	4350	4035	4035	4035	4035	4035
iPad Pro 9.7"	4355	3678	3678	3678	3678	3678
iPad Pro 12.9"	4355	3678	3678	3678	3678	3678
iPad	4355	3678	3678	3678	3678	3678

Table 9: Wi-Fi CAVS Certificates

Hardware DRBG:

Cryptographic Function	*Algorithm*	*Modes/Options*	*Algorithm Certificate Number*
DRBG offered by the CPU for the Secure Enclave used by the Apple Key Store.	[SP800-90A]	CTR_DRBG	1479, 1480, 1481, 1482, 1483, 1484
	[SP800-90A]	ECB	4518, 4519, 4520, 4521, 4522, 4523

Table 10: Hardware DRBG CAVS Certificates

8.3 User Data Protection (FDP)

8.3.1 Protection of Files

When a new file is created on an iOS device, it’s assigned a class by the app that creates it. Each class uses different policies to determine when the data is accessible. As described above each class has a dedicated class key which is stored in wrapped form. Note that for the classes other than 'No Protection' to work the user must have an active passcode lock set for the device.

The basic classes and policies are described below.

Complete Protection (referred to as "class A" in some documents)

Files in this class can only be accessed when the device is unlocked.

Protected Unless Open (referred to as "class B" in some documents)

This class is for files that may need to be written while the device is locked.

Protected Until First User Authentication (referred to as "class C" in some documents)

This class is for files that are protected until the user has successfully authenticated. Unlike the 'Complete Protection' class, the class key for this class is not wiped when the device is locked, but after a re-boot the user has to authenticate before files in this class can be accessed. So, once the user has authenticated after reboot the key is available until the phone is shutdown or rebooted.

No Protection (referred to as "class D" in some documents)

Files in this class can be always accessed. Still the files themselves are encrypted using a file specific key, but this key can be unwrapped without using the passcode key derived from the user's passcode.

Note: class A, class B and class C keys require that the user has defined a PIN. Unless he has done this only class D keys exist.

All data in files is considered private data, since all files are encrypted. Sensitive data is data protected with a class A or class B key, since this data is not accessible when the device is locked.

8.3.2 Application Access to Files

An iOS app’s interactions with the file system are limited mostly to the directories inside the app’s sandbox. During installation of a new app, the installer creates a number of containers for the app. Each container has a specific role. The bundle container holds the app’s bundle, whereas the data container holds data for both the application and the user. The data container is further divided into a number of directories that the app can use to sort and organize its data. The app may also request access to additional containers—for example, the iCloud container—at runtime.

When an application is removed, all of its files are also removed.

8.3.3 Declaring the Required Device Capabilities of an App

All apps must declare the device-specific capabilities they need to run. The value of the UIRequiredDeviceCapabilities key is either an array or dictionary that contains additional keys identifying features your app requires (or specifically prohibits). If you specify the value of the key using an array, the presence of a key indicates that the feature is required; the absence of a key indicates that the feature is not required and that the app can run without it. If a dictionary is specified instead, each key in the dictionary must have a Boolean value that indicates whether the feature is required or prohibited. A value of true indicates the feature is required and a value of false indicates that the feature must not be present on the device.

8.3.4 App Groups

Apps and extensions owned by a given developer account can share content when configured to be part of an App Group. It is up to the developer to create the appropriate groups on the Apple Developer Portal and include the desired set of apps and extensions. Once configured to be part of an App Group, apps have access to the following:

- A shared on-disk container for storage, which will stay on the device as long as at least one app from the group is installed
- Shared preferences
- Shared keychain items

The Apple Developer Portal guarantees that App Group IDs are unique across the app ecosystem.

8.3.5 Restricting App Access to System Services

The TOE allows a user to restrict the system services an app can access. The services that can be restricted on a per-app basis are:

- Location Services
- Contacts
- Calendar
- Reminders
- Photos
- Microphone
- Camera

8.3.6 Keychain Data Protection

Many apps need to handle passwords and other short but sensitive bits of data, such as keys and login tokens. The iOS keychain provides a secure way to store these items.

The keychain is implemented as a SQLite database stored on the file system. There is only one database; the securityd daemon determines which keychain items each process or app can access. Keychain access APIs result in calls to the daemon, which queries the app's "keychain-access-groups" and the "application-identifier" entitlement. Rather than limiting access to a single process, access groups allow keychain items to be shared between apps.

Keychain items can only be shared between apps from the same developer. This is managed by requiring third-party apps to use access groups with a prefix allocated to them through the iOS Developer Program, or in iOS 10.2, via application groups. The prefix requirement and application group uniqueness are enforced through code signing, Provisioning Profiles, and the iOS Developer Program. iOS provides a user interface (UI) in the Settings dialog that allows importing of keys for use for Apple-provided applications such as Safari or VPN.

Keychain data is protected using a class structure similar to the one used in file Data Protection. These classes have behaviors equivalent to file Data Protection classes, but use distinct keys and are part of APIs that are named differently.

The following table shows the keychain classes and their equivalent file system classes.

Keychain data protection class	***File data protection class***
When unlocked	NSFileProtectionComplete
While locked	NSFileProtectionCompleteUnlessOpen
After first unlock	NSFileProtectionCompleteUntilFirstUserAuthentication
Always	NSFileProtectionNone

Table 11: Keychain to File-system Mapping

In addition, there are the Keychain data protection classes with the additional "ThisDeviceOnly" added to their class name. Keychain items in those classes cannot be moved to a different device using backup and restore; key chain items in those classes are bound to the device.

Among the data stored in keychain items are digital certificates used for setting up VPN connections and certificates and private keys installed by the Configuration Profile.

Keychains can use access control lists (ACLs) to set policies for accessibility and authentication requirements. Items can establish conditions that require user presence by specifying that they can't be accessed unless authenticated entering the device's passcode. ACLs are evaluated inside the Secure Enclave and are released to the kernel only if their specified constraints are met.

8.4 Identification and Authentication (FIA)

The user of the device is authenticated using a passcode. Except for making emergency calls users need to authenticate using a password. All passcode entries are obscured by "*" symbols for each character after one second as the user input occurs. The following passcode policies can be defined for managed devices:

- The minimum length of the passcode
- The minimum number of special characters a valid passcode must contain
- The maximum number of consecutive failed attempts to enter the passcode (which can be value between 2 and 10, the default is 10)
- The number of minutes for which the device can be idle before it gets locked by the system
- The maximum number of days a passcode can remain unchanged
- The size of the passcode history (the maximum value is 50)

Those parameters for the passcode policy can be defined in the Passcode Policy Payload section of a configuration profile defined by a system administrator for a managed device. For details see section 8.5 below. The number of failed authentication attempts is maintained in a system file which will persist in the event of graceful or ungraceful loss of power to the TOE. The counter maintaining the number of failed consecutive logon attempts is increased by one immediately once the TOE has identified that the password is incorrect. The increment of the counter is completed before the UI informs the user about the failed logon attempt.

The time between consecutive authentication attempts is at least the time it takes the PBKDF2 function to execute. This is calibrated to be at least 80 milliseconds. In addition, the TOE enforces a 5 second delay between repeated failed authentication attempts. When a user exceeds the number of consecutive failed login attempts, the user's partition is erased (by erasing the encryption key). The OS partition is mounted READONLY upon boot and is never modified during the use of the TOE except during a software update or restore. Note that entering the same incorrect password multiple times consecutively causes the failed login counter to increment only once for those multiple attempts even though these are all failed login attempts.

Additionally, authentication credentials are not stored on the TOE in any location. Successful authentication attempts are achieved exclusively by a successful key derivation that decrypts the keybag in the Secure Enclave Processor with the respective class keys.

8.4.1 Certificates

There are a number of certificates used by the TOE. First there is the Apple certificate used to verify the integrity and authenticity of software updates. This certificate is installed in ROM during manufacturing.

Other certificates used for setting up trusted channels or decrypt/verify protected e-mail can be imported by a user (if allowed by the policy) or installed using configuration profiles.

Certificates can be installed for:

- Active Directory
- Email (Secure/Multipurpose Internet Mail Extensions (S/MIME))
- Host pairing
- Single-Sign On

- IPSec
- TLS
- EAP-TLS, other supported EAP protocols

Note that only IPSec, TLS and EAP-TLS are addressed by [PP_MD_V3.0]. Certificates have a certificate type that defines their respective application area. This ensures that only certificates defined for a specific application area are used. In addition, the database containing trust anchors for all certificates is protected via integrity check and write protection. The certificate types supported by the TOE are:

- AppleX509Basic
- AppleSSL
- AppleSMIME
- AppleEAP
- AppleIPsec
- AppleCodeSigning
- AppleIDValidation
- AppleTimeStamping

External entities can be authenticated using a digital certificate. Out of the box, iOS includes a number of preinstalled root certificates.

Code signing certificates need to be assigned by Apple and can be imported into a device. The issue of such a certificate can be by app developers or by enterprises that want to deploy apps from their MDM to managed devices. All apps must have a valid signature that can be verified by a code signing certificate before they are installed on a device.

iOS can update certificates wirelessly, if any of the preinstalled root certificates become compromised. To disable this, there is a restriction that prevents over-the-air certificate updates.

The list of supported certificate and identity formats are:

- X.509 certificates with RSA keys
- File extensions cer, .crt, .der, .p12, and .pfx

To use a root certificate that isn't preinstalled, such as a self-signed root certificate created by the organization managing the TOE, they can be distributed using one of the methods listed below:

- When reviewed and accepted by the user
- Using the configuration profile
- Downloaded from a web site

When attempting to establish a connection using a remote certificate, the certificate is first checked to ensure it is valid. Certificates are validated against the common name (CN) portion of the distinguished name (DN). If the CN does not match the corresponding domain name system (DNS) or IP Address of the server being accessed, validation and subsequently the connection will fail. If the certificate is valid, the attempt to establish the connection continues. If the certificate is invalid, the next step is up to the application. The application should provide an indication to the user that the certificate is invalid and options to accept or reject.

TLS is implemented as a stack that can be utilized by third-party applications. The API informs the calling application that the certificate is not valid. For example, Safari will display a message to the user that the remote certificate validation failed and allow the user to examine the certificate with the option to allow the connection or not.

iOS can be configured to disable the user option to accept invalid TLS certificates using the "Allow user to accept untrusted TLS certificates" setting. This is described in the iOS Security Guide [iOS_SEC].

8.4.2 MDM Server Reference ID

The initial MDM Payload contains a mandatory ServerURL string. The URL that the device contacts to retrieve device management instructions must begin with the https:// URL scheme, and may contain a port number (for example ":1234").

The MDM check-in protocol is used during initialization to validate a device's eligibility for MDM enrollment and to inform the MD server that a device's Push Token has been updated. If a check-in server URL is provided in the MDM payload, the check-in protocol is used to communicate with that check-in server. If no check-in server URL is provided, the main MDM Server URL is used instead.

A managed Mobile Device uses an identity to authenticate itself to the MDM Server over TLS (Secure Sockets Layer (SSL)). This identity can be included in the profile as a Certificate payload, or can be generated by enrolling the device with Simple Certificate Enrollment Protocol (SCEP)[1]. Each MDM Server must be registered with Apple at the MDM Server Device Enrollment Program (DEP) management portal. The DEP provides details about the server entity to identify it uniquely throughout the organization deploying the MDM Server. Each server can be identified by either its system-generated UUID or by a user-provided name assigned by one of the organization's users. Both the UUID and server name must be unique within the organization. Registered MDM servers can include third party servers. iOS devices automatically connect to the MDM Server during setup if the device is enrolled into the DEP and is assigned to an MDM Server. During the device enrollment, the MDM enrollment service returns a JavaScript Object Notation (JSON) dictionary with the keys to mobile devices shown in the following table.

Key	Value
server_name	An identifiable name for the MDM Server
server_uuid	A system-generated server identifier
admin_id	Apple ID of the person who generated the current tokens that are in use
facilitator_id	Legacy equivalent to the admin_id key. **This key is deprecated and may not be returned in future responses.**
org_name	The organization name
org_email	The organization email address
org_phone	The organization phone
org_address	The organization address

Table 12: MDM Server Reference Identifiers

8.5 Specification of Management Functions (FMT)

In FMT_SMF_EXT.3.1 no "any assigned functions" are selected and so these are not documented in this [ST] as supported by the platforms.

Since all the Mobile Devices specified in this [ST] use iOS, there are no differences between supported management functions and policies between the different mobile devices. The supported management functions for iOS are described in [iOS_CFG]

Table 4: Management Functions, describes all management functions of the devices as well as the MDM Agent that are available when the device is enrolled in MDM.

8.5.1 Enrollment

The methods by which an MDM Agent can be enrolled are as follows.

[1] More information about SCEP can be found at https://datatracker.ietf.org/doc/draft-nourse-scep/

- Manually, using Apple's Profile Manager
- Manually, using Apple Configurator 2
- Distributing an enrollment profile via email, or a web site
- Device Enrollment Program (This is an automated and enforced method of automatically enrolling new devices.)

A more detailed description is found in section 8.4.2, above. In addition, the enrollment process is discussed in the MDM Protocol reference [iOS_MDM].

8.5.2 Configuration Profiles

The TOE can be configured using "Configuration Profiles" that are installed on the TOE. Configuration Profiles are XML files that may contain settings for a number of configurable parameters. For details on the different payloads and keys that can be defined see the document "Configuration Profile Key Reference" ([iOS_CFG]) that can be downloaded from the Apple web site.

Configuration profiles can be deployed in one of five different ways:

- using the Apple Configurator tool,
- via an email message,
- via a web page,
- using over-the-air configuration as described in [iOS OTAConfig], and
- using over-the-air configuration via a MDM Server.

To preserve the integrity, authenticity and confidentiality of Configuration Profiles they can be required to be digitally signed and encrypted.

Managed items relevant for this [ST] that have to be configured using Configuration Profiles are as follows.

- The password policy—the administrator can define this using the Passcode Policy Payload and
    - define the minimum password length,
    - define requirements for the password complexity,
    - define the maximum password lifetime,
    - define the maximum time of inactivity after which the device is locked automatically, and
    - define the maximum number of consecutive authentication failures after which the device is wiped.
- The VPN policy as follows:
    - if VPN is always on,
    - define the authentication method (Shared Secret or Certificate), and
    - specification of certificates or shared keys.
- The Wi-Fi policy as follows:
    - the EAP types allowed,
    - the service set identifiers (SSIDs) allowed to connect to,
    - the encryption type(s) allowed,
    - enabling/disabling Wi-Fi hotspot functionality.
- General restrictions as follows:
    - allowing or disallowing specific services (e. g. remote backup) or using devices like the camera,
    - allowing or disallowing notifications when locked, and
    - allowing or disallowing a prompt when an untrusted certificate is presented in a TLS/HTTPS connection.

Note that the microphone cannot be disabled in general but a user can restrict access to the microphone on a per-app basis.

Other functions that can be enabled/disabled by an administrator are:

- the installation of applications by a user,
- the possibility to perform backups to iCloud,
- the ability to submit diagnostics automatically,
- the ability to use the fingerprint device (Touch ID) for user authentication[2],
- the ability to see notifications on the lock screen,
- the ability to take screen shots,
- the ability to accept untrusted TLS certificates, and
- the ability to perform unencrypted backups (via iTunes).

Further restrictions can be enforced for enrolled devices. Those include:

- the ability to modify the account,
- the ability to modify the cellular data usage,
- the ability to pair with a host other than the supervision host,
- the ability for the user to install configuration profiles or certificates interactively, and
- the ability for the user to use 'Enable Restrictions' interface.

A user can access management functions available to him via the menus of the graphical user interface. The functions he can perform are described in [iPhone_UG] and [iPad_UG].

Configuration profiles can also be deployed such that users are unable to override or remove restrictions set in place by Administrators or MDM Administrators. Depending on the behavior defined in the configuration profile, users will be unable to access, perform, or relax management functions defined in Table 4: Management Functions. In the most restrictive mode, users will not be able to access the options to alter the above functionality at all. In less restrictive modes, the user is only able to select more secure options.

8.5.3 Unenrollment

The Configuration Profile Key Reference, [iOS_CFG], describes unenrollment options through specifying the key "PayloadRemovalDisallowed". This is an optional key. If present and set to true the user cannot delete the profile unless the profile has a removal password and the user provides it.

It is up to the MDM Server to ensure that this key is set appropriately.

In supervised mode the MDM payload can be locked to the device.

In addition, [iOS_MDM] describes the additional ability to restrict the installation and removal of configuration profiles from other sources. This is achieved using the AccessRights key which has a value of a logical OR of the following bits.

Required

- 1—Allow inspection of installed configuration profiles
- 2—Allow installation and removal of configuration profiles
- 4—Allow device lock and passcode removal
- 8—Allow device erase
- 16—Allow query of Device Information (device capacity, serial number)
- 32—Allow query of Network Information (phone/SIM numbers, MAC addresses)
- 64—Allow inspection of installed provisioning profiles
- 128—Allow installation and removal of provisioning profiles
- 256—Allow inspection of installed applications
- 512—Allow restriction-related queries
- 1024—Allow security-related queries
- 2048—Allow manipulation of settings
- 4096—Allow app management

[2] The use of the fingerprint device for biometric authentication is not considered in the evaluated configuration.

Note that the AccessRights key may not be zero. If bit 2 is specified, then bit 1 must also be specified. If bit 128 is specified, then bit 64 must also be specified.

8.6 Protection of the TSF (FPT)

8.6.1 Secure Boot

Each step of the startup process contains components that are cryptographically signed by Apple to ensure integrity and that proceed only after verifying the chain of trust. This includes the bootloaders, kernel, kernel extensions, and baseband firmware.

When an iOS device is turned on, its application processor immediately executes code from read-only memory known as the Boot ROM. This immutable code, known as the hardware root of trust, is laid down during chip fabrication, and is implicitly trusted. The Boot ROM code contains the Apple Root CA public key, which is used to verify that the Low-Level Bootloader (LLB) is signed by Apple before allowing it to load. This is the first step in the chain of trust where each step ensures that the next is signed by Apple. When the LLB finishes its tasks, it verifies and runs the next-stage bootloader, iBoot, which in turn verifies and runs the iOS kernel.

This secure boot chain helps ensure that the lowest levels of software are not tampered with and allows iOS to run only on validated Apple devices.

For devices with cellular access, the baseband subsystem also utilizes its own similar process of secure booting with signed software and keys verified by the baseband processor.

For devices with an A7 or later A-series processor, the Secure Enclave coprocessor also utilizes a secure boot process that ensures its separate software is verified and signed by Apple.

If one step of this boot process is unable to load or verify the next process, startup is stopped and the device displays the “Connect to iTunes” screen. This is called recovery mode. If the Boot ROM is not able to load or verify LLB, it enters DFU (Device Firmware Upgrade) mode. In both cases, the device must be connected to iTunes via USB and restored to factory default settings.

8.6.2 Joint Test Action Group (JTAG) Disablement

Typical JTAG ports such as Data In, Data Out, Clock, etc are not present for physical access on iOS devices.

8.6.3 Secure Software Update

Software updates to the TOE are released regularly to address emerging security concerns and also provide new features; these updates are provided for all supported devices simultaneously. Users receive iOS update notifications on the device and through iTunes, and updates are delivered wirelessly, encouraging rapid adoption of the latest security fixes.

The startup process described above helps ensure that only Apple-signed code can be installed on a device. To prevent devices from being downgraded to older versions that lack the latest security updates, iOS uses a process called System Software Authorization. If downgrades were possible, an attacker who gains possession of a device could install an older version of iOS and exploit a vulnerability that’s been fixed in the newer version.

On a device with an A7 or later A-series processor, the Secure Enclave coprocessor also utilizes System Software Authorization to ensure the integrity of its software and prevent downgrade installations.

iOS software updates can be installed using iTunes or over-the-air (OTA) on the device via HTTPS trusted channel. With iTunes, a full copy of iOS is downloaded and installed. OTA software updates download only the components required to complete an update, improving network efficiency, rather than downloading the entire OS. Additionally, software updates can be cached on a local network server running the caching service on OS X Server so that iOS devices do not need to access Apple servers to obtain the necessary update data.

During an iOS upgrade, iTunes (or the device itself, in the case of OTA software updates) connects to the Apple installation authorization server and sends it a list of cryptographic measurements for each part of the installation bundle to be installed (for example, LLB, iBoot, the kernel, and OS image), a random anti-replay value (nonce), and the device's unique ID (ECID).

The authorization server checks the presented list of measurements against versions for which installation is permitted and, if it finds a match, adds the ECID to the measurement and signs the result. The server passes a complete set of signed data to the device as part of the upgrade process. Adding the ECID "personalizes" the authorization for the requesting device. By authorizing and signing only for known measurements, the server ensures that the update takes place exactly as provided by Apple.

The boot-time, chain-of-trust evaluation verifies that the signature comes from Apple and that the measurement of the item loaded from disk, combined with the device's ECID, matches what was covered by the signature.

These steps ensure that the authorization is for a specific device and that an old iOS version from one device can't be copied to another. The nonce prevents an attacker from saving the server's response and using it to tamper with a device or otherwise alter the system software.

Note that this ensures the integrity and authenticity of software updates. A TLS trusted channel is provided for this process.

8.6.4 Security Updates

Apple generally does not disclose, discuss, or confirm security issues until a full investigation is complete and any necessary patches or releases are available. Once an issue has been confirmed and a patch has been made available references containing technical details on the patches / Common Vulnerabilities and Exposures (CVEs), etc are released. Apple also distributes information about security issues in its products through security advisories. Advisories are provided through the security-announce mailing list. Resources include the following.

Contact Apple About Security Issues
https://support.apple.com/en-us/HT201220
Apple Security Updates (Advisories)
https://support.apple.com/en-us/HT201222
Security-Announce Mailing List (receive Apple security advisories through)
https://lists.apple.com/mailman/listinfo/security-announce/

8.6.5 Domain Isolation

All applications are executed in their own domain or 'sandbox' which isolates the application from other applications and the rest of the system. They are also restricted from accessing files stored by other applications or from making changes to the device configuration. Each application has a unique home directory for its files, which is randomly assigned when the application is installed. If a third-party application needs to access information other than its own, it does so only by using services explicitly provided by iOS.

System files and resources are also shielded from the user's apps. The majority of iOS runs as the non-privileged user "mobile," as do all third-party apps. The entire OS partition is mounted as read-only. Unnecessary tools, such as remote login services, aren't included in the system software, and APIs do not allow apps to escalate their own privileges to modify other apps or iOS itself.

Access by third-party apps to user information and features is controlled using declared entitlements. Entitlements are key value pairs that are signed in to an app and allow authentication beyond runtime factors like a UNIX user ID. Since entitlements are digitally signed, they cannot be changed. Entitlements are used extensively by system apps and daemons to perform specific privileged operations that would otherwise require the process to run as root. This greatly reduces the potential for privilege escalation by a compromised system application or daemon.

Address space layout randomization (ASLR) protects against the exploitation of memory corruption bugs. All TSF binaries and libraries use ASLR to ensure that all memory regions are randomized upon launch. Xcode, the iOS development environment, automatically compiles third-party programs with ASLR support turned on. Address space layout randomization is used for every sandbox used to execute applications in. There are 8 bits of randomness taken from the application processor TRNG involved in the randomization.

In addition, The Memory Management Unit (MMU) supports memory address translation using a translation table maintained by the OS kernel. For each page, the MMU maintains flags that allow or deny the read, write or execution of data. Execution in this case allows the CPU to fetch instructions from a given page.

8.6.6 Device Locking

The TOE is locked after a configurable time of user inactivity or upon request of the user. When the device is locked, the class key for the 'Complete Protection' class is wiped 10 seconds after locking, making files in that class inaccessible. This also applies for the class key of the 'Accessible when unlocked' keychain class.

The lock screen of a device can be defined and set for supervised devices by an administrator using Apple Configurator or an MDM.

The TOE can be locked remotely either via the iCloud "Lost Mode" function or by an MDM system if the device is enrolled in management.

8.6.7 Time

The following security functional requirements make use of time:

- ALC_TSU_EXT
- FAU_GEN.1.2
- FIA_UAU.7
- FIA_X509_EXT.1.1
- FIA_X509_EXT.3.1
- FMT_SMF_EXT.1.1 Function 1:
- FMT_SMF_EXT.1.1 Function 2:
- FPT_STM.1.1
- FTA_SSL_EXT.1

When the device starts and the "Set Automatically" setting is set on the device the following services are used to synchronize the real-time clock on the device:

For A7 and later platforms the devices set time by:

1. GPS.
2. If GPS is unavailable, NTP will be used.

If the "Set Automatically" setting is not set on the device the user can manually set the clock using an appropriate reference facility.

When configured and maintained according to the Network, Identity and Time Zone (NITZ), Global Positioning Satellites (GPS), Network Time Protocol (NTP) standards or the cellular carrier time service the time may be considered reliable.

8.6.8 Inventory of TSF Binaries and Libraries

All user space binaries (applications as well as shared libraries) are subject to address space layout randomization. The logic is implemented in the binary loader and agnostic of a particular file or its contents. An inventory of TSF binaries and libraries has been provided to NIAP since the list is considered proprietary.

8.6.9 Self-Tests

Self-tests are performed by the Apple iOS CoreCrypto modules.

8.6.9.1 Apple iOS CoreCrypto Module V7.0

The module performs self-tests to ensure the integrity of the module and the correctness of the cryptographic functionality at start up. In addition, the random bit generator requires continuous verification. The FIPS Self Tests application runs all required module self-tests. This application is invoked by the iOS startup process upon device power on.

The execution of an independent application for invoking the self-tests in the libcorecrypto.dylib makes use of features of the iOS architecture: the module, implemented in libcorecrypto.dylib, is linked by libcommoncrypto.dylib which is linked by libSystem.dylib. The libSystem.dylib is a library that must be loaded into every application for operation. The library is stored in the kernel cache and therefore is not available on the disk as directly visible files. iOS ensures that there is only one physical instance of the library and maps it to all application linking to that library. In this way, the module always stays in memory. Therefore, the self-test during startup time is sufficient as it tests the module instance loaded in memory which is subsequently used by every application on iOS.

All self-tests performed by the module are listed and described in this section.

8.6.9.1.1 Power-Up Tests

The following tests are performed each time the Apple iOS CoreCrypto Module, v7.0 starts and must be completed successfully for the module to operate in the FIPS approved mode. If any of the following tests fail, the device powers itself off. To rerun the self-tests on demand, the user must reboot the device.

8.6.9.1.2 Cryptographic Algorithm Tests

Algorithm	Modes	Test
TripleData Encryption Standard (DES)	CBC	KAT (Known Answer Test) Separate encryption / decryption operations are performed
Generic-software implementation (non-optimized based on LibTomCrypt): AES-128, AES-192, AES-256	CBC, Electronic Codebok (ECB), GCM	KAT Separate encryption / decryption operations are performed
Generic-software implementation (non-optimized based on Gladman): AES-128, AES-192, AES-256	CBC	KAT Separate encryption / decryption operations are performed
Optimized-software implementation: AES-128, AES-192, AES-256	CBC, ECB, GCM	KAT Separate encryption / decryption operations are performed

Algorithm	Modes	Test
Hardware implementation: AES-128, AES-192, AES-256	CBC	KAT Separate encryption / decryption operations are performed
DRBG	N/A	KAT
HMAC-SHA-1, HMAC-SHA-224, HMAC-SHA-256, HMAC-SHA-384, HMAC-SHA-512	N/A	KAT
RSA	SIG(ver), SIG(gen) Encrypt / decrypt	KAT, pair-wise consistency checks Separate encryption /decryption operations are performed
ECDSA	SIG(ver), SIG(gen)	Pair-wise consistency checks

Table 13: Apple iOS CoreCrypto Module V7.0 Cryptographic Algorithm Tests

8.6.9.1.3 Software / Firmware Integrity Tests

A software integrity test is performed on the runtime image of the Apple iOS CoreCrypto Module, v7.0. The CoreCrypto's HMAC-SHA-256 is used as an Approved algorithm for the integrity test. If the test fails, then the device powers itself off.

8.6.9.1.4 Critical Function Tests

No other critical function test is performed on power up.

8.6.9.1.5 Conditional Tests

The following sections describe the conditional tests supported by the Apple iOS CoreCrypto Module, v7.0.

- **Continuous Random Number Generator Test**
  The Apple iOS CoreCrypto Module, v7.0 performs a continuous random number generator test, whenever CTR_DRBG is invoked.
- **Pair-wise Consistency Test**
  The Apple iOS CoreCrypto Module, v7.0 does generate asymmetric keys and performs all required pair-wise consistency tests, the encryption/decryption as well as signature verification tests, with the newly generated key pairs.
- **SP 800-90A Assurance Tests**
  The Apple iOS CoreCrypto Module, v7.0 performs a subset of the assurance tests as specified in section 11 of SP 800-90A, in particular it complies with the mandatory documentation requirements and performs know-answer tests and prediction resistance.
- **Critical Function Test**
  No other critical function test is performed conditionally.

8.6.9.2 Apple iOS CoreCrypto Kernel Module V7.0

The module performs self-tests to ensure the integrity of the module and the correctness of the cryptographic functionality at start up. In addition, the DRBG requires continuous verification. The FIPS Self Tests functionality runs all required module self-tests. This functionality is invoked by the iOS Kernel startup process upon device initialization. If the self-tests succeed, the Apple iOS CoreCrypto Kernel Module, v7.0 instance is maintained in the memory of the iOS Kernel on the device and made available to each calling kernel service without reloading. All self-tests performed by the module are listed and described in this section.

8.6.9.2.1 Power-Up Tests

The following tests are performed each time the Apple iOS CoreCrypto Kernel Module, v7.0 starts and must be completed successfully for the module to operate in the FIPS Approved Mode. If any of the following tests fails the device shuts down automatically. To run the self-tests on demand, the user may reboot the device.

8.6.9.2.2 Cryptographic Algorithm Tests

Algorithm	Modes	Test
Triple-DES	CBC	KAT (Known Answer Test) Separate encryption / decryption operations are performed
AES implementations selected by the module for the corresponding environment AES-128, AES-192, AES-256	ECB, CBC	KAT Separate encryption / decryption operations are performed
DRBG	N/A	KAT
SHA implementations selected by the module for the corresponding environment SHA-1, SHA-224, SHA-256, SHA-384, SHA-512	N/A	KAT
HMAC-SHA-1, HMAC-SHA-224, HMAC-SHA-256, HMAC-SHA-384, HMAC-SHA-512	N/A	KAT
ECDSA	SIG(ver), SIG(gen)	pair-wise consistency test
RSA	SIG(ver)	KAT

Table 14:Apple iOS CoreCrypto Kernel Module V7.0 Cryptographic Algorithm Tests

8.6.9.2.3 Software / Firmware Integrity Tests

A software integrity test is performed on the runtime image of the Apple iOS CoreCrypto Kernel Module, v7.0. The CoreCrypto's HMAC-SHA256 is used as an Approved algorithm for the integrity test. If the test fails, then the device powers itself off.

8.6.9.2.4 Critical Function Tests

No other critical function test is performed on power up.

8.6.9.3 Conditional Tests

The following sections describe the conditional tests supported by the Apple iOS CoreCrypto Kernel Module, v7.0.

- **Continuous Random Number Generator Test**
  The Apple iOS CoreCrypto Kernel Module, v7.0 performs a continuous random

number generator test, whenever CTR_DRBG is invoked. In addition, the seed source implemented in the operating system kernel also performs a continuous self-test.

- **Pair-wise Consistency Test**
  The Apple iOS CoreCrypto Kernel Module, v7.0 generates asymmetric ECDSA key pairs and performs all required pair-wise consistency tests (signature generation and verification) with the newly generated key pairs.
- **9SP800-90A Assurance Tests**
  The Apple iOS CoreCrypto Kernel Module, v7.0 performs a subset of the assurance tests as specified in section 11 of SP800-90A, in particular it complies with the mandatory documentation requirements and performs know-answer tests and prediction resistance.
- **Critical Function Test**
  No other critical function test is performed conditionally.

8.7 TOE Access (FTA)

8.7.1 Session Locking

iOS devices can be configured to transit to a locked state after a configurable time interval of inactivity. This time can be defined by an administrator using a Configuration Profile. Displaying notifications when in the locked state can be prohibited via the allowLockScreenNotificationsView key in the Restrictions Payload of a Configuration Profile.

8.7.2 Restricting Access to Wireless Networks

Users and administrators can restrict the wireless networks an iOS device connects to. Using the configuration profile administrators can define the wireless networks the device is allowed to connect to and the EAP types allowed for authentication. This also includes the following attributes:

- Specification of the CA(s) from which the TSF will accept WLAN authentication server certificates(s)
- Security type
- Suthentication protocol
- Client credentials to be used for authentication

For the list of radios supported by each device see Table 1: Devices Covered by the Evaluation and Table 2: Cellular Protocols Supported. The standards listed there define the frequency ranges.

8.7.3 Lock Screen Banner Display

An advisory warning message regarding unauthorized use of the TOE can be defined using an image that is presented during the lock screen. Configuration for this is described in FMT_SMF_EXT.1.1 Function 36.

8.8 Trusted Path/Channels (FTP)

The TOE provides secured (encrypted and mutually authenticated) communication channels between itself and other trusted IT products through the use of 802.11-2012, 802.1X, and EAP-TLS, TLS and IPsec.

IPsec supports authentication using shared keys or certificate based authentication.

8.8.1 EAP-TLS and TLS

The TOE supports EAP-TLS using TLS version 1.0 and supports the following ciphersuites:

1. TLS_RSA_WITH_AES_128_CBC_SHAas defined in RFC 5246
2. TLS_RSA_WITH_AES_256_CBC_SHA as defined in RFC 5246

EAP-TLS can be configured as one of the EAP types accepted using the AcceptEAPTypes key in the Wi-Fi payload of the configuration profile.

When configuring the TOE to utilize EAP-TLS as part of a Wi-Fi Protected Access 2 (WPA2) protected Wi-Fi-network, the CA certificate(s) to which the server's certificate must chain can be configured using the PayLoadCertificateAnchorUUID key in the Wi-Fi payload of the configuration profile.

Using the TLSAllowTrustExceptions key in the Wi-Fi payload of the configuration profile the administrator can enforce that untrusted certificates are not accepted and the authentication fails if such an untrusted certificate is presented.

The TOE provides mobile applications API service for TLS versions 1.0 and 1.2 including support for following ciphersuites:

1. TLS_RSA_WITH_AES_128_CBC_SHA as defined in RFC 5246
2. TLS_RSA_WITH_AES_256_CBC_SHA as defined in RFC 5246
3. TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA as defined in RFC 4492
4. TLS_ECDHE_RSA_WITH_AES_256_CBC_SHA as defined in RFC 4492
5. TLS_ECDHE_ECDSA_WITH_AES_128_CBC_SHA as defined in RFC 4492
6. TLS_ECDHE_ECDSA_WITH_AES_256_CBC_SHA as defined in RFC 4492
7. TLS_RSA_WITH_AES_128_CBC_SHA256 as defined in RFC 5246
8. TLS_RSA_WITH_AES_256_CBC_SHA256 as defined in RFC 5246
9. TLS_ECDHE_ECDSA_WITH_AES_128_CBC_SHA256 as defined in RFC 5289
10. TLS_ECDHE_ECDSA_WITH_AES_256_CBC_SHA384 as defined in RFC 5289
11. TLS_ECDHE_ECDSA_WITH_AES_128_GCM_SHA256 as defined in RFC 5289
12. TLS_ECDHE_ECDSA_WITH_AES_256_GCM_SHA384 as defined in RFC 5289

When an application uses the provided APIs to attempt to establish a trusted channel, the TOE will compare the DN contained within the peer certificate (specifically the Subject CN, as well as any Subject Alternative Name fields, IP Address, or Wildcard certificate if applicable) to the DN of the requested server. If the DN in the certificate does not match the expected DN for the peer, then the application cannot establish the connection.

Applications can request from the TOE the list of ciphersuites supported and then define which of the supported ciphersuites they enable for the TLS protected session they are going to set up. Once the connection has been set up the application can retrieve the ciphersuite negotiated with the communication partner.

When setting up a TLS session, the library function for the handshake (SSLHandshake) will indicate, via a result code, any error that occurred during the certificate chain validation. Communication between the Agent and the MDM Server is protected by TLS using the above supported cipher specifications.

8.8.2 AlwaysOn VPN

For managed and supervised devices, the TOE has the option to be configured with an 'AlwaysOn' VPN where the organization has full control over device traffic by tunneling all IP traffic back to the organization using an Internet Key Exchange (IKE) v2 based IPSec tunnel. A specific set of configuration key values dedicated to the VPN type 'AlwaysOn' allows the specification of the interfaces (cellular or Wi-Fi) for which the VPN is 'AlwaysOn' (default is: for both), the specification of exceptions from this service (only VoiceMail and Airprint can be listed as exceptions), and the definition of exceptions for Captive networking (if any).

For IKEv2 the configuration contains (among other items):

- The IP address or hostname of the VPN server
- The identifier of the IKEv2 client in one of the supported formats
- The authentication method (shared key or certificate)
- The server certificate (for certificate based authentication)

- If extended authentication is enabled
- The encryption algorithm (allows for 3DES (default), AES-128, and AES-256. Single DES shall not be used in the evaluated configuration)
- The integrity algorithm (allows for SHA1-96 (default), SHA1-128, SHA2-256, SHA2-384, SHA2-512)

8.8.3 Bluetooth

The TOE supports Bluetooth 4.0 with the following Bluetooth profiles:

- Hands-Free Profile (HFP 1.6)
- Phone Book Access Profile (PBAP)
- Advanced Audio Distribution Profile (A2DP)
- Audio/Video Remote Control Profile (AVRCP 1.4)
- Personal Area Network Profile (PAN)
- Human Interface Device Profile (HID)
- Message Access Profile (MAP)

Users can pair their device with a remote Bluetooth device using the 'Set up Bluetooth Device' option from the Bluetooth status menu. They can also remove a device from the device list.

Manual authorization is implicitly configured since pairing can only occur when explicitly authorized through the Settings»Bluetooth interface. During the pairing time, another device (or the iOS) can send a pairing request. Commonly, a six digit number is displayed on both sides which must be manually matched by a user, i.e. the PIN is shown and the user must accept it before the pairing completes. If one device does not support this automatic exchange of a PIN, a window for entering a manual PIN is shown. That PIN must match on both sides.

iOS requires that remote Bluetooth devices support an encrypted connection. Devices that want to pair with the TOE via Bluetooth are required by Apple to use Secure Simple Pairing, which uses ECDH based authentication and key exchange. See the Specification of the Bluetooth System 4.0, Volume 2, Part H, chapter 7 [BT] for details. No data can be transferred via Bluetooth until pairing has been completed.
The only time the device is Bluetooth discoverable is when the Bluetooth configuration panel is active and in the foreground (there is no toggle switch for discoverable or not discoverable—unless the configuration panel is the active panel, the device is not discoverable).

Connections via Bluetooth/LE are secured using AES-128 in counter with CBC-MAC (CCM) mode. A local database is kept of all Bluetooth device addresses for paired devices which is checked prior to any automatic connection attempt. Additionally, Bluetooth devices may not establish more than one connection. Multiple connection attempts from the same BD_ADDR for an established connection will be discarded. For details of the security of Bluetooth/LE see the Specification of the Bluetooth System 4.0 [BT], Volume 6, Part E, chapter 1.

8.8.4 Wireless LAN

The TOE implements the wireless LAN protocol as defined in IEEE 802.11 (2012). The TOE uses the random number generator of the CoreCrypto cryptographic module for the generation of keys and other random values used as part of this protocol.

As required by IEEE 802.11 (2012), the TOE implements the CTR with CBC-MAC protocol (CCMP) with AES (128-bit key) as defined in section 11.4.3 of 802.11. This protocol is mandatory for IEEE 802.11 (2012) and is also the default protocol for providing confidentiality and integrity for wireless LANs that comply with IEEE 802.11. The implementation of the AES algorithm is performed by the bulk encryption operation of the WLAN chip.

AES key wrapping as defined in NIST SP 800-38F is used to wrap the Group Temporal Key (GTK), which is sent in an Extensible Authentication Protocol (EAPOL) key frame in message three of the 4-way handshake defined in section 11.6.2 of IEEE 802.11 (2012).

AES key unwrapping is performed as described in NIST SP 800-38F section 6.1, Algorithm 2: W^-1(C), and in section 6.2, Algorithm 4: KW-AD(C).

W^-1

- Initialize the variables
    - Let s = 6(n-1)
    - Let C_1, C_2, ..., C_n be the semiblocks such that C = C_1 || C_2 || ... || C_n
    - Let A^s = C_1
    - For i = 2, ..., n: let _i^s = C i
- Calculate the intermediate values
    - A^(t-1) = MSB_(64)(CIPH^(-1)_K ((A^t ⊕ [t] 64) || R_n^t))
    - R_2^(t-1) = LSB_(64)(CIPH^(-1)_K ((A^t ⊕ [t] 64) || R_n^t))
    - For i = 2, ... , n-1, R_(i+1)^(t-1) = R_i^t
- Output the results
    - Let S_1 =A^0
    - For i = 2, ... , n: S_i = R_i^0
    - Return S_1 || S_2 || ... || S_n

Figure 3 of SP 800-38F depicts how the unwrapping is accomplished.

KW-AD(C)

- Let ICV1 = 0xA6A6A6A6A6A6A6A6
- Let S = W^-1(C)
- If MSB_64(S) != ICV1, then return FAIL and stop
- Return P = LSB_(64(n-1))(S)

The TOE implements WPA2 Enterprise, certified by the Wi-Fi Alliance, certificates WFA20103, WFA20606, WFA20607, WFA55890, WFA55891, WFA55892, WFA56011, and WFA56012.

8.9 Security Audit (FAU)

8.9.1 Audit Records

iOS logging capabilities are able to collect a wide array of information concerning TOE usage and configuration. The available commands and responses constitute audit records and must be configured by TOE administrators using profiles that are further explained in section 8.5.2. These profiles must also be used to determine the audit storage capacity as well as default action when capacity is reached.

Although the specific audit record format is determined via configuration profile, the following attributes form the baseline:

- Date and time the audit record was generated
- Process ID or information about the cause of the event
- Information about the intended operation
- Success or failure (where appropriate)

Audit record information is not available to TOE users or administrators on TOE devices and is only accessible externally on trusted workstations via the Apple Configurator or to an MDM server on enrolled devices.

Depending on the underlying OS of the trusted workstation or MDM server, the audit records are transferred to the following locations:

- macOS:

    - ~/Library/Logs/CrashReporter/MobileDevice/[Your_Device_Name]/
- Windows 10, 8, 7, and Vista:
    - C:\Users\[Your_User_Name]\AppData\Roaming\Apple Computer\Logs\CrashReporter\MobileDevice\[Your_Device_Name]\

8.9.2 MDM Agent Alerts

The MDM agent generates and sends an alert in response to an MDM server request (i.e., applying a policy, receiving a reachability event). The Status key field in Table 9: Wi-Fi CAVS Certificates is used as the alert message to satisfy the FAU_ALT_EXT.2 requirements. The MDM Agent's response is being used as the alert transfer mechanism.

When a Configuration Profile is sent to an MDM Agent, the MDM Agent responds using an "Alert", the *MDM Result Payload*, a plist-encoded dictionary containing the following keys, as well as other keys returned by each command.

<table>
<tr><th>Key</th><th>Type</th><th>Content</th></tr>
<tr><td>Status</td><td>String</td><td>Status. Legal values are described as:
<table>
<tr><th>Status value</th><th>Description</th></tr>
<tr><td>Acknowledged</td><td>Everything went well.</td></tr>
<tr><td>Error</td><td>An error has occurred. See the ErrorChain for details.</td></tr>
<tr><td>CommandFormatError</td><td>A protocol error has occurred. The command may be malformed.</td></tr>
<tr><td>Idle</td><td>The device is idle (there is no status).</td></tr>
<tr><td>NotNow</td><td>The device received the command, but cannot perform it at this time. It will poll the server again in the future.</td></tr>
</table></td></tr>
<tr><td>UDID</td><td>String</td><td>UDID of the device</td></tr>
<tr><td>CommandUUID</td><td>String</td><td>UUID of the command that this response is for (if any)</td></tr>
<tr><td>ErrorChain</td><td>Array</td><td><i>Optional</i>—Array of dictionaries representing the chain of errors that occurred</td></tr>
</table>

Table 15: MDM Agent Status Commands

During installation:

- The user or administrator tells the device to install an MDM payload. The structure of this payload is described in the Structure of MDM Payloads section of [iOS_MDM].
- The device connects to the check-in server. The device presents its identity certificate for authentication, along with its UDID and push notification topic.

Note: Although UDIDs are used by MDM, the use of UDIDs is deprecated for iOS apps.

If the server accepts the device, the device provides its push notification device token to the server. The server should use this token to send push messages to the device. This check-in message also contains a PushMagic string. The server must remember this string and include it in any push messages it sends to the device.

During normal operation:

- The server (at some point in the future) sends out a push notification to the device.
- The device polls the server for a command in response to the push notification.
- The device performs the command.
- The MDM Agent contacts the server to report the result of the last command and to request the next command.

From time to time, the device token may change. When a change is detected, the device automatically checks in with the MDM Server to report its new push notification token.

Note: The device polls only in response to a push notification; it does not poll the server immediately after installation. The server must send a push notification to the device to begin a transaction.

The MDM Agent initiates communication with the MDM Server in response to a push notification by establishing a TLS connection to the MDM Server URL. The MDM Agent validates the server's certificate, and then uses the identity specified in its MDM payload as the client authentication certificate for the connection.

When an MDM Server wants to communicate with iPhone or iPad, a silent notification is sent to the MDM Agent via the Apple Push Notification (APN) service, prompting it to check in with the server. The process of notifying the MDM Agent does not send any proprietary information to or from the APNS. The only task performed by the push notification is to wake the device so it checks in with the MDM Server.

8.9.2.1 Queuing of Alerts

In cases where the TLS channel is unavailable, for example because the device is out of range of a suitable network, an alert in regard to the successful installation of policies is queued until the device is able to communicate with the server again. The queue cannot become long, because if the device is out of communication with the MDM server no additional requests can be received. If the MDM Server does not receive the alert, the MDM Server should re-initiate the transfer until a response is received from the device.

There are certain times when the device is not able to do what the MDM Server requests. For example, databases cannot be modified while the device is locked with Data Protection. When a device cannot perform a command due to these types of situations, it will send the NotNow status without performing the command. The server may send another command immediately after receiving this status, but chances are the following command will also be refused.

After sending a NotNow status, the device will poll the server at some future time. The device will continue to poll the server until a successful transaction is completed.

The device does not cache the command that was refused. If the server wants the device to retry the command, it must send the same command again later, when the device polls the server.

The server does not need to send another push notification in response to this status. However, the server may send another push notification to the device to have it poll the server immediately.

The following commands are guaranteed to execute on iOS, and never return NotNow.

- DeviceInformation

- ProfileList
- DeviceLock
- EraseDevice
- ClearPasscode
- CertificateList
- ProvisioningProfileList
- InstalledApplicationList
- Restrictions

8.9.2.2 Alerts on successful application of policies

Candidate policies are generated by the administrator and disseminated as a configuration profile using one of the methods already described in section 8.5.2 above.

The protocol for managing configuration profiles between the MDM Server and the MDM Agent is defined by the MDM Protocol [iOS_MDM].

When the application of policies to a mobile device is successful the MDM Agent replies with an MDM Result Payload with Status value "Acknowledged".

If a policy update is not successfully installed then the MDM Agent replies with an MDM Result Payload with Status value "Error" or CommandFormatError, "Idle" and "NotNow".

8.9.2.3 Alerts on receiving periodic reachability events

Periodic reachability events are initiated by the MDM Server using Push Notifications. When a periodic reachability event is received the MDM Agent contacts the server in the manner described in section 8.9.1, above.

8.10 Mapping to the Security Functional Requirements

The following table provides a mapping of the SFRs defined in chapter 6 of this [ST] to the functions implemented by the TOE, referring to the previous sections of this TSS where additional information is required.

SFR	Comment
FAU_ALT_EXT.2	Alerts are implemented as described in the instantiation of the MDM protocol, especially MDM Command payloads and MDM Result payloads. Please see section 8.9 for further details about alerts. The use of trusted channels for alerts is described in section 8.8
FAU_GEN.1	Audit data generation is provided by iOS logging functionality in accordance with events enumerated in Table 3: Audit Events.
FAU_SEL.1	Audit data generation is configurable by profile using the Apple Configurator or MDM Protocol.
FAU_STG.1	Audit data is not accessible on TOE devices.
FAU_STG.4	Audit capacity is determined by configuration profile and can remove the oldest audit record when the audit trail is full.
FCS_CKM.1(1)	RSA and elliptic curve cryptography (ECC) key generation are functions of the Apple iOS CoreCrypto Modules. The modules can generate RSA key pairs with modulus sizes of 2048 to 3072, as provided by the PP, in increments of 32 bits. ECC key pairs can be generated for NIST curves P-256, P-384. The following CAVS Certificates are applicable: RSA: 2264, 2265, 2266, 2267, 2268, 2269, 2270, 2271, 2272, 2273, 2274, 2299. ECDSA: 957, 958, 959, 960, 961, 962, 963, 964, 965, 966, 967, 997
FCS_CKM.1(2)	PRF-384 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 in the TOE as part of the WPA implementation and is used for the key generation of AES keys when the Counter Mode CBC-MAC Protocol (CCMP) cipher (defined in section 11.4.3.1 of IEEE 802.11-2012) is used. The Random Bit Generator used is the one of the main device. Compliance to the requirements of FCS_RBG_EXT.1 has been analyzed in the proprietary EAR which has been provided to NIAP. See section 8.8.4 for Wi-Fi Alliance certificates.
FCS_CKM.2(1)	RSA and ECC based key establishment are functions of the Apple iOS CoreCrypto Modules. The following CAVS Certificates are applicable: 959, 960, 961, 962, 963, 964, 965, 966, 967, 968, 969, 1010
FCS_CKM.2(2)	AES Key Wrap in an EAPOL-Key frame is implemented as defined in IEEE 802.11-2012, "Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications", sections 11.6.2 and 11.6.3. Option 2 in 11.6.2 of IEEE 802.11-2012 which describes the value for 2 or 3 of the Key Descriptor Version subfield of the Key Information field of the EAPOL key frame. This value indicates that HMAC-SHA1-128 is the EAPOL-Key MIC and NIST AES key wrap is used for EAPOL-Key encryption. This is implemented by the TOE. The following CAVS Certificates are applicable:

SFR	Comment
FCS_CKM_EXT.1	The TOE supports a device unique REK (256-bit AES keys) that is generated in the production environment and installed during production. This REK is protected by the hardware such that it can only be used for encryption and decryption operation and only in the Secure Enclave. It cannot be read or modified by any software or firmware in the TOE. The proprietary EAR, which has been provided to NIAP, has analyzed the random bit generator (RBG) used in the production environment for compliance to the requirements defined in FCS_RBG_EXT.1
FCS_CKM_EXT.2	All the data encryption keys (DEKs) described in the key hierarchy diagram in Figure 4: Key Hierarchy in iOS are generated using the RBG of the secure enclave and are AES keys with 256-bit key size or Curve25519 keys with 256-bit key size using the functions of the Apple iOS CoreCrypto Kernel Module. The Key Generation functions are pre-programmed to use only this RBG and key-size, which cannot be changed.
FCS_CKM_EXT.3	All the KEKs described in the key hierarchy diagram in Figure 4: Key Hierarchy in iOS are generated using the RBG of the secure enclave and are AES keys with 256-bit key size using the functions of the Apple iOS CoreCrypto Kernel Module. The Passcode Key is generated using PBKDF with the UID as the key and a salt that is generated using the RBG in the Secure Enclave. This RBG in the Secure Enclave has been analyzed for compliance with the requirements of FCS_RBG_EXT.1 in the proprietary EAR, which has been provided to NIAP.
FCS_CKM_EXT.4	Keys in volatile memory are overwritten by zeros when no longer needed. Keys in non-volatile flash memory are cleared by a block erase when no longer needed. *Note that in accordance with TD0028 and TD0057 a read-verify is not performed after clearing the keys.*
FCS_CKM_EXT.5	A wipe operation is performed after exceeding the limit number of failed authentication attempts or upon receiving a request from an authorized administrator. Wiping is performed by clearing the block with the highest level of KEKs shown in the diagram in Figure 4: Key Hierarchy in iOS using a block erase of the 'effaceable area' of the non-volatile flash memory.
FCS_CKM_EXT.6	The salt used for the password-based key derivation function is generated using the RBG in the Secure Enclave, which (as noted before) has been analyzed for compliance with the requirements of FCS_RBG_EXT.1 in the proprietary EAR, which has been provided to NIAP. Other salts and random values are generated using the RBG of the main device.
FCS_COP.1(1)	AES-CBC and AES-GCM are implemented as approved functions in the Apple iOS CoreCrypto Modules as well as in the Wi-Fi chip. AES-CCM and AES Key Wrap are implemented as non-approved functions in the Apple iOS CoreCrypto Modules. AES-CCMP is implemented as defined in IEEE 802.11-2012, "Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications", section 11.4.3. See section 8.8.4 for Wi-Fi Alliance certificates. Cryptographic key sizes are 128-bit and 256-bit. The following CAVS Certificates are applicable: 4156, 4157, 4158, 4159, 4160, 4161, 4162, 4163, 4164, 4166, 4169, 4170, 4255, 4256, 4257, 4258, 4259, 4260 and all certificates listed in Table 9: Wi-Fi CAVS Certificates

SFR	Comment
FCS_COP.1(2)	SHA-1, SHA-256, SHA-394, and SHA-512 are implemented as approved functions in the Apple iOS CoreCrypto Modules. The following CAVS Certificates are applicable: 3431, 3432, 3433, 3434, 3435, 3436, 3437, 3438, 3439, 3440, 3441, 3442, 3557, 3558, 3559, 3560, 3561, 3562
FCS_COP.1(3)	RSA and ECDSA signature generation are implemented as approved functions in the Apple iOS CoreCrypto Modules. ECDSA using NIST curves P-256 and P-384 is implemented as an approved function. Message digest sizes are 160 and 256, 384, 512 bits The following CAVS Certificates are applicable: RSA: 2264, 2265, 2266, 2267, 2268, 2269, 2270, 2271, 2272, 2273, 2274, 2299. ECDSA: 957, 958, 959, 960, 961, 962, 963, 964, 965, 966, 967, 997
FCS_COP.1(4)	HMAC-SHA1, HMAC-SHA-256, HMAC-SHA384, and HMAC-SHA-512 are implemented as approved functions in the Apple iOS CoreCrypto Modules. The key attributes used are defined in the SFR, FCS_COP.1.1(4). The following CAVS Certificates are applicable: 2733, 2734, 2735, 2736, 2737, 2738, 2739, 2740, 2741, 2742, 2743, 2744, 2854, 2855, 2856, 2857, 2858, 2859
FCS_COP.1(5)	PBKDF is implemented as a *vendor affirmed* function in the Apple iOS CoreCrypto Modules. The password is fed as a binary string to the SHA algorithm. The settings for the algorithm (padding, blocking, etc.) are described in section 0 of this [ST]. The output of the hash function is used to form the sub-mask and is conditioned to be the same length as the KEK, 256-bits, that is specified in FCS_CKM_EXT.3.
FCS_HTTPS_EXT.1	The TOE provides a library that allows the implementation to set up TLS protected connections. This is used by browsers (including Safari) to implement the HTTPS protocol. The TOE can be configured to not establish a connection if the peer certificate is deemed invalid.
FCS_IV_EXT.1	All initialization vectors are implemented as defined in table 14 of [PP_MD_V3.0] and [EP_MDM_AGENT_V3.0].
FCS_RBG_EXT.1	There are two Random Bit Generators implemented in the TOE that are used for security functionality specified in this [ST]: one in the Secure Enclave (used to generate the salt used in the PBKDF for generating the Passcode Key), and one in the main device (used for the generation of all other random bits used for security functionality specified in the [ST]). Both RBGs are based on hardware noise sources and both have been analyzed in the proprietary EAR, which has been provided to NIAP, to have a minimum of 256 bits of entropy. The cryptographic library provides an interface for application programs that those programs can use to request random bits from the RBG in the main device. The following CAVS Certificates are applicable: 1264, 1265, 1266, 1267, 1268, 1269, 1270, 1271, 1272, 1273, 1274, 1275 and all certificates listed in Table 10: Hardware DRBG CAVS Certificates
FCS_SRV_EXT.1	All functions mentioned can be accessed by applications via interfaces to the cryptographic library.

SFR	Comment
FCS_STG_EXT.1	Cryptographic keys are stored in keychains. Keys may be installed by an administrator as part of a configuration profile of may be installed by an application. Access control to keychain items is described in section 8.3.6, *Keychain Data Protection* in this [ST].
FCS_STG_EXT.2	The key hierarchy implemented by the TOE is explained in section 8.2.1 and Figure 4 of this [ST]. All keys are wrapped using AES in Key Wrap mode with 256-bit keys.
FCS_STG_EXT.3	The integrity of wrapped keys is implemented through the MAC before the unwrapped key is used.
FCS_STG_EXT.4	iOS provides platform-provided key storage for keys which is utilized by the MDM Agent by calling the iOS API.
FCS_TLSC_EXT.1	EAP TLS with TLS 1.0 only. TLS 1.2 is implemented by the TOE including all mandatory and optional ciphersuites listed in [PP_MD_V3.0]. A client certificate can be installed on the device via a configuration profile.
FDP_ACF_EXT.1	The TOE implements an access control policy to data as described in section 8.3.1(for files) and section 8.3.2(for keychain items) of this [ST].
FDP_DAR_EXT.1	All data-at-rest is protected using the class key of the class the file belongs to. Similar all keychain items are protected by their class key. Figure 4 explains the KEKs used to protect the class keys.
FDP_DAR_EXT.2	The class key for the NSFileProtectionCompleteUnlessOpen is used for the case of protecting data when the device is locked. The description of the NSFileProtectionCompleteUnlessOpen class describes the use of the asymmetric key pair over Curve25519 for this purpose.
FDP_IFC_EXT.1	The TOE can be configured (using a Configuration Profile) such that all IP traffic flows through an IPsec VPN client (AlwaysOn VPN).
FDP_STG_EXT.1	The Trust Anchor Database is stored as a protected item.
FDP_UPC_EXT.1	The TOE provides a library that applications can use to set up a TLS protected connection, set up a HTTPS protected connection, use Bluetooth Basic Rate/Enhanced Data Rate (BR/EDR) and Bluetooth Low Energy (LE) to set up a protected communication channel and initiate communication via this channel.
FIA_AFL_EXT.1	The TSF can be configured (using a Configuration Profile) to limit the number of consecutive failed authentication attempts to any number between 2 and 10.
FIA_BLT_EXT.1	The TSF can be configured to require explicit user authorization before pairing with a remote Bluetooth device. See section 8.8.3 of this [ST] for more detail.
FIA_ENR_EXT.2	The MDM Agent records the reference ID of the MDM Server in flash memory. See section 8.4.2
FIA_PAE_EXT.1	The TOE conforms to IEEE 802.1X for Port Access Entity (PAE) in the Supplicant role.
FIA_PMG_EXT.1	The TSF 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 TSF enforces a delay of at least 5 seconds between user authentication attempts.
FIA_UAU.7	When a password is entered, the character entered is shown for a short time in clear and then replaced by a dot.

SFR	Comment
FIA_UAU_EXT.2	The TSF can be configured to allow no actions to be performed before the user is authenticated.
FIA_X509_EXT.1	X.509 certificates are validated 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. Certificate Authority (CA) certificates are only accepted if the basicConstraints extension exists and the CA flag in this extension is set. The Online Certificate Status Protocol (OCSP) is supported to determine if a certificate has been revoked.
FIA_X509_EXT.2	X.509v3 certificates are supported for authentication for EAP-TLS, IPsec, TLS, HTTPS, and code signing for software updates, code signing for mobile applications.
FIA_X509_EXT.3	The library provided for certificate handling and management provides a service for certificate validation. The return value indicates the success or failure of the validation with detailed failure codes indicating why the validation failed.
FMT_MOF_EXT.1	Table 4: Management Functions in this [ST] indicates the management functions and the entity (user or administrator) that can perform the function.
FMT_POL_EXT.2	MDM policies can be validated against enterprise signatures as outlined in section 8.5.2 Configuration Profiles.
FMT_SMF_EXT.1	Table 4: Management Functions indicates the management functions that can be performed by either the administrator, the user, or both. In the case of the administrator functions most of them (except for remote wipe and remote locking) are performed by installing Configuration Profiles on the TOE. Administration functions for the user are available through the user interface.
FMT_SMF_EXT.2	Upon unenrollment and when a remote wipe command is issued protected data is wiped by erasing the top level KEKs. Since all data-at-rest is encrypted with one of those keys, the device is wiped.
FMT_SMF_EXT.3	The MDM Agent is integrated with the platform and provides an interface to ensure that administrator-provided management functions and certificate import are performed. See section 8.5.
FMT_UNR_EXT.1	Unenrollment is handled as described in section 8.5.3
FPT_AEX_EXT.1	Address space layout randomization is used for every sandbox used to execute applications in. There are 8 bits of randomness taken from the application processor TRNG involved in the randomization.
FPT_AEX_EXT.2	Hardware enforced memory protection is used for every memory page.
FPT_AEX_EXT.3	Stack-based buffer overflow protection is implemented for every sandbox.
FPT_AEX_EXT.4	The TSF protects itself as well as applications from modification by untrusted subjects. Applications execute in their own address space separated from the address space of other applications and separated from the address space used by the TSF. The TOE does not support Unstructured Supplementary Service Data (USSD) or Man-Machine Interface (MMI) codes and also does not support auxiliary boot modes.
FPT_JTA_EXT.1	JTAG ports are not provided

SFR	Comment
FPT_KST_EXT.1	Keys are stored in keychains and are therefore encrypted by one of the KEKs for keychain objects.
FPT_KST_EXT.2	Keys are never transmitted in plaintext outside the TOE by the TSF.
FPT_KST_EXT.3	Plaintext keys can also not be exported by TOE users. There is no function that allows this.
FPT_NOT_EXT.1	In the case of a self-test failure or software integrity verification failure the device will not start. Self-tests for cryptographic functions are performed by the cryptographic modules.
FPT_STM.1	The TSF maintains a time stamp for its own use.
FPT_TST_EXT.1	The TSF runs a suite of self-tests during initial start-up, as required by FIPS 140-2, for all cryptographic functionality.
FPT_TST_EXT.2	The TSF verifies the integrity of the boot chain as described in section 8.6.1of this [ST].
FPT_TUD_EXT.1	Users can query the current version of the TOE firmware and software as well as the current version of the hardware model via the user interface. Users can also query the current version of installed mobile applications via the user interface.
FPT_TUD_EXT.2	All software updates and all applications need to be digitally signed. This signature is verified before installing any update or application. Details of this process are described in section 8.6.3 of this ST.
FTA_SSL_EXT.1	A Configuration Profile can be used to define the maximum time of inactivity before the TSF initiates a transition to the locked state. In addition, both the user and the administrator can initiate a transition to the locked state. During such a transition the display is overwritten and the decrypted class key for the NSFileProtection Complete class is zeroized.
FTA_WSE_EXT.1	Management of acceptable WLAN attributes can be configured via profile outlined in section 8.5.2.
FTP_ITC_EXT.1	The TSF provides the specified list of protocols that allow for the establishment of a trusted channel between itself and another trusted IT product. It is always the TSF that initiates such a communication via a trusted channel.
FTA_TAB.1	It is possible for an enterprise to define the display of a managed device that is provided before a user session is established. This can include an enterprise defined warning message regarding unauthorized use of the TOE.

Table 16: Mapping of SFRs

9 List of SFRs in the ST

This list is intended to help assist the evaluator/validator. It can be removed before this [ST] is published.

Abbreviations and Acronyms

A2DP	Advanced Audio Distribution Profile
ACL	Access Control List
AES	Advanced Encryption Standard
API	Application Programmer Interface
APN	Apple Push Notification
APNS	Apple Push Notification Service
ARM	Advanced RISC Machine
ASLR	Address Space Layout Randomization
AVRCP	Audio/Video Remote Control Profile
BR/EDR	Basic Rate/Enhanced Data Rate
CAVS	Cryptographic Algorithm Validation System
CBC	Cypher Block Chaining
CC	Common Criteria
CCM	Counter with CBC-MAC
CCMP	Counter Mode CBC-MAC Protocol
CDMA	Code Division Multiple Access
CN	Common Name
CTR	
CVE	Common Vulnerabilities and Exposures
DC-HSDPA	Dual-Carrier High Speed Packet Access
DEK	Data Encryption Key
DEP	Device Enrollment Program
DES	Data Encryption Standard
DFU	Device Firmware Upgrade
DH	Diffie-Hellman
DN	Distinguished Name
DNS	Domain Name Server
DRBG	Deterministic Random Bit Generator
DRNG	Deterministic Random Number Generator
DSP	
EAL	Evaluation Assurance Level
EAPOL	Extensible Authentication Protocol Over LAN
EAP-TLS	Extensible Authentication Protocol Transport Layer Security
EAR	Entropy Assessment Report
EC	Elliptic Curve
ECC	Elliptic Curve Cryptography
ECDH	Elliptic Curve Diffie-Hellman
ECDSA	Elliptic Curve Digital Signal Algorithm
ECID	Electronic Chip Identification
EP	Extended Package (for a Protection Profile)
EV-DO	Evolution-Data Optimized
FDD-LTE	Frequency-Division Duplex-Long Term Evolution
FIA	Identification and Authentication

FIPS	Federal Information Processing Standard
GCM	Galois/Counter Mode
GID	Group Key
GPS	Gobal Positioning Satellites
GSM	Global System for Mobile Communication
GTK	Group Temporal Key
HFP	Hands-Free Profile
HID	Human Interface Device Profile
HMAC	Keyed-hash Message Authentication Code
HSPA+	High Speed Packet Access Plus
IKE	Internet Key Exchange
IPsec	Internet Protocol Security
IV	Initialization Vector
JSON	JavaScript Object Notation
JTAG	Joint Test Action Group
KAT	Known Answer Test
KDF	Key Derivation Function
KEK	Key Encryption Key
KW	
LE	Low Energy
LLB	Low-Level Bootloader
LTE	Long Term Evolution
MAC	Message Authentication Code
MAP	Message Access Profile
MD	Mobile Device
MDF	Mobile Device Fundamentals
MDFPP	Mobile Device Fundamentals Protection Profile
MDM	Mobile Device Management
MMI	Man-Machine Interface
MMU	Memory Management Unit
NITZ	Network, Identity and Time Zone
NTP	Network Time Protocol
OCSP	Online Certificate Status Protocol
OSP	Organizational Security Policy
PAE	Port Access Entity
PAN	Personal Area Network Profile
PBAP	Phone Book Access Profile
PBKDF	Password-Based Key Derivation Function
PHY	Physical Layer
PKCS	Public Key Cryptography Standards
PKG	
PKV	
PRF	Pseudorandom Function
RBG	Random Bit Generator
REK	Root Encryption Key

RFC	Request for Comment
RISC	Reduced Instruction Set Computing
RSA	Rivest-Shamire-Adleman
S/MIME	Secure/Multipurpose Internet Mail Extensions
SAR	Security Assurance Requirement
SCEP	Simple Certificate Enrollment Protocol
SFR	Security Functional Requirement
SHA	Secure Hash Algorithm
SHS	Secure Hash Standard
SIG	
SP	
SSID	Service Set Identifier
SSL	Secure Sockets Layer
ST	Security Target
TD-LTE	Time Division Long-Term Evolution
TD-SCDMA	Time Division Synchronous Code Division Multiple Access
TLS	Transport Layer Security
TOE	Target of Evaluation
Triple-DES	
TRNG	True Random Number Generators
TSF	TOE Security Functionality
TSS	TOE Summary Specification
UI	User Interface
UMTS	Universal Mobile Telecommunications System
USSD	Unstructured Supplementary Service Data
UUID	Universally Unique ID
VPN	Virtual Private Network
VPP	Volume Purchase Program
WLAN	Wireless Local Area Network
WPA2	Wi-Fi Protected Access 2
XN	Execute Never