Microsoft Windows

Common Criteria Evaluation

Microsoft Windows 8.1

Microsoft Windows Phone 8.1

Security Target

Document Information	
Version Number	1.0
Updated On	August 21, 2015

This is a preliminary document and may be changed substantially prior to final commercial release of the software described herein.

The information contained in this document represents the current view of Microsoft Corporation on the issues discussed as of the date of publication. Because Microsoft must respond to changing market conditions, it should not be interpreted to be a commitment on the part of Microsoft, and Microsoft cannot guarantee the accuracy of any information presented after the date of publication.

This document is for informational purposes only. MICROSOFT MAKES NO WARRANTIES, EXPRESS OR IMPLIED, AS TO THE INFORMATION IN THIS DOCUMENT.

Microsoft may have patents, patent applications, trademarks, copyrights, or other intellectual property rights covering subject matter in this document. Except as expressly provided in any written license agreement from Microsoft, the furnishing of this document does not give you any license to these patents, trademarks, copyrights, or other intellectual property.

The example companies, organizations, products, people and events depicted herein are fictitious. No association with any real company, organization, product, person or event is intended or should be inferred.

Microsoft, Active Directory, Visual Basic, Visual Studio, Windows, the Windows logo, Windows NT, and Windows Server are either registered trademarks or trademarks of Microsoft Corporation in the United States and/or other countries.

The names of actual companies and products mentioned herein may be the trademarks of their respective owners.

TABLE OF CONTENTS

LIST OF TABLES

1 Security Target Introduction

This section presents the following information required for a Common Criteria (CC) evaluation:

- Identifies the Security Target (ST) and the Target of Evaluation (TOE);
- Specifies the security target conventions and conformance claims; and,
- Describes the organization of the security target.

1.1 Security Target, TOE, and Common Criteria (CC) Identification

ST Title: Microsoft Windows 8.1and Windows Phone 8.1 Security Target

ST Version: version 1.0, August 21, 2015

TOE Software Identification: The following Windows Operating Systems (OS):

- Microsoft Windows 8.1 Pro Edition (64-bit version)
- Microsoft Windows Phone 8.1 GDR2

The following security updates and patches must be applied to the above Windows 8.1 products:

- All critical updates as of April 30, 2015

The following security updates must be applied to the above Windows Phone 8.1 products:

- All critical updates as of April 30, 2015

TOE Hardware Identification: The following hardware platforms and components are included in the evaluated configuration:

- **Microsoft Surface 3,** Windows 8.1 Pro, 64-bit, Intel Atom Z8700, Marvell 8897 Wi-Fi a/b/g/n adapter, Bluetooth 4.0, Bluetooth LE, Intel TPM 2.0
- **Microsoft Lumia 635**, Windows Phone 8.1, Qualcomm Snapdragon 400, GSM, HSPA, LTE, Qualcomm WCN3620 Wi-Fi b/g/n adapter, Qualcomm TPM 2.0
- **Microsoft Lumia 830**, Windows Phone 8.1, Qualcomm Snapdragon 400, GSM, HSPA, LTE, Qualcomm WCN3620 Wi-Fi b/g/n adapter, Qualcomm TPM 2.0

All devices include IEEE 802.11 Wi-Fi and Bluetooth 4.0.

TOE Guidance Identification: The following administrator, user, and configuration guides were evaluated as part of the TOE:

- *Common Criteria Supplemental Admin Guidance* along with all the documents referenced therein.

Evaluation Assurance: As specified in section 5.2.1 and specific Assurance Activities associated with the security functional requirements from section 5.2.2.

CC Identification: CC for Information Technology (IT) Security Evaluation, Version 3.1, Revision 4, September 2012.

1.2 CC Conformance Claims

This TOE and ST are consistent with the following specifications:

- Common Criteria for Information Technology Security Evaluation Part 2: Security functional requirements, Version 3.1, Revision 4, September 2012, extended (Part 2 extended)
- Common Criteria for Information Technology Security Evaluation Part 3: Security assurance requirements Version 3.1, Revision 4, September 2012, extended with ALC_TSU_EXT.1
- Protection Profile for Mobile Device Fundamentals, Version 1.1, February 12, 2014 (MDF PP)
- CC Part 3 assurance requirements specified in Section 5.2.1 and Evaluation Assurance Activities specified in section 5.2.2

1.3 Conventions, Terminology, Acronyms

This section specifies the formatting information used in the security target.

1.3.1 Conventions

The following conventions have been applied in this document:

- Security Functional Requirements (SFRs): Part 2 of the CC defines the approved set of operations that may be applied to functional requirements: iteration, assignment, selection, and refinement.
    - Iteration: allows a component to be used more than once with varying operations.
    - Assignment: allows the specification of an identified parameter.
    - Selection: allows the specification of one or more elements from a list.
    - Refinement: allows the addition of details.

  The conventions for the assignment, selection, refinement, and iteration operations are described in Section 5.

- Other sections of the security target use a bold font to highlight text of special interest, such as captions.

1.3.2 Terminology

The following terminology is used in the security target:

Term	Definition
Access	Interaction between an entity and an object that results in the flow or modification of data.
Access control	Security service that controls the use of resources[1] and the disclosure and modification of data[2].

[1] Hardware and software
[2] Stored or communicated

Accountability	Tracing each activity in an IT system to the entity responsible for the activity.
Active Directory	Active Directory manages enterprise identities, credentials, information protection, system and application settings through AD Domain Services, Federation Services, Certificate Services and Lightweight Directory Services.
Administrator	An authorized user who has been specifically granted the authority to manage some portion or the entire TOE and thus whose actions may affect the TOE Security Policy (TSP). Administrators may possess special privileges that provide capabilities to override portions of the TSP.
Assurance	A measure of confidence that the security features of an IT system are sufficient to enforce the IT system's security policy.
Attack	An intentional act attempting to violate the security policy of an IT system.
Authentication	A security measure that verifies a claimed identity.
Authentication data	The information used to verify a claimed identity.
Authorization	Permission, granted by an entity authorized to do so, to perform functions and access data.
Authorized user	An authenticated user who may, in accordance with the TOE Security Policy, perform an operation.
Availability	Timely[3], reliable access to IT resources.
Compromise	Violation of a security policy.
Confidentiality	A security policy pertaining to disclosure of data.
Critical cryptographic security parameters	Security-related information appearing in plaintext or otherwise unprotected form and whose disclosure or modification can compromise the security of a cryptographic module or the security of the information protected by the module.
Cryptographic boundary	An explicitly defined contiguous perimeter that establishes the physical bounds (for hardware) or logical bounds (for software) of a cryptographic module.
Cryptographic key (key)	A parameter used in conjunction with a cryptographic algorithm that determines: • the transformation of plaintext data into ciphertext data • the transformation of ciphertext data into plaintext data • a digital signature computed from data • the verification of a digital signature computed from data • a data authentication code computed from data
Cryptographic module	The set of hardware, software, and/or firmware that implements approved security functions, including cryptographic algorithms and key generation, which is contained within the cryptographic boundary.
Cryptographic module security policy	A precise specification of the security rules under which a cryptographic module must operate.
Defense-in-depth	A security design strategy whereby layers of protection are utilized to establish an adequate security posture for an IT system.
Discretionary Access Control (DAC)	A means of restricting access to objects based on the identity of subjects and groups to which the objects belong. The controls are discretionary

[3] According to a defined metric

	meaning that a subject with a certain access permission is capable of passing that permission (perhaps indirectly) on to any other subject.
Edition	A distinct variation of a Windows OS version. Examples of editions are Windows Server 2012 [Standard] and Windows Server 2012 Datacenter.
Enclave	A collection of entities under the control of a single authority and having a homogeneous security policy. They may be logical, or based on physical location and proximity.
Entity	A subject, object, user or external IT device.
General-Purpose Operating System	A general-purpose operating system is designed to meet a variety of goals, including protection between users and applications, fast response time for interactive applications, high throughput for server applications, and high overall resource utilization.
Identity	A means of uniquely identifying an authorized user of the TOE.
Integrated Windows authentication	An authentication protocol formerly known as NTLM or Windows NT Challenge/Response.
Named object	• An object that exhibits all of the following characteristics: • The object may be used to transfer information between subjects of differing user identities within the TOE Security Function (TSF). • Subjects in the TOE must be able to request a specific instance of the object. • The name used to refer to a specific instance of the object must exist in a context that potentially allows subjects with different user identities to request the same instance of the object.
Object	An entity under the control of the TOE that contains or receives information and upon which subjects perform operations.
Operating environment	The total environment in which a TOE operates. It includes the physical facility and any physical, procedural, administrative and personnel controls.
Persistent storage	All types of data storage media that maintain data across system boots (e.g., hard disk, removable media).
Public object	An object for which the TSF unconditionally permits all entities “read” access under the Discretionary Access Control SFP. Only the TSF or authorized administrators may create, delete, or modify the public objects.
Resource	A fundamental element in an IT system (e.g., processing time, disk space, and memory) that may be used to create the abstractions of subjects and objects.
SChannel	A security package (SSP) that provides network authentication between clients and servers.
Secure State	Condition in which all TOE security policies are enforced.
Security attributes	TSF data associated with subjects, objects and users that is used for the enforcement of the TSP.
Security-enforcing	A term used to indicate that the entity (e.g., module, interface, subsystem) is related to the enforcement of the TOE security policies.
Security-supporting	A term used to indicate that the entity (e.g., module, interface, subsystem) is not security-enforcing; however, the entity’s implementation must still preserve the security of the TSF.
Security context	The security attributes or rules that are currently in effect. For SSPI, a

	security context is an opaque data structure that contains security data relevant to a connection, such as a session key or an indication of the duration of the session.
Security package	The software implementation of a security protocol. Security packages are contained in security support provider libraries or security support provider/authentication package libraries.
Security principal	An entity recognized by the security system. Principals can include human users as well as autonomous processes.
Security Support Provider (SSP)	A dynamic-link library that implements the SSPI by making one or more security packages available to applications. Each security package provides mappings between an application's SSPI function calls and an actual security model's functions. Security packages support security protocols such as Kerberos authentication and Integrated Windows Authentication.
Security Support Provider Interface (SSPI)	A common interface between transport-level applications. SSPI allows a transport application to call one of several security providers to obtain an authenticated connection. These calls do not require extensive knowledge of the security protocol's details.
Security Target (ST)	A set of security requirements and specifications to be used as the basis for evaluation of an identified TOE.
Subject	An active entity within the TOE Scope of Control (TSC) that causes operations to be performed. Subjects can come in two forms: trusted and untrusted. Trusted subjects are exempt from part or all of the TOE security policies. Untrusted subjects are bound by all TOE security policies.
Target of Evaluation (TOE)	An IT product or system and its associated administrator and user guidance documentation that is the subject of an evaluation.
Threat	Capabilities, intentions and attack methods of adversaries, or any circumstance or event, with the potential to violate the TOE security policy.
Unauthorized individual	A type of threat agent in which individuals who have not been granted access to the TOE attempt to gain access to information or functions provided by the TOE.
Unauthorized user	A type of threat agent in which individuals who are registered and have been explicitly granted access to the TOE may attempt to access information or functions that they are not permitted to access.
Universal Unique Identifier (UUID)	UUID is an identifier that is unique across both space and time, with respect to the space of all UUIDs. A UUID can be used for multiple purposes, from tagging objects with an extremely short lifetime, to reliably identifying very persistent objects across a network.
User	Any person who interacts with the TOE.
User Principal Name (UPN)	An identifier used by Microsoft Active Directory that provides a user name and the Internet domain with which that username is associated in an e-mail address format. The format is [*AD username*]@[associated *domain*]; an example would be *john.smith@microsoft.com.*
Uniform Resource Locator (URL)	The address that is used to locate a Web site. URLs are text strings that must conform to the guidelines in RFC 2396.
Version	A Version refers to a release level of the Windows operating system. Windows 7 and Windows 8 are different versions.

Vulnerability	A weakness that can be exploited to violate the TOE security policy.

1.3.3 Acronyms

The acronyms used in this security target are specified **in Appendix A: List of Abbreviations**

Appendix A: List of Abbreviations.

1.4 ST Overview and Organization

The Windows 8.1 and Windows Phone TOE provides the following security services:

- Cryptographic support
- User data protection
- Identification and Authentication (I&A)
- Protection of the TOE Security Functions (TSF)
- TOE access/session control
- Trusted path/channel
- Security management

This security target contains the following additional sections:

- TOE Description (Section 2): Provides an overview of the TSF and boundary.
- Security Problem Definition (Section 3): Describes the threats, organizational security policies and assumptions that pertain to the TOE.
- Security Objectives (Section 4): Identifies the security objectives that are satisfied by the TOE and the TOE operational environment.
- Security Requirements (Section 5): Presents the security functional and assurance requirements met by the TOE.
- TOE Summary Specification (TSS) (Section 6): Describes the security functions provided by the TOE to satisfy the security requirements and objectives.
- Protection Profile Conformance Claim (Section 7): Presents the rationale concerning compliance of the ST with the ***Protection Profile for Mobile Device Fundamentals***.
- Rationale for Modifications to the Security Requirements (Section 8): Presents the rationale for the security objectives, requirements, and TOE Summary Specification as to their consistency, completeness and suitability.

2 TOE Description

The TOE includes the Windows 8.1 operating system, the Microsoft Windows Phone operating system, supporting hardware, and those applications necessary to manage, support and configure the operating system.

2.1 Product Types

Windows 8.1, and Windows Phone 8.1 are preemptive multitasking, multiprocessor, and multi-user operating systems. In general, operating systems provide users with a convenient interface to manage underlying hardware. They control the allocation and manage computing resources such as processors, memory, and Input/Output (I/O) devices. Windows 8.1and Windows Phone 8.1, collectively referred to as Windows, expand these basic operating system capabilities to controlling the allocation and managing higher level IT resources such as security principals (user or machine accounts), files, printing

objects, services, window station, desktops, cryptographic keys, network ports traffic, directory objects, and web content. Multi-user operating systems such as Windows keep track of which user is using which resource, grant resource requests, account for resource usage, and mediate conflicting requests from different programs and users.

2.2 Product Description

The TOE includes two product variants of Windows 8.1and Windows Phone:

- Windows 8.1 Pro
- Windows Phone 8.1

Windows 8.1 is suited for business desktops, notebook, and convertible computers. It is the workstation product and while it can be used by itself, it is also designed to serve as a client within Windows domains.

Windows Phone 8.1 is based on the same core operating system as Windows 8.1 and provides a simplified user interface that makes Windows Phone a communications hub for voice, text, and web access.

In terms of security, Windows 8.1 and Phone 8.1 share the same security characteristics.

2.3 Security Environment and TOE Boundary

The TOE includes both physical and logical boundaries. Its operational environment is that of a networked environment with IEEE 802.11 (Wi-Fi), mobile broadband networks (3G/4G and LTE) and Bluetooth networks.

2.3.1 Logical Boundaries

The logical boundary of the TOE includes:

- The **Boot Manager**, which is invoked by the computer's bootstrapping code.
- The **Windows Loader** which loads the operating system into the computer's memory.
- **Windows OS Resume** which reloads an image of the executing operating system from a hibernation file as part of resuming from a hibernated state.
- The **Windows Kernel** which contains device drivers for the Windows NT File System, full volume encryption, the crash dump filter, and the kernel-mode cryptographic library.
- The **IPv4 / IPv6 network stack** in the kernel.
- The **IPsec** module in user-mode.
- The **IKE and AuthIP Keying Modules** service which hosts the IKE and Authenticated Internet Protocol (AuthIP) keying modules. These keying modules are used for authentication and key exchange in Internet Protocol security (IPsec).[4]
- The **Remote Access Service** device driver in the kernel, which is used primarily for ad hoc or user-defined VPN connections; known as the "RAS IPsec VPN" or "RAS VPN".

[4] AuthIP key exchange was not examined in the Common Criteria portion of this evaluation.

- The **IPsec Policy Agent** service which enforces IPsec policies.
- **Windows Explorer** for Windows 8.1 which can be used to manage the OS and check the integrity of Windows files and updates.
- The **Windows Phone** shell for Windows Phone 8.1 which can be used to manage the device.
- The **Windows Trusted Installer** which installs updates to the Windows operating system.
- The **Key Isolation Service** which protects secret and private keys.
- The **App Container** which is the execution environment for the Windows Store Applications which are the only applications covered by this evaluation.

2.3.2 Physical Boundaries

Physically, each TOE tablet or phone consists of an ARMv7 Thumb-2or x64 computer architecture. The TOE executes on processors from Intel (x64) or Qualcomm (ARM). Refer to section 1.1 for the specific list of hardware included in the evaluation.

A set of devices may be attached as part of the TOE:

- Display Monitors
- Fixed Disk Drives (including disk drives and solid state drives)
- Removable Disk Drives (including USB storage)
- Network Adaptor
- Keyboard
- Mouse
- Printer
- Audio Adaptor
- CD-ROM Drive
- Smart Card Reader
- Trusted Platform Module (TPM) version 2.0

While this list of devices is larger than is needed to evaluate the requirements in the Mobile Device Fundamentals protection profile, it is the same set of devices as the General Purpose Operating System Protection Profile evaluation for Windows 8. By using the same set of devices for both evaluations, consumers can gain assurance by using both core OS capabilities and Mobile Device Fundamentals in combination.

2.4 TOE Security Services

This section summarizes the security services provided by the TOE:

- **Cryptographic Support:** Windows provides FIPS-140-2 validated cryptographic functions that support encryption/decryption, cryptographic signatures, cryptographic hashing, cryptographic key agreement, and random number generation. The TOE additionally provides support for public keys, credential management and certificate validation functions and provides support for the National Security Agency's Suite B cryptographic algorithms. Windows also provides extensive auditing support of cryptographic operations, the ability to replace cryptographic

functions and random number generators with alternative implementations,[5] and a key isolation service designed to limit the potential exposure of secret and private keys. In addition to using cryptography for its own security functions, Windows offers access to the cryptographic support functions for user-mode and kernel-mode programs. Public key certificates generated and used by Windows authenticate users and machines as well as protect both user and system data in transit.

  - **Software-based disk encryption:** Windows implements BitLocker to provide encrypted data storage for fixed and removable volumes and protects the disk volume's encryption key with one or more intermediate keys and authorization factor
  - **IPsec:** Windows implements IPsec to provide protected, authenticated, confidential, and tamper-proof networking between two peer computers.

- **User Data Protection**: In the context of this evaluation Windows protects user data at rest and provides secure storage of X.509v3 certificates.
- **Identification and Authentication**: In the context of this evaluation, Windows provides the ability to use, store, and protect X.509 certificates that are used for IPsec and authenticates the user to their mobile device.
- **Protection of the TOE Security Functions**: Windows provides a number of features to ensure the protection of TOE security functions. Windows protects against unauthorized data disclosure and modification by using a suite of Internet standard protocols including IPsec, IKE, and ISAKMP. Windows ensures process isolation security for all processes through private virtual address spaces, execution context, and security context. The Windows data structures defining process address space, execution context, memory protection, and security context are stored in protected kernel-mode memory. Windows includes self-testing features that ensure the integrity of executable program images and its cryptographic functions. Finally, Windows provides a trusted update mechanism to update Windows binaries itself.
- **Session Locking:** Windows provides the ability for a user to lock their session either immediately or after a defined interval. Windows constantly monitors the mouse, keyboard, and touch display for activity and locks the computer after a set period of inactivity. Windows allows an authorized administrator to configure the system to display a logon banner before the logon dialog.
- **Trusted Path for Communications**: Windows uses the IPsec suite of protocols to provide a Virtual Private Network Connection (VPN) between itself, acting as a VPN client, and a VPN gateway in addition to providing protected communications for HTTPS and TLS.
- **Security Management:** Windows includes several functions to manage security policies. Policy management is controlled through a combination of access control, membership in administrator groups, and privileges.

[5] These options were not included in the Windows Mobile Device Common Criteria evaluation.

3 Security Problem Definition

The security problem definition consists of the threats to security, organizational security policies, and usage assumptions as they relate to Windows 8.1, and Windows Phone. The assumptions, threats, and policies are copied from the Protection Profile for Mobile Device Fundamentals ("MDF PP").

3.1 Threats to Security

Table 3-1 presents known or presumed threats to protected resources that are addressed by Windows 8.1 and Windows Phone based on conformance to the protection profile.

Table 3-1 MDF PP Threats Addressed by Windows 8.1 and Windows Phone

Threat	Description
T.EAVESDROP	**Network Eavesdropping**: An attacker is positioned on a wireless communications channel or elsewhere on the network infrastructure. Attackers may monitor and gain access to data exchanged between the Mobile Device and other endpoints.
T.NETWORK	**Network Attack**: An attacker is positioned on a wireless communications channel or elsewhere on the network infrastructure. Attackers may initiate communications with the Mobile Device or alter communications between the Mobile Device and other endpoints in order to compromise the Mobile Device. These attacks include malicious software update of any applications or system software on the device. These attacks also include malicious web pages or email attachments which are usually delivered to devices over the network.
T.PHYSICAL	**Physical Access**: The loss or theft of the Mobile Device may give rise to loss of confidentiality of user data including credentials. These physical access threats may involve attacks which attempt to access the device through external hardware ports, through its user interface, and also through direct and possibly destructive access to its storage media. The goal of such attacks is to access data from a lost or stolen device which is not expected to return to its user. **Note:** Defending against device re-use after physical compromise is out of scope for this security target.
T.FLAWAPP	**Malicious or Flawed Application**: Applications loaded onto the Mobile Device may include malicious or exploitable code. This code could be included intentionally by its developer or unknowingly by the developer, perhaps as part of a software library. Malicious apps may attempt to exfiltrate data to which they have access. They may also conduct attacks against the platform's system software which will provide them with additional privileges and the ability to conduct further malicious activities. Malicious applications may be able to control the device's sensors (GPS, camera, microphone) to

	gather intelligence about the user's surroundings even when those activities do not involve data resident or transmitted from the device. Flawed applications may give an attacker access to perform network-based or physical attacks that otherwise would have been prevented.
T.PERSISTENT	**Persistent Access**: Persistent access to 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.

3.2 Organizational Security Policies

An organizational security policy is a set of rules or procedures imposed by an organization upon its operations to protect its sensitive data and IT assets. **Table 3-2** describes organizational security policies that are addressed by Windows 8.1, and Windows Phone which are necessary for conformance to the MDF PP.

Table 3-2 Organizational Security Policies

Security Policy	Description
[None]	There are no Organizational Security Policies for the Mobile Device protection profile.

3.3 Secure Usage Assumptions

Table 3-3 describes the core security aspects of the environment in which Windows 8.1 and Windows Phone is intended to be used. It includes information about the physical, personnel, procedural, and connectivity aspects of the environment.

The following specific conditions are assumed to exist in an environment where the TOE is employed in order to conform to the MDF PP:

Table 3-3 Secure Usage Assumptions

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

4 Security Objectives

This section defines the security objectives of Windows 8.1 and Windows Phone and its supporting environment. Security objectives, categorized as either TOE security objectives or objectives by the supporting environment, reflect the stated intent to counter identified threats, comply with any organizational security policies identified, or address identified assumptions. All of the identified threats, organizational policies, and assumptions are addressed under one of the categories below.

4.1 TOE Security Objectives

Table 4-1 describes the security objectives for Windows 8.1 and Windows Phone which are needed to comply with the MDF PP.

Table 4-1 Security Objectives for the TOE

Security Objective	Source
O.COMMS	The TOE will provide the capability to communicate using one (or more) standard protocols as a means to maintain the confidentiality of data that are transmitted outside of the TOE.
O.STORAGE	The TOE will provide the capability to encrypt all user and enterprise data and authentication keys to ensure the confidentiality of data that it stores.
O.CONFIG	The TOE will provide the capability to configure and apply security policies. This ensures the Mobile Device can protect user and enterprise data that it may store or process.
O.AUTH	The TOE will provide the capability to authenticate the user and endpoints of a trusted path to ensure they are communicating with an authorized entity with appropriate privileges.
O.INTEGRITY	The TOE will provide the capability to perform self-tests to ensure the integrity of critical functionality, software/firmware and data has been maintained. The TOE will also provide a means to verify the integrity of downloaded updates.

4.2 Security Objectives for the Operational Environment

The TOE is assumed to be complete and self-contained and, as such, is not dependent upon any other products to perform properly. However, certain objectives with respect to the general operating environment must be met. **Table 4-2** describes the security objectives for the operational environment as specified in the MDF PP.

Table 4-2 Security Objectives for the Operational Environment

Environment Objective	Description
OE.CONFIG	TOE administrators will configure the Mobile Device security functions correctly to create the intended security policy.
OE.NOTIFY	The Mobile User will immediately notify the administrator if the

	Mobile Device is lost or stolen.
OE.PRECAUTION	The Mobile User exercises precautions to reduce the risk of loss or theft of the Mobile Device.

5 Security Requirements

The section defines the Security Functional Requirements (SFRs) and Security Assurance Requirements (SARs) for the TOE. The requirements in this section have been drawn from the Protection Profile for Mobile Device Fundamentals, Version 1.1, February 12, 2014, the Common Criteria.

Conventions:

Where requirements are drawn from the MDF PP, the requirements are copied verbatim, except for some changes to required identifiers to match the iteration convention of this document, from that protection profile and only operations performed in this security target are identified.

The extended requirements, extended component definitions and extended requirement conventions in this security target are drawn from the protection profile; the security target reuses the conventions from the protection profile which include the use of the word “Extended” and the “_EXT” identifier to denote extended functional requirements. The security target assumes that the protection profile correctly defines the extended components and so they are not reproduced in the security target.

Where applicable the following conventions are used to identify operations:

- **Iteration**: Iterated requirements (components and elements) are identified with letter following the base component identifier. For example, iterations of FMT_MOF.1 are identified in a manner similar to FMT_MOF.1(Audit) (for the component) and FCS_COP.1.1(Audit) (for the elements).
- **Assignment**: Assignments are identified in brackets and bold (e.g., **[assigned value]**).
- **Selection**: Selections are identified in brackets, bold, and italics (e.g., ***[selected value]***).
    - Assignments within selections are identified using the previous conventions, except that the assigned value would also be italicized and extra brackets would occur (e.g., ***[selected value [assigned value]]***).
- **Refinement**: Refinements are identified using bold text (e.g., **added text**) for additions and strike-through text (e.g., ~~deleted text~~) for deletions.

5.1 TOE Security Functional Requirements

This section specifies the SFRs for the TOE.

Table 5-1 TOE Security Functional Requirements

Requirement Class	Requirement Component
Cryptographic Support (FCS)	Cryptographic Key Generation for Key Establishment (FCS_CKM.1(ASYM KA))
	Cryptographic Key Generation for Authentication (FCS_CKM.1(ASYM AU))
	Cryptographic Key Generation for WLAN (FCS_CKM.1(WLAN))
	Cryptographic Key Distribution for WLAN (FCS_CKM.2)
	Extended: Cryptographic Key Support for Root Encryption Key (FCS_CKM_EXT.1)
	Extended: Cryptographic Key Random Generation for Data Encryption Keys (FCS_CKM_EXT.2(128))

	Extended: Cryptographic Key Random Generation for Data Encryption Keys (FCS_CKM_EXT.2(256))
	Extended: Cryptographic Key Generation for Key Encryption Keys (FCS_CKM_EXT.3)
	Extended: Cryptographic Key Destruction (FCS_CKM_EXT.4)
	Extended: TSF Wipe (FCS_CKM_EXT.5)
	Extended: Cryptographic Salt Generation (FCS_CKM_EXT.6)
	Cryptographic Operation for Data Encryption/Decryption (FCS_COP.1(SYM))
	Cryptographic Operation for Hashing (FCS_COP.1(HASH))
	Cryptographic Operation for Signature Algorithms (FCS_COP.1(SIGN))
	Cryptographic Operation for Keyed Hash Algorithms (FCS_COP.1(HMAC))
	Cryptographic Operation for Password Based Key Derivation (FCS_COP.1(PBKD))
	Extended: Initialization Vector Generation (FCS_IV_EXT.1)
	Extended: Random Bit Generation (FCS_RBG_EXT.1)
	Extended: Cryptographic Algorithm Services (FCS_SRV_EXT.1)
	Extended: Cryptographic Key Storage (FCS_STG_EXT.1)
	Extended: Encrypted Cryptographic Key Storage (FCS_STG_EXT.2)
	Extended: Integrity of Encrypted Key Storage (FCS_STG_EXT.3)
	Extended: EAP TLS Protocol (FCS_TLS_EXT.1)
	Extended: TLS Protocol (FCS_TLS_EXT.2)
	Extended: HTTPS Protocol (FCS_HTTPS_EXT.1)
User Data Protection (FDP)	Extended: Security Attribute Based Access Control (FDP_ACF_EXT.1)
	Extended: Data at Rest Protection (FDP_DAR_EXT.1(128))
	Extended: Data at Rest Protection (FDP_DAR_EXT.1(256))
	Extended: Sensitive Data Encryption (FDP_DAR_EXT.2))
	Extended: Certificate Data Storage (FDP_STG_EXT.1)
Identification & Authentication (FIA)	Extended: Authorization Failure Handling (FIA_AFL_EXT.1)
	Extended: Bluetooth Authentication (FIA_BLT_EXT.1)
	Extended: PAE Authentication (FIA_PAE_EXT.1)
	Extended: Password Management (FIA_PMG_EXT.1)
	Extended: Authorization Throttling (FIA_TRT_EXT.1)
	Protected Authorization Feedback (FIA_UAU.7)
	Extended: Authentication for Cryptographic Operation (FIA_UAU_EXT.1)
	Extended: Timing of Authentication (FIA_UAU_EXT.2)
	Extended: Re-Authorizing (FIA_UAU_EXT.3)
	Extended: Validation of Certificates (FIA_X509_EXT.1)
	Extended: X.509 Certificate Authentication (FIA_X509_EXT.2)
	Extended: Request Validation of Certificates (FIA_X509_EXT.3)
Security Management (FMT)	Management of Security Functions Behavior by the User (FMT_MOF.1(USER))
	Management of Security Functions Behavior by the Organization (FMT_MOF.1(ORG))
	Specifications of Management Functions (FMT_SMF.1)
	Extended: Specification of Remediation Actions (FMT_SMF_EXT.1)
Protection of the TSF (FPT)	Extended: Anti-Exploitation Services for Address Space Layout Randomization (FPT_AEX_EXT.1)

	Extended: Anti-Exploitation Services for Memory Page Permissions (FPT_AEX_EXT.2)
	Extended: Anti-Exploitation Services for Stack Overflow Protection (FPT_AEX_EXT.3)
	Extended: Domain Isolation (FPT_AEX_EXT.4)
	Extended: Plaintext Key Storage (FPT_KST_EXT.1)
	Extended: No Key Transmission (FPT_KST_EXT.2)
	Extended: No Plaintext Key Transport (FPT_KST_EXT.3)
	Extended: Self-Text Event Notification (FPT_NOT_EXT.1)
	Reliable Time Stamps (FPT_STM.1)
	Extended: TSF Cryptographic Functionality Testing (FPT_TST_EXT.1)
	Extended: TSF Integrity Testing (FPT_TST_EXT.2)
	Extended: Trusted Update: TSF Version Query (FPT_TUD_EXT.1)
	Extended: Trusted Update Verification (FPT_TUD_EXT.2)
TOE Access (FTA)	Extended: TSF- and User-initiated Locked State (FTA_SSL_EXT.1)
	Extended: Wireless Network Access (FTA_WSE_EXT.1)
	Default TOE Access Banners (FTA_TAB.1)
Trusted Path/Channels (FTP)	Extended: Trusted Channel Communication (FTP_ITC_EXT.1)

5.1.1 Cryptographic Support (FCS)

The functional requirements described in this are only those portions of the cryptographic functions implemented within Windows which are needed to meet the requirements of the Mobile Device Fundamentals protection profile. The intent is to describe only a subset of the product rather than a comprehensive review of Windows.

5.1.1.1 Cryptographic Key Generation for Key Establishment (FCS_CKM.1(ASYM KA))

Application Note: FCS_CKM.1(ASYM KA) corresponds to FCS_CKM.1(1) in the MDF protection profile.

FCS_CKM.1.1(ASYM KA) The TSF shall generate asymmetric cryptographic keys used for key establishment in accordance with:

- NIST Special Publication 800-56B, ”Recommendation for Pair-Wise Key Establishment Schemes Using Integer Factorization Cryptography” for RSA-based key establishment schemes and

[

- ***NIST Special Publication 800-56A, “Recommendation for Pair-Wise Key Establishment Schemes Using Discrete Logarithm Cryptography” for finite field- based key establishment schemes;***
- ***NIST Special Publication 800-56A, “Recommendation for Pair-Wise Key Establishment Schemes Using Discrete Logarithm Cryptography” for elliptic curve- based key establishment schemes and implementing “NIST curves” P-256, P-384 and [P-521] (as defined in FIPS PUB 186-4, “Digital Signature Standard”)***

]

and specified cryptographic key sizes equivalent to, or greater than, a symmetric key strength of 112 bits.

5.1.1.2 Cryptographic Key Generation for Authentication (FCS_CKM.1(ASYM AU))

Application Note: FCS_CKM.1(ASYM AU) corresponds to FCS_CKM.1(2) in the MDF protection profile.

FCS_CKM.1.1(ASYM AU) The TSF shall generate asymmetric cryptographic keys used for authentication in accordance with a specified cryptographic key generation algorithm

[

- ***FIPS PUB 186-4, "Digital Signature Standard (DSS)", Appendix B.3 for RSA schemes;***
- ***FIPS PUB 186-4, "Digital Signature Standard (DSS)", Appendix B.4 for ECDSA schemes and implementing "NIST curves" P-256, P-384 and [P-521];***

]

and specified cryptographic key sizes [equivalent to, or greater than, a symmetric key strength of 112 bits].

5.1.1.3 Cryptographic Key Generation for WLAN (FCS_CKM.1(WLAN))

Application Note: FCS_CKM.1(WLAN) corresponds to FCS_CKM.1(3) in the MDF protection profile.

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

5.1.1.4 Cryptographic Key Distribution for WLAN (FCS_CKM.2)

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

5.1.1.5 Extended: Cryptographic Key Support for Root Encryption Key (FCS_CKM_EXT.1)

FCS_CKM_EXT.1.1 The TSF shall support a [***hardware-isolated***] REK with an [***symmetric***] key of size [***256 bits***].

FCS_CKM_EXT.1. System software on the TSF shall be able only to request [***NIST SP 800-108 key derivation***] by the key and shall not be able to read, import, or export a REK.

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

5.1.1.6 Extended: Cryptographic Key Random Generation for Data Encryption Keys (FCS_CKM_EXT.2(128))[6]

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

[6] This iteration of FCS_CKM_EXT.2 is for Windows Phone which always uses a 128 bit DEK.

5.1.1.7 Extended: Cryptographic Key Random Generation for Data Encryption Keys (FCS_CKM_EXT.2(256))[7]

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

5.1.1.8 Extended: Cryptographic Key Generation for Key Encryption Keys (FCS_CKM_EXT.3)

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

FCS_CKM_EXT.3.2 The TSF shall generate all KEKs using one 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) generate the KEK using a key generation scheme that meets this profile (as specified in FCS_CKM.1(1))[8]

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

].

5.1.1.9 Extended: Cryptographic Key Destruction (FCS_CKM_EXT.4)

FCS_CKM_EXT.4.1 The TSF shall destroy cryptographic keys in accordance with the specified cryptographic key destruction method

[

- ***by clearing the KEK encrypting the target key,***
- ***in accordance with the following rules:***
  - ***For volatile EEPROM the destruction shall be executed by a single direct overwrite consisting of a pseudo random pattern using the TSF's RBG (as specified in FCS_RBG_EXT.1), followed a read-verify.***
  - ***For volatile flash memory the destruction shall be executed by [a single direct overwrite consisting of zeros followed by a read-verify, a block erase followed by a read-verify.]***

].

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

5.1.1.10 Extended: TSF Wipe (FCS_CKM_EXT.5)

FCS_CKM_EXT.5.1 The TSF shall wipe all protected data by

[

- ***Cryptographically erasing the encrypted DEKs and/or the KEKs in non-volatile memory by following the requirements in***

[7] This iteration of FCS_CKM_EXT.2 is for Windows 8.1 which can use either a 128 bit DEK or a 256-bit DEK, the administrative guidance restricts the DEK in the evaluated configuration to 256 bits.

[8] FCS_CKM.1(1) in the protection profile corresponds to FCS_CKM.1(ASYM KA).

FCS_CKM_EXT.4.1;

]

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

5.1.1.11 Extended: Cryptographic Salt Generation (FCS_CKM_EXT.6)

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

5.1.1.12 Cryptographic Operation for Data Encryption/Decryption (FCS_COP.1(SYM))

Application Note: FCS_COP.1(SYM) corresponds to FCS_COP.1(1) in the MDF protection profile.

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

- AES-CBC (as defined in 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 Key Wrap with Padding (KWP) (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***].

5.1.1.13 Cryptographic Operation for Hashing (FCS_COP.1(HASH))

Application Note: FCS_COP.1(HASH) corresponds to FCS_COP.1(2) in the MDF protection profile.

FCS_COP.1.1(HASH) The TSF shall perform [cryptographic hashing] in accordance with a specified cryptographic algorithm SHA-1 and [***SHA-256, SHA-384, SHA-512***] and message digest sizes 160 and [***256, 384, 512 bits***] that meet the following: [FIPS Pub 180-4].

5.1.1.14 Cryptographic Operation for Signature Algorithms (FCS_COP.1(SIGN))

Application Note: FCS_COP.1(SIGN) corresponds to FCS_COP.1(3) in the MDF protection profile.

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

- FIPS PUB 186-4, “Digital Signature Standard (DSS)”,Section 4 for RSA schemes

  [
- ***FIPS PUB 186-4, “Digital Signature Standard (DSS)”, Section 5 for ECDSA schemes and implementing “NIST curves” P-256, P-384 and [P-521]***

  ]

  and cryptographic key sizes [equivalent to, or greater than, a symmetric key strength of 112 bits].

5.1.1.15 Cryptographic Operation for Keyed Hash Algorithms (FCS_COP.1(HMAC))

Application Note: FCS_COP.1(HMAC) corresponds to FCS_COP.1(4) in the MDF protection profile.

FCS_COP.1.1(HMAC) 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, 256***] and message digest sizes 160 and [***256, 384, 512***] bits that meet the following: [FIPS Pub 198-1, "The Keyed-Hash Message Authentication Code, and FIPS Pub 180-4, "Secure Hash Standard].

5.1.1.16 Cryptographic Operation for Password Based Key Derivation (FCS_COP.1(PBKD))

Application Note: FCS_COP.1(PBKD) corresponds to FCS_COP.1(5) in the MDF protection profile.

FCS_COP.1.1(PBKD) The TSF shall perform [Password-based Key Derivation Functions] in accordance with a specified cryptographic algorithm [HMAC-[***SHA-1, SHA-256, SHA-384, SHA-512***]] and output cryptographic key sizes [***128, 256***] that meet the following: [NIST SP 800-132].

5.1.1.17 Extended: Initialization Vector Generation (FCS_IV_EXT.1)

FCS_IV_EXT.1.1 The TSF shall generate IVs in accordance with Table 11: References and IV Requirements for NIST-approved Cipher Modes. [9]

5.1.1.18 Extended: Random Bit Generation (FCS_RBG_EXT.1)

FCS_RBG_EXT.1.1 The TSF shall perform all deterministic random bit generation services in accordance with [***NIST Special Publication 800-90A using [CTR_DRBG (AES), Dual_EC_DRBG (any)]***].

FCS_RBG_EXT.1.2 The deterministic RBG shall be seeded by an entropy source that accumulates entropy from [***a software-based noise source, TSF-hardware-based noise source***] with a minimum of [***256 bits***] of entropy at least equal to the greatest security strength (according to NIST SP 800-57) of the keys and hashes that it will generate.

FCS_RBG_EXT.1.3 The TSF shall be capable of providing output of the RBG to applications running on the TSF that request random bits.

5.1.1.19 Extended: Cryptographic Algorithm Services (FCS_SRV_EXT.1)

FCS_SRV_EXT.1.1 The TSF shall provide a mechanism for [**Windows Store** applications] to request the TSF to perform the following cryptographic operations:

- FCS_COP.1(**SYM** ~~1~~)
- FCS_COP.1(**SIGN** ~~3~~)
- FCS_COP.1(**HASH** ~~2~~)
- FCS_COP.1(**HMAC** ~~4~~)
- FCS_COP.1(**PBKD** ~~5~~)
- FCS_CKM.1(**ASYM KA** ~~1~~)
- [***FCS_CKM.1(ASYM AU ~~2~~)***].

5.1.1.20 Extended: Cryptographic Key Storage (FCS_STG_EXT.1)

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

FCS_STG_EXT.1.2 The TSF shall be capable of importing keys/secrets into the secure key storage

[9] This refers to Table 11 of the MDF PP.

upon request of [***the user, the administrator***] and [***applications running on the TSF***].

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

FCS_STG_EXT.1.4 The TSF shall have the capability to allow only the application that imported the key/secret the use of the key/secret. Exceptions may only be explicitly authorized by [***the user, the administrator***].

FCS_STG_EXT.1.5 The TSF shall allow only the application that imported the key/secret to request that the key/secret be destroyed. Exceptions may only be explicitly authorized by [***the user, the administrator***].

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

FCS_STG_EXT.2.1 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 REK,***
    b. ***encryption by a KEK chaining to a REK],***
2) ***Protected by the REK and the password with [***
    a. ***encryption by a REK and the password-derived KEK***
    b. ***encryption by a KEK chaining to a REK and the password-derived KEK***]

].

FCS_STG_EXT.2.2 DEKs and KEKs and [***no other keys***] shall be encrypted using one of the following methods: [***SP800-56B key establishment scheme, using AES in the [CCM, CBC mode]***].

5.1.1.22 Extended: Integrity of Encrypted Key Storage (FCS_STG_EXT.3)

FCS_STG_EXT.3.1 The TSF shall protect the integrity of any encrypted KEK by [

- ***[CCM] cipher mode for encryption according to FCS_STG_EXT.2;***
- ***a keyed hash (FCS_COP.1(4)) using a key protected by a key protected by FCS_STG_EXT.2;*** [10]

].

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

5.1.1.23 Extended: EAP TLS Protocol (FCS_TLS_EXT.1)

FCS_TLS_EXT.1.1 The TSF shall implement the EAP-TLS protocol as specified in RFC 5216 implementing TLS 1.0 (RFC 2246) and [***none***] supporting the following ciphersuites: [

- Mandatory Ciphersuites in accordance with RFC 3268:
    - TLS_RSA_WITH_AES_128_CBC_SHA

[

- ***TLS_RSA_WITH_AES_256_CBC_SHA***

]

[10] FCS_COP.1(4) in the protection profile is FCS_COP.1(HMAC) in the security target.

]

FCS_TLS_EXT.1.2 The TSF shall verify that the server certificate presented for EAP-TLS [***chains to one of the specified CAs, contains the specified FQDN of the acceptable authentication server certificate***].

5.1.1.24 Extended: TLS Protocol (FCS_TLS_EXT.2)

FCS_TLS_EXT.2.1 The TSF shall implement one or more of the following protocols TLS 1.2 (RFC 5246) and [***TLS 1.0 (RFC 2246), TLS 1.1 (RFC 4346)***] supporting the following ciphersuites:

- Mandatory Ciphersuites:
  - TLS_RSA_WITH_AES_128_CBC_SHA

[

- ***TLS_RSA_WITH_AES_256_CBC_SHA***
- ***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_ECDHE_ECDSA_WITH_AES_128_GCM_SHA256 as defined in RFC 5289***
- ***TLS_ECDHE_ECDSA_WITH_AES_256_GCM_SHA384 as defined in RFC 5289***
- ***TLS_ECDHE_ECDSA_WITH_AES_128_CBC_SHA256 as defined in RFC 6460***
- ***TLS_ECDHE_ECDSA_WITH_AES_256_CBC_SHA384 as defined in RFC 6460***]

]

FCS_TLS_EXT.2.2 The TSF shall not establish a trusted channel if the distinguished name (DN) contained in a certificate does not match the expected DN for the peer.

5.1.1.25 Extended: HTTPS Protocol (FCS_HTTPS_EXT.1)

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

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

5.1.2 User Data Protection (FDP)

5.1.2.1 Extended: Security Attribute Based Access Control (FDP_ACF_EXT.1)

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

5.1.2.2 Extended: Data at Rest Protection (FDP_DAR_EXT.1(128))[11]

FDP_DAR_EXT.1.1(128) Encryption shall cover all protected data.

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

[11] This iteration of FDP_DAR_EXT.1 is for Windows Phone which always uses a 128 bit DEK.

5.1.2.3 Extended: Data at Rest Protection (FDP_DAR_EXT.1(256))[12]

FDP_DAR_EXT.1.1(256) Encryption shall cover all protected data.

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

5.1.2.4 Extended: Sensitive Data Encryption (FDP_DAR_EXT.2)

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

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

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

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

5.1.2.5 Extended: Certificate Data Storage (FDP_STG_EXT.1)

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

5.1.3 Identification and Authentication (FIA)

5.1.3.1 Extended: Authorization Failure Handling (FIA_AFL_EXT.1)

FIA_AFL_EXT.1.1 The TSF shall detect when [a configurable positive integer within [**a range of acceptable values from 1 to 999**]] of unsuccessful authentication attempts occur related to [last successful authentication by that user].[13]

FIA_AFL_EXT.1.2 When the defined number of unsuccessful authentication attempts has been [**surpassed**], the TSF shall [perform [***full wipe of all protected data, a remediation action set by the administrator***].[14]

5.1.3.2 Extended: Bluetooth Authentication (FIA_BLT_EXT.1)

FIA_BLT_EXT.1.1 The TSF shall require Bluetooth mutual authentication between devices prior to any data transfer over the Bluetooth link.

5.1.3.3 Extended: PAE Authentication (FIA_PAE_EXT.1)

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

5.1.3.4 Extended: Password Management (FIA_PMG_EXT.1)

FIA_PMG_EXT.1.1 The TSF shall support the following for the Password Authentication Factor:

1. Passwords shall be able to be composed of any combination of [**upper and lower case characters**], number, and special characters: [“!”, “@”, “#”,

[12] This iteration of FDP_DAR_EXT.1 is for Windows 8.1 which can use either a 128 bit DEK or a 256-bit DEK, the administrative guidance restricts the DEK in the evaluated configuration to 256 bits.

[13] Note that a lockout value of 0 denotes the account will never be locked out.

[14] The Windows Phone will wipe protected data, the typical remediation action for Windows 8.1 is to lock out the user account.

"$", "%", "^", "&", "*", "(", ")"];

2. Password length up to [**at least 15**] characters shall be supported.

5.1.3.5 Extended: Authorization Throttling (FIA_TRT_EXT.1)

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

5.1.3.6 Protected Authorization Feedback (FIA_UAU.7)

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

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

FIA_UAU_EXT.1.1 The TSF shall require the user to present the Password Authentication Factor prior to decryption of protected data and keys at startup.

5.1.3.8 Extended: Timing of Authentication (FIA_UAU_EXT.2)

FIA_UAU_EXT.2.1 The TSF shall allow [***no actions for Windows 8.1 and emergency call for Windows Phone***] on behalf of the user to be performed before the user is authenticated. [15]

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

5.1.3.9 Extended: Re-Authorizing (FIA_UAU_EXT.3)

FIA_UAU_EXT.3.1 The TSF shall require the user to enter the correct Password Authentication Factor when the user changes the Password Authentication Factor, and following TSF- and user-initiated locking in order to transition to the unlocked state, and [***no other conditions***].

5.1.3.10 Extended: Validation of Certificates (FIA_X509_EXT.1)

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

- RFC 5280 certificate validation and certificate path validation.
- The certificate path must terminate with a certificate in the Trust Anchor Database.
- The TSF shall validate a certificate path by ensuring the presence of the basicConstraints extension and that the cA flag is set to TRUE for all CA certificates.
- The TSF shall validate the revocation status of the certificate using [***the Online Certificate Status Protocol (OCSP) as specified in RFC 2560, a Certificate Revocation List (CRL) as specified in RFC 5759***].
- The TSF shall validate the extendedKeyUsage field according to the following rules:
  - Certificates used for trusted updates and executable code integrity verification shall have the Code Signing purpose (id-

[15] The only actions that an unauthenticated user can take when a Windows device is locked is to bring up the authentication dialog or turn the device off. An unauthenticated user may place an emergency call for the Windows Phone or turn the device off.

kp 3 with OID 1.3.6.1.5.5.7.3.3).
- 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.

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

5.1.3.11 Extended: X.509 Certificate Authentication (FIA_X509_EXT.2)

FIA_X509_EXT.2.1 The TSF shall use X.509v3 certificates as defined by RFC 5280 to support authentication for EAP-TLS exchanges, and [***IPsec, TLS, HTTPS,***], and [***code signing for system software updates, code signing for mobile applications, code signing for integrity verification,*** [***none***]].

FIA_X509_EXT.2.2 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, not accept the certificate**[16]].

FIA_X509_EXT.2.3 The TSF shall not establish a trusted communication channel if the peer certificate is deemed invalid.

FIA_X509_EXT.2.4 The TSF shall not [***install, execute***] code if the code signing certificate is deemed invalid.

FIA_X509_EXT.2.5 The TSF shall generate a Certificate Request Message as specified in RFC 2986 and be able to provide the following information in the request: public key, Common Name, Organization, Organizational Unit, and Country.

5.1.3.12 Extended: Request Validation of Certificates (FIA_X509_EXT.3)

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

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

5.1.4 Security Management (FMT)

5.1.4.1 Management of Security Functions Behavior by the User (FMT_MOF.1(USER))

Application Note: FMT_MOF.1(USER) corresponds to FMT_MOF.1(1) in the MDF protection profile.

This functional requirement includes the full set of selections from the protection profile for readability, selections which are not used are marked with a strikethrough font.

FMT_MOF.1.1(USER) The TSF shall restrict the ability to [perform] the functions [

1. enroll the TOE in management
[
2. ***enable/disable the VPN protection,***
3. ***enable/disable [Wi-Fi, cellular radios],***
4. ***enable/disable data transfer capabilities over [USB port for Windows 8.1 , Bluetooth],***
5. ***enable/disable [personal Hotspot connections, tethered connections]***, [17]

[16] Windows will not accept the certificate for validation failures for IPsec, software updates, mobile applications, and integrity verification. Windows will present the user with an option to accept the certificate for TLS/HTTPS.

6. ***enable/disable display notification in the locked state of: [***
    a. ***email notifications,***
    b. ***calendar appointments,***
    c. ***contact associated with phone call notification,***
    d. ***text message notification,***
    e. ***other application-based notification]***
7. ~~enable/disable developer modes,~~
8. ***~~enable data-at rest protection~~***
9. ***enable removable media's data-at-rest protection,***[18]
10. ~~enable/disable local authentication bypass,~~
11. ***configure the Access Point Name and proxy used for communications between the cellular network and other networks***
12. ~~configure the Bluetooth trusted channel~~
    a. ~~disable the Discoverable mode~~
    b. ~~disallow Bluetooth connections using versions 1.0, 1.1, 1.2, 2.0, and [assignment: other Bluetooth version numbers]~~
    c. ~~[selection: restrict Bluetooth profiles, disable legacy pairing and JustWorks pairing, and [selection: [assignment: other pairing methods], no other pairing methods]]~~,
13. ~~wipe sensitive data~~
14. ***import keys/secrets into the secure key storage,***
15. ***destroy user-imported keys/secrets and*** [***[any other keys/secrets]***] in the secure key storage,
16. ***~~remove imported X.509v3 certificates and~~*** [***~~[any other X.509v3 certificate]~~***] ***~~in the Trust Anchor Database,~~***
17. ***approve import and removal by applications of X.509v3 certificates in the Trust Anchor Database,***
18. ~~configure whether to establish a trusted channel or disallow establishment if the TSF cannot establish a connection to determine the validity of a certificate,~~
19. ***enable/disable cellular voice functionality,***
20. ***enable/disable device messaging capabilities,***
21. ***enable/disable the cellular protocols used to connect to cellular network base stations,***
22. ~~enable/disable voice command control of device functions,~~
23. ~~read audit logs kept by the TSF,~~
24. ***~~configure [certificate, public key] used to validate digital signature on applications,~~***
25. ***~~approve exceptions for shared use of keys/secrets by multiple applications~~***
26. ***~~approve exceptions for destruction of keys/secrets by applications that did not import the key/secret~~***
27. [***no other management functions***]

]] to the user.

[17] For Windows 8.1 only, the Lumia phone does not provide tethered connections.
[18] For Windows 8.1 only.

5.1.4.2 Management of Security Functions Behavior by the Organization (FMT_MOF.1(ORG))

Application Note: FMT_MOF.1(ORG) corresponds to FMT_MOF.1(2) in the MDF protection profile.

This functional requirement includes the full set of selections from the protection profile for readability, selections which are not used are marked with a strikethrough font.

FMT_MOF.1.1(ORG) The TSF shall restrict the ability to perform the functions [

1. configure password policy:
    a. minimum password length
    b. minimum password complexity
    c. maximum password lifetime
2. configure session locking policy:
    a. screen-lock enabled/disabled
    b. screen lock timeout
    c. number of authentication failures
3. enable/disable [***camera, microphone***]
4. configure application installation policy by [
    a. specifying authorized application repository(s),
    ~~b. specifying a set of allowed applications and versions (an application whitelist)~~
    c. ***denying installation of applications***],

[

~~***5. enable/disable the VPN protection***~~

~~***6. enable/disable [Wi-Fi, mobile broadband radios, Bluetooth]***~~

~~7. ***enable/disable data transfer capabilities over*** [***USB port for Windows 8.1, Bluetooth***],~~

8. enable/disable [wireless remote access connections except for personal Hotspot service, personal Hotspot connections, tethered connections], [19]

9. specify wireless networks (SSIDs) to which the TSF may connect

10. configure security policy for each wireless network:
    a. [specify the CA(s) from which the TSF will accept WLAN authentication server certificate(s), specify the FQDN(s) of acceptable WLAN authentication server certificate(s)]
    b. ability to specify security type
    c. ability to specify authentication protocol
    d. specify the client credentials to be used for authentication
    ~~e. [none]~~

~~11. enable/disable developer modes,~~

12. enable data-at rest protection,

~~13. ***enable removable media's data-at rest protection,***~~

~~14. enable/disable local authentication bypass,~~

~~15. configure the Access Point Name and proxy used for communications between the cellular network and other networks~~

~~16. configure the Bluetooth trusted channel~~
    ~~a. disable the Discoverable mode~~

[19] For Windows 8.1 only, the Lumia phone does not provide tethered connections.

~~b. disallow Bluetooth connections using versions 1.0, 1.1, 1.2, 2.0, and [assignment: other Bluetooth version numbers]~~
~~c. [selection: restrict Bluetooth profiles, disable legacy pairing and JustWorks pairing, and [selection: [assignment: other pairing methods], no other pairing methods]],~~
~~17. enable/disable display notification in the locked state of: [~~
~~a. email notifications,~~
~~b. calendar appointments,~~
~~c. contact associated with phone call notification,~~
~~d. text message notification,~~
~~e. other application-based notifications,~~
~~f. none]~~
18. import and remove X.509v3 certificates into/from the Trust Anchor Database,
~~19. configure whether to establish a trusted channel or disallow establishment if the TSF cannot establish a connection to determine the validity of a certificate,~~
~~20. approve import and removal by applications of X.509v3 certificates in the Trust Anchor Database,~~
~~21. enable/disable cellular voice functionality,~~
~~***22. enable/disable device messaging capabilities,***~~
~~23. enable/disable the cellular protocols used to connect to cellular network base stations,~~
~~24. enable/disable voice command control of device functions,~~
25. configure [certificate] used to validate digital signature on applications,
26. remove applications,
27. update system software,
28. install applications,
~~29. approve exceptions for shared use of keys/secrets by multiple applications~~
~~30. approve exceptions for destruction of keys/secrets by applications that did not import the key/secret~~
31. ***[none]***
]
to the administrator when the device is enrolled and according to the administrator- configured policy.

5.1.4.3 Specifications of Management Functions (FMT_SMF.1)

This functional requirement includes the full set of selections from the protection profile for readability, selections which are not used are marked with a strikethrough font.

FMT_SMF.1.1 The TSF shall be capable of performing the following management functions: [
1. configure password policy:
    a. minimum password length
    b. minimum password complexity
    c. maximum password lifetime
2. configure session locking policy:

a. screen-lock enabled/disabled
b. screen lock timeout
c. number of authentication failures

3. enable/disable the VPN protection
4. enable/disable [**Wi-Fi, mobile broadband radios, Bluetooth**]
5. enable/disable [**camera, microphone**]
6. specify wireless networks (SSIDs) to which the TSF may connect
7. configure security policy for each wireless network:
    a. [***specify the CA(s) from which the TSF will accept WLAN authentication server certificate(s), specify the FQDN(s) of acceptable WLAN authentication server certificate(s)***]
    b. ability to specify security type
    c. ability to specify authentication protocol
    d. specify the client credentials to be used for authentication
    e. [**none**]
8. transition to the locked state
9. full wipe of protected data
10. configure application installation policy by [
    a. specifying authorized application repository(s),
    ~~b. specifying a set of allowed applications and versions (an application whitelist)~~
    c. ***denying installation of applications***],
11. import keys/secrets into the secure key storage,
12. destroy imported keys/secrets and [***[any other keys/secrets]***] in the secure key storage,
13. import X.509v3 certificates into the Trust Anchor Database,
14. remove imported X.509v3 certificates and [***[any other X.509v3 certificates]***] in the Trust Anchor Database,
15. enroll the TOE in management
16. remove applications
17. update system software
18. install applications
    [
19. enable/disable data transfer capabilities over [USB port for Windows 8.1, Bluetooth],
20. enable/disable [***wireless remote access connections to the TOE except for personal Hotspot service, personal Hotspot connections, tethered connections***], [20]

~~21. enable/disable developer modes,~~ [21]

22. enable data-at rest protection, [22]

23. enable removable media's data-at-rest protection, [23]

~~24. enable/disable local authentication bypass,~~ [24]

[20] For Windows 8.1 only, the Lumia phone does not provide tethered connections.
[21] See footnote above in FMT_MOF.1(ORG).
[22] See footnote above in FMT_MOF.1(ORG).
[23] See footnote above in FMT_MOF.1(ORG).
[24] See footnote above in FMT_MOF.1(ORG).

25. configure the Access Point Name and proxy used for communications between the cellular network and other networks [25]

~~26. configure the Bluetooth trusted channel:~~

~~a. disable the Discoverable mode~~

~~b. disallow Bluetooth connections using versions 1.0, 1.1, 1.2, 2.0, and [assignment: other Bluetooth version numbers]~~

~~c. [selection: restrict Bluetooth profiles, disable legacy pairing and JustWorks pairing, and [selection: [assignment: other pairing methods], no other pairing methods]],~~

27. enable/disable display notification in the locked state of: [

a. email notifications,

b. calendar appointments,

c. contact associated with phone call notification,

d. text message notification,

e. other application-based notifications,

]

28. wipe sensitive data,

29. alert the administrator,

30. remove Enterprise applications,

31. approve import and removal by applications of X.509v3 certificates in the Trust Anchor Database,

32. configure whether to establish a trusted channel or disallow establishment if the TSF cannot establish a connection to determine the validity of a certificate,

33. enable/disable cellular voice functionality, [26]

34. enable/disable device messaging capabilities, [27]

~~35.~~ enable/disable the cellular protocols used to connect to cellular network base stations,~~-~~ [28]

~~36. enable/disable voice command control of device functions,~~

~~37. read audit logs kept by the TSF,~~

38. configure [certificate] used to validate digital signature on applications,

39. approve exceptions for shared use of keys/secrets by multiple applications,

40. approve exceptions for destruction of keys/secrets by applications that did not import the key/secret,

41. configure the unlock banner***,***

42. ***[enable/disable Location services]*** [29]

].

5.1.4.4 Extended: Specification of Remediation Actions (FMT_SMF_EXT.1)

FMT_SMF_EXT.1.1 The TSF shall offer [***alert the administrator, remove Enterprise applications***,***]*** upon unenrollment and [w***hen the defined number of unsuccessful***

[25] For the Lumia 635 and 830, the Surface 3 computer does not include a broadband modem.
[26] For the Lumia 635 and 830, the Surface 3 computer does not include a broadband modem.
[27] For the Lumia 635 and 830, the Surface 3 computer does not include a broadband modem.
[28] For the Lumia 635 and 830, the Surface 3 computer does not include a broadband modem.

authentication attempts has been surpassed wipe the device].

5.1.5 Protection of the TSF (FPT)

5.1.5.1 Extended: Anti-Exploitation Services for Address Space Layout Randomization (FPT_AEX_EXT.1)

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

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

FPT_AEX_EXT.1.3 The TSF shall provide [address space layout randomization (ASLR) to the kernel].

FPT_AEX_EXT.1.4 The base address of any kernel-space memory mapping will consist of at least 4 unpredictable bits.

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

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

FPT_AEX_EXT.2.2 The TSF shall be able to enforce a policy that write and execute permissions are not simultaneously granted on every page of physical memory.

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

FPT_AEX_EXT.3.1 TSF processes that execute in a non-privileged execution domain on the application processor shall implement stack-based buffer overflow protection.

5.1.5.4 Extended: Domain Isolation (FPT_AEX_EXT.4)

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

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

5.1.5.5 Extended: Plaintext Key Storage (FPT_KST_EXT.1)

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

5.1.5.6 Extended: No Key Transmission (FPT_KST_EXT.2)

FPT_KST_EXT.2.1 The TSF shall not transmit any plaintext key material from the cryptographic module.

5.1.5.7 Extended: No Plaintext Key Transport (FPT_KST_EXT.3)

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

5.1.5.8 Extended: Self-Test Event Notification (FPT_NOT_EXT.1)

FPT_NOT_EXT.1.1 The TSF shall transition to non-operational mode and [***log failures in the audit record,***[30] ***notify the administrator***] when the following types of failures occur:

[30] While the evaluation did not include auditing functional requirements, the failures in this requirement always generate audit events.

- failures of the self-tests
- TSF software integrity verification failures
- [***no other failures***].

5.1.5.9 Reliable Time Stamps (FPT_STM.1)

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

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

FPT_TST_EXT.1.1 The TSF shall run a suite of self-tests [during initial start-up (on power on)] to demonstrate the correct operation of [all cryptographic functionality].

5.1.5.11 Extended: TSF Integrity Testing (FPT_TST_EXT.2)

FPT_TST_EXT.2.1 The TSF shall verify the integrity of the Application Processor bootloader software, Application Processor OS kernel, and [***operating system executable code and application executable code***], stored in mutable media prior to its execution through the use of [***digital signature using a hardware-protected asymmetric key***].

5.1.5.12 Extended: Trusted Update: TSF Version Query (FPT_TUD_EXT.1)

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

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

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

5.1.5.13 Extended: Trusted Update Verification (FPT_TUD_EXT.2)

FPT_TUD_EXT.2.1 The TSF shall verify [software updates to the TSF] using [a digital signature by the manufacturer] prior to installing those updates.

FPT_TUD_EXT.2.2 The boot integrity [***key***] shall only be updated by [verified software].

FPT_TUD_EXT.2.3 The digital signature verification key shall [***be validated to a public key in the Trust Anchor Database, match a hardware-protected public key***].

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

FPT_TUD_EXT.2.5 The TSF shall ~~by default~~ only accept mobile applications cryptographically verified by [***a built-in X.509v3 certificate***].[31]

FPT_TUD_EXT.2.6 The TSF shall verify that software updates to the TSF are a current or later version than the current version of the TSF.

5.1.6 TOE Access (FTA)

5.1.6.1 Extended: TSF- and User-initiated Locked State (FTA_SSL_EXT.1)

FTA_SSL_EXT.1.1 The TSF shall transition to a locked state after a time interval of inactivity and a user initiated lock, and upon transitioning to the locked state, the TSF shall perform the following operations:

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

[31] All Windows Store Applications must signed by a Microsoft-approved certificate.

b) [**Disabling any activity of the user's data access / TSF controlled display devices other than unlocking the session and displaying application status**].

5.1.6.2 Extended: Wireless Network Access (FTA_WSE_EXT.1)

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

5.1.6.3 Default TOE Access Banners (FTA_TAB.1)

FTA_TAB.1.1 Before establishing a user session, the TSF shall display an Administrator-specified advisory notice and consent warning message regarding use of the TOE.

5.1.7 Trusted Path/Channels (FTP)

5.1.7.1 Extended: Trusted Channel Communication (FTP_ITC_EXT.1)

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

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

FTP_ITC_EXT.1.3 The TSF shall initiate communication via the trusted channel for connection to a wireless access point and [**remote management operations**].

5.2 TOE Security Assurance Requirements

The security assurance requirements for the TOE are the requirements defined in the MDF PP Assurance Package as specified in Part 3 of the Common Criteria. No operations are applied to the assurance components.

In addition, the assurance activities from the Protection Profile for Mobile Device Fundamentals are used to determine that Windows satisfies the mobile device security functional requirements. These assurance activities are described in section 5.2.2.

5.2.1 CC Part 3 Assurance Requirements

The following table is the collection of CC Part 3 assurance requirements from the Protection Profile for Mobile Device Fundamentals.

Table 5-2 TOE Security Assurance Requirements

Requirement Class	Requirement Component
ASE: Security Target	ASE_INT.1: ST introduction
	ASE_CCL.1: Conformance claims
	ASE_OBJ.1 Security objectives

	ASE_ECD.1 Extended components definition
	ASE_REQ.1 Stated security requirements
	ASE_TSS.1 TOE summary specification
ADV: Design	ADV_FSP.1: Basic functional specification
AGD: Guidance Documents	AGD_OPE.1: Operational user guidance
	AGD_PRE.1: Preparative procedures
ALC: Life-cycle Support	ALC_CMC.1: Labeling of the TOE
	ALC_CMS.1: TOE CM Coverage
	ALC_TSU_EXT.1: Timely Security Updates
ATE: Testing	ATE_IND.1: Independent testing - sample
AVA: Vulnerability Assessment	AVA_VAN.1: Vulnerability survey

5.2.1.1 Timely Security Updates (ALC_TSU_EXT.1)

Developer action elements:

ALC_TSU_EXT.1.1D The developer shall provide a description in the TSS of how timely security updates are made to the TOE.

Content and presentation elements:

ALC_TSU_EXT.1.1C The description shall include the process for creating and deploying security updates for the TOE software/firmware.

Application Note: The software to be described includes the operating systems of the application processor and the baseband processor, as well as any firmware and applications. The process description includes the TOE developer processes as well as any third-party (carrier) processes. The process description includes each deployment mechanism (e.g., over- the-air updates, per-carrier updates, downloaded updates).

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.

Application Note: The total length of time may be presented as a summation of the periods of time that each party (e.g., TOE developer, mobile carrier) on the critical path consumes. The time period until public availability per deployment mechanism may differ; each is described.

ALC_TSU_EXT.1.3C The description shall include the mechanisms publicly available for reporting security issues pertaining to the TOE.

Application Note: The reporting mechanism could include web sites, email addresses, as well as a means to protect the sensitive nature of the report (e.g., public keys that could be used to encrypt the details of a proof-of-concept exploit).

5.2.2 Mobile Device Fundamentals PP Assurance Activities

This section copies the assurance activities from the protection profile in order to ease reading and comparisons between the protection profile and the security target.

5.2.2.1 Cryptographic Support

5.2.2.1.1 Cryptographic Key Generation for Key Establishment (FCS_CKM.1(ASYM KA))

This assurance activity will verify the key generation and key establishments schemes used on the TOE.

Key Generation: The evaluator shall verify the implementation of the key generation routines of the supported schemes using the applicable tests below.

Key Generation for RSA-Based Key Establishment Schemes

The evaluator shall verify the implementation of RSA Key Generation by the TOE using the Key Generation test. This test verifies the ability of the TSF to correctly produce values for the key components including the public verification exponent *e*, the private prime factors *p* and *q*, the public modulus *n* and the calculation of the private signature exponent *d*.

Key Pair generation specifies 5 ways (or methods) to generate the primes *p* and *q*. These include:

1. Random Primes:
    - Provable primes
    - Probable primes
2. Primes with Conditions:
    - Primes *p1, p2, q1,q2, p* and *q* shall all be provable primes
    - Primes *p1, p2, q1,* and *q2* shall be provable primes and *p* and *q* shall be probable primes
    - Primes *p1, p2, q1,q2, p* and *q* shall all be probable primes

To test the key generation method for the Random Provable primes method and for all the Primes with Conditions methods, the evaluator must seed the TSF key generation routine with sufficient data to deterministically generate the RSA key pair. This includes the random seed(s), the public exponent of the RSA key, and the desired key length. For each key length supported, the evaluator shall have the TSF generate 25 key pairs. The evaluator shall verify the correctness of the TSF's implementation by comparing values generated by the TSF with those generated from a known good implementation.

Key Generation for Finite-Field Cryptography (FFC) – Based 56A Schemes

FFC Domain Parameter and Key Generation Tests

The evaluator shall verify the implementation of the Parameters Generation and the Key Generation for FFC by the TOE using the Parameter Generation and Key Generation test. This test verifies the ability of the TSF to correctly produce values for the field prime p, the cryptographic prime q (dividing p-1), the cryptographic group generator g, and the calculation of the private key x and public key y.

The Parameter generation specifies 2 ways (or methods) to generate the cryptographic prime q and the field prime p:

Cryptographic and Field Primes:

- Primes q and p shall both be provable primes
- Primes q and field prime p shall both be probable primes

and two ways to generate the cryptographic group generator g:

Cryptographic Group Generator:

- Generator g constructed through a verifiable process
- Generator g constructed through an unverifiable process.

The key generation specifies 2 ways to generate the private key x:

Private Key:

- len(q) bit output of RBG where 1 <=x <= q-1
- len(q) + 64 bit output of RBG, followed by a mod q-1 operation where 1<= x<=q-1.

The security strength of the RBG must be at least that of the security offered by the FFC parameter set.

To test the cryptographic and field prime generation method for the provable primes method and/or the group generator g for a verifiable process, the evaluator must seed the TSF parameter generation routine with sufficient data to deterministically generate the parameter set.

For each key length supported, the evaluator shall have the TSF generate 25 parameter sets and key pairs. The evaluator shall verify the correctness of the TSF's implementation by comparing values generated by the TSF with those generated from a known good implementation. Verification must also confirm

- g != 0,1
- q divides p-1
- g^q mod p = 1
- g^x mod p = y

for each FFC parameter set and key pair.

Key Generation for Elliptic Curve Cryptography (ECC) - Based 56A Schemes

ECC Key Generation Test

For each supported NIST curve, i.e., P-256, P-284 and P-521, the evaluator shall require the implementation under test (IUT) to generate 10 private/public key pairs. The private key shall be generated using an approved random bit generator (RBG). To determine correctness, the evaluator shall

submit the generated key pairs to the public key verification (PKV) function of a known good implementation.

ECC Public Key Verification (PKV) Test

For each supported NIST curve, i.e., P-256, P-284 and P-521, the evaluator shall generate 10 private/public key pairs using the key generation function of a known good implementation and modify five of the public key values so that they are incorrect, leaving five values unchanged (i.e., correct). The evaluator shall obtain in response a set of 10 PASS/FAIL values.

Key Establishment Schemes

The evaluator shall verify the implementation of the key establishment schemes of the supported by the TOE using the applicable tests below.

SP800-56A Key Establishment Schemes

The evaluator shall verify a TOE's implementation of SP800-56A key agreement schemes using the following Function and Validity tests. These validation tests for each key agreement scheme verify that a TOE has implemented the components of the key agreement scheme according to the specifications in the Recommendation. These components include the calculation of the DLC primitives (the shared secret value Z) and the calculation of the derived keying material (DKM) via the Key Derivation Function (KDF). If key confirmation is supported, the evaluator shall also verify that the components of key confirmation have been implemented correctly, using the test procedures described below. This includes the parsing of the DKM, the generation of MACdata and the calculation of MACtag.

Function Test

The Function test verifies the ability of the TOE to implement the key agreement schemes correctly. To conduct this test the evaluator shall generate or obtain test vectors from a known good implementation of the TOE supported schemes. For each supported key agreement scheme-key agreement role combination, KDF type, and, if supported, key confirmation role- key confirmation type combination, the tester shall generate 10 sets of test vectors. The data set consists of one set of domain parameter values (FFC) or the NIST approved curve (ECC) per 10 sets of public keys. These keys are static, ephemeral or both depending on the scheme being tested.

The evaluator shall obtain the DKM, the corresponding TOE‘s public keys (static and/or ephemeral), the MAC tag(s), and any inputs used in the KDF, such as the Other Information field OI and TOE id fields.

If the TOE does not use a KDF defined in SP 800-56A, the evaluator shall obtain only the public keys and the hashed value of the shared secret.

The evaluator shall verify the correctness of the TSF‘s implementation of a given scheme by using a known good implementation to calculate the shared secret value, derive the keying material DKM, and compare hashes or MAC tags generated from these values.

If key confirmation is supported, the TSF shall perform the above for each implemented approved MAC algorithm.

Validity Test

The Validity test verifies the ability of the TOE to recognize another party‘s valid and invalid key agreement results with or without key confirmation. To conduct this test, the evaluator shall obtain a list of the supporting cryptographic functions included in the SP800-56A key agreement implementation to determine which errors the TOE should be able to recognize. The evaluator generates a set of 24 (FFC) or 30 (ECC) test vectors consisting of data sets including domain parameter values or NIST approved curves, the evaluator‘s public keys, the TOE‘s public/private key pairs, MACTag, and any inputs used in the KDF, such as the other info and TOE id fields.

The evaluator shall inject an error in some of the test vectors to test that the TOE recognizes invalid key agreement results caused by the following fields being incorrect: the shared secret value Z, the DKM, the other information field OI, the data to be MACed, or the generated MACTag. If the TOE contains the full or partial (only ECC) public key validation, the evaluator will also individually inject errors in both parties‘ static public keys, both parties‘ ephemeral public keys and the TOE‘s static private key to assure the TOE detects errors in the public key validation function and/or the partial key validation function (in ECC only). At least two of the test vectors shall remain unmodified and therefore should result in valid key agreement results (they should pass).

The TOE shall use these modified test vectors to emulate the key agreement scheme using the corresponding parameters. The evaluator shall compare the TOE‘s results with the results using a known good implementation verifying that the TOE detects these errors.

SP800-56B Key Establishment Schemes

At this time, detailed test procedures for RSA-based key establishment schemes are not available. In order to show that the TSF complies with 800-56A and/or 800-56B, depending on the selections made, the evaluator shall ensure that the TSS contains the following information:

- The TSS shall list all sections of the appropriate 800-56 standard(s) to which the TOE complies.
- For each applicable section listed in the TSS, for all statements that are not "shall" (that is, "shall not", "should", and "should not"), if the TOE implements such options it shall be described in the TSS. If the included functionality is indicated as "shall not" or "should not" in the standard, the TSS shall provide a rationale for why this will not adversely affect the security policy implemented by the TOE.
- For each applicable section of 800-56A and 800-56B (as selected), any omission of functionality related to "shall" or “should” statements shall be described.

5.2.2.1.2 Cryptographic Key Generation for Authentication (FCS_CKM.1(ASYM AU))

If the TSF implements a FIPS 186-4 signature scheme, this requirement is verified under FCS_COP.1.1(3).

If the ESF implements the ANSI X9.31-1998 scheme, the evaluator shall check to ensure that the TSS describes how the key-pairs are generated. In order to show that the TSF implementation complies with ANSI X9.31-1998, the evaluator shall ensure that the TSS contains the following information:

- The TSS shall list all sections of the standard to which the TOE complies;
- For each applicable section listed in the TSS, for all statements that are not "shall" (that is, "shall not", "should", and "should not"), if the TOE implements such options it shall be described in the TSS. If the included functionality is indicated as "shall not" or "should not" in the standard, the TSS shall provide a rationale for why this will not adversely affect the security policy implemented by the TOE;
- For each applicable section of Appendix B, any omission of functionality related to "shall" or “should” statements shall be described.

5.2.2.1.3 Cryptographic Key Generation for WLAN (FCS_CKM.1(WLAN))

The cryptographic primitives will be verified through assurance activities specified later in this PP. The evaluator shall verify that the TSS describes how the primitives defined and implemented by this PP are used by the TOE in establishing and maintaining secure connectivity to the wireless clients. The TSS shall also provide a description of the developer‘s method(s) of assuring that their implementation conforms to the cryptographic standards; this includes not only testing done by the developing organization, but also any third-party testing that is performed (e.g. WPA2 certification). The evaluator shall ensure that the description of the testing methodology is of sufficient detail to determine the extent to which the details of the protocol specifics are tested.

5.2.2.1.4 Cryptographic Key Distribution for WLAN (FCS_CKM.2)

The evaluator shall check the TSS to ensure that it describes how the GTK is unwrapped prior to being installed for use on the TOE using the AES implementation specified in this PP. The evaluator shall also perform the following tests:

Test 1: The evaluator shall successfully connect the TOE to the access point. As the TOE is connected, the evaluator shall observe that the GTK is not transmitted in the clear between the TOE and the Access Point.

Test 2: The evaluator shall cause a broadcast message to be sent by the Access Point to which the TOE is connected. The evaluator shall ensure the message is encrypted and cannot be read in transit, and that the TOE is able to decrypt and read the message sent.

5.2.2.1.5 Extended: Cryptographic Key Support for Root Encryption Key (FCS_CKM_EXT.1)

The evaluator shall review the TSS to determine that a REK is supported by the product, that the TSS includes a description of the protection provided by the product for a REK, and that the TSS includes a description of the method of generation of a REK.

The evaluator shall verify that the description of the protection of a REK describes how any reading, import, and export of that REK is prevented. (For example, if the hardware protecting the REK is removable, the description should include how other devices are prevented from reading the REK.) The

evaluator shall verify that the TSS describes how encryption/decryption actions are isolated so as to prevent applications and system-level processes from reading the REK while allowing encryption/decryption by the key.

If "hardware-isolated" is selected and REK(s) are isolated from the rich OS by a separate processor execution environment, the evaluator shall verify that the description includes how the rich OS is prevented from accessing the memory containing REK key material , which software is allowed access to the REK, how any other software in the execution environment is prevented from reading that key material, and what other mechanisms prevent the REK key material from being written to shared memory locations between the rich OS and the separate execution environment.

If key derivation is performed using a REK, the evaluator shall ensure that the TSS description includes a description of the key derivation function and shall verify the key derivation uses an approved derivation mode and key expansion algorithm according to SP 800-108. (Additional key expansion algorithms are defined in other NIST Special Publications.)

The evaluator shall verify that the generation of a REK meets the FCS_RBG_EXT.1.1 and FCS_RBG_EXT.1.2 requirements:

- If REK(s) is/are generated on-device, the TSS shall include a description of the generation mechanism including what triggers a generation, how the functionality described by FCS_RBG_EXT.1 is invoked, and whether a separate instance of the RBG is used for REK(s).
- If REK(s) is/are generated off-device, the TSS shall include evidence that the RBG meets FCS_RBG_EXT.1.2. This will likely a second set of RBG documentation equivalent to the documentation provided for the RBG assurance activities. In addition, the TSS shall describe the manufacturing process that prevents the device manufacturer from accessing any REKs.

5.2.2.1.6 Extended: Cryptographic Key Random Generation for Data Encryption Keys (FCS_CKM_EXT.2)

The evaluator shall review the TSS to determine that it describes how the functionality described by FCS_RBG_EXT.1 is invoked to generate DEKs. The evaluator uses the description of the RBG functionality in FCS_RBG_EXT.1 or documentation available for the operational environment to determine that the key size being requested is identical to the key size and mode to be used for the encryption/decryption of the data.

5.2.2.1.7 Extended: Cryptographic Key Generation for Key Encryption Keys (FCS_CKM_EXT.3)

The evaluator shall examine the password hierarchy TSS to ensure that the formation of all KEKs is described and that the key sizes match that described by the ST author.

- The evaluator shall review the TSS to verify that it contains a description of the PBKDF use to derive KEKs. This description must include the size and storage location of salts. This activity may be performed in combination with that for FCS_COP.1(5).

- If the KEK is generated by an RBG, the evaluator shall review the TSS to determine that it describes how the functionality described by FCS_RBG_EXT.1 is invoked. The evaluator uses the

description of the RBG functionality in FCS_RBG_EXT.1 or documentation available for the operational environment to determine that the key size being requested is greater than or equal to the key size and mode to be used for the encryption/decryption of the data.

- If the KEK is generated according to an asymmetric key scheme, the evaluator shall review the TSS to determine that it describes how the functionality described by FCS_CKM.1(1) is invoked. The evaluator uses the description of the key generation functionality in FCS_CKM.1(1) or documentation available for the operational environment to determine that the key strength being requested is greater than or equal to 112 bits.

- If the KEK is formed from a combination, the evaluator shall verify that the TSS describes the method of combination and that this method is either an XOR, a KDF, or encryption. If a KDF is used, the evaluator shall ensure that the TSS description includes a description of the key derivation function and shall verify the key derivation uses an approved derivation mode and key expansion algorithm according to SP 800-108. (Additional key expansion algorithms are defined in other NIST Special Publications.)

5.2.2.1.8 Extended: Cryptographic Key Destruction (FCS_CKM_EXT.4)

The evaluator shall check to ensure the TSS describes when each of the plaintext keys (DEKs, software-based key storage, and KEKs) are cleared including on system power off, on wipe function, on disconnection of trusted channels, when no longer needed by the trusted channel per the protocol, when transitioning to the locked state (and possibly including immediately after use, while in the locked state, etc.); and the type of clearing procedure that is performed (cryptographic erase, overwrite with zeros, overwrite three or more times by a different alternating pattern, overwrite with random pattern, or block erase). If different types of memory are used to store the materials to be protected, the evaluator shall check to ensure that the TSS describes the clearing procedure in terms of the memory in which the data are stored (for example, "secret keys stored on flash are cleared by overwriting once with zeros, while secret keys stored on the internal persistent storage device are cleared by overwriting three times with a random pattern that is changed before each write").

Assurance Activity Note: The following tests require the developer to provide access to a test platform that provides the evaluator with tools that are typically not found on consumer Mobile Device products.

For each key clearing situation, including on system power off, on wipe function, on disconnection of trusted channels, when no longer needed by the trusted channel per the protocol, and when transitioning to the locked state (and possibly including immediately after use, while in the locked state, etc.) the evaluator shall repeat the following test.

Test 1: The evaluator shall utilize appropriate combinations of specialized operational environment and development tools (debuggers, simulators, etc.) for the TOE and instrumented TOE builds to test that keys are cleared correctly, including all intermediate copies of the key that may have been created internally by the TOE during normal cryptographic processing with that key.

Cryptographic TOE implementations in software shall be loaded and exercised under a debugger to perform such tests. The evaluator shall perform the following test for each key subject to clearing, including intermediate copies of keys that are persisted encrypted by the TOE:

1. Load the instrumented TOE build in a debugger.
2. Record the value of the key in the TOE subject to clearing.
3. Cause the TOE to perform a normal cryptographic processing with the key from #1.
4. Cause the TOE to clear the key.
5. Cause the TOE to stop the execution but not exit.
6. Cause the TOE to dump the entire memory footprint of the TOE into a binary file.
7. Search the content of the binary file created in #4 for instances of the known key value from #1.

The test succeeds if no copies of the key from #1 are found in step #7 above and fails otherwise.

The evaluator shall perform this test on all keys, including those persisted in encrypted form, to ensure intermediate copies are cleared.

Test 2: In cases where the TOE is implemented in firmware and operates in a limited operating environment that does not allow the use of debuggers, the evaluator shall utilize a simulator for the TOE on a general purpose operating system. The evaluator shall provide a rationale explaining the instrumentation of the simulated test environment and justifying the obtained test results.

5.2.2.1.9 Extended: TSF Wipe (FCS_CKM_EXT.5)

The evaluator shall check to ensure the TSS describes how the device is wiped; and the type of clearing procedure that is performed (cryptographic erase or overwrite) and, if overwrite is performed, the overwrite procedure (overwrite with zeros, overwrite three or more times by a different alternating pattern, overwrite with random pattern, or block erase). If different types of memory are used to store the data to be protected, the evaluator shall check to ensure that the TSS describes the clearing procedure in terms of the memory in which the data are stored (for example, "data stored on flash are cleared by overwriting once with zeros, while data stored on the internal persistent storage device are cleared by overwriting three times with a random pattern that is changed before each write").

Assurance Activity Note: The following tests require the developer to provide access to a test platform that provides the evaluator with tools that are typically not found on consumer Mobile Device products.

The assurance activities differ for the two wipe methods:

Test for Method 1: The evaluator shall enable encryption according to the AGD guidance. The evaluator shall use the test outlined for FCS_CKM_EXT.4, implementing the wipe command according to the AGD guidance provided for FMT_SMF.1 and as defined in Test 1, Step 4 of the assurance activities specified following FCS_CKM_EXT.4.

Test for Method 2: The evaluator shall enable encryption according to the AGD guidance. The evaluator shall create user data (protected data), for example, by using an application. The evaluator shall use a tool provided by the developer to examine this data stored in memory. The evaluator shall initiate the wipe command according to the AGD guidance provided for FMT_SMF.1. The evaluator shall use a tool provided by the developer to examine the same data location in memory to verify that the data has

been wiped according to the method described in the TSS. This test shall be repeated for each type of memory used to store the data to be protected.

5.2.2.1.10 Extended: Cryptographic Salt Generation (FCS_CKM_EXT.6)

The ST author shall provide a description in the TSS regarding the salt generation. The evaluator shall confirm that the salt is generating using an RBG described in FCS_RBG_EXT.1.

5.2.2.1.11 Cryptographic Operation for Data Encryption/Decryption (FCS_COP.1(SYM))

AES-CBC Tests

AES-CBC Known Answer Tests

There are four Known Answer Tests (KATs), described below. In all KATs, the plaintext, ciphertext, and IV values shall be 128-bit blocks. The results from each test may either be obtained by the evaluator directly or by supplying the inputs to the implementer and receiving the results in response. To determine correctness, the evaluator shall compare the resulting values to those obtained by submitting the same inputs to a known good implementation.

KAT-1. To test the encrypt functionality of AES-CBC, the evaluator shall supply a set of 10 plaintext values and obtain the ciphertext value that results from AES-CBC encryption of the given plaintext using a key value of all zeros and an IV of all zeros. Five plaintext values shall be encrypted with a 128-bit all-zeros key, and the other five shall be encrypted with a 256-bit all-zeros key.

To test the decrypt functionality of AES-CBC, the evaluator shall perform the same test as for encrypt, using 10 ciphertext values as input and AES-CBC decryption.

KAT-2. To test the encrypt functionality of AES-CBC, the evaluator shall supply a set of 10 key values and obtain the ciphertext value that results from AES-CBC encryption of an all-zeros plaintext using the given key value and an IV of all zeros. Five of the keys shall be 128-bit keys, and the other five shall be 256-bit keys.

To test the decrypt functionality of AES-CBC, the evaluator shall perform the same test as for encrypt, using an all-zero ciphertext value as input and AES-CBC decryption.

KAT-3. To test the encrypt functionality of AES-CBC, the evaluator shall supply the two sets of key values described below and obtain the ciphertext value that results from AES encryption of an all-zeros plaintext using the given key value and an IV of all zeros. The first set of keys shall have 128 128-bit keys, and the second set shall have 256 256-bit keys. Key i in each set shall have the leftmost i bits be ones and the rightmost N-i bits be zeros, for i in [1,N].

To test the decrypt functionality of AES-CBC, the evaluator shall supply the two sets of key and ciphertext value pairs described below and obtain the plaintext value that results from AES-CBC decryption of the given ciphertext using the given key and an IV of all zeros. The first set of key/ciphertext pairs shall have 128 128-bit key/ciphertext pairs, and the second set of key/ciphertext pairs shall have 256 256-bit key/ciphertext pairs. Key i in each set shall have the leftmost i bits be ones

and the rightmost N-i bits be zeros, for i in [1,N]. The ciphertext value in each pair shall be the value that results in an all-zeros plaintext when decrypted with its corresponding key.

KAT-4. To test the encrypt functionality of AES-CBC, the evaluator shall supply the set of 128 plaintext values described below and obtain the two ciphertext values that result from AES-CBC encryption of the given plaintext using a 128-bit key value of all zeros with an IV of all zeros and using a 256-bit key value of all zeros with an IV of all zeros, respectively. Plaintext value i in each set shall have the leftmost i bits be ones and the rightmost 128-i bits be zeros, for i in [1,128].

To test the decrypt functionality of AES-CBC, the evaluator shall perform the same test as for encrypt, using ciphertext values of the same form as the plaintext in the encrypt test as input and AES-CBC decryption.

AES-CBC Multi-Block Message Test

The evaluator shall test the encrypt functionality by encrypting an i-block message where 1 < *i* <=10. The evaluator shall choose a key, an IV and plaintext message of length i blocks and encrypt the message, using the mode to be tested, with the chosen key and IV. The ciphertext shall be compared to the result of encrypting the same plaintext message with the same key and IV using a known good implementation.

The evaluator shall also test the decrypt functionality for each mode by decrypting an *i*-block message where 1 < *i* <=10. The evaluator shall choose a key, an IV and a ciphertext message of length i blocks and decrypt the message, using the mode to be tested, with the chosen key and IV. The plaintext shall be compared to the result of decrypting the same ciphertext message with the same key and IV using a known good implementation.

AES-CBC Monte Carlo Tests

The evaluator shall test the encrypt functionality using a set of 200 plaintext, IV, and key 3- tuples. 100 of these shall use 128 bit keys, and 100 shall use 256 bit keys. The plaintext and IV values shall be 128-bit blocks. For each 3-tuple, 1000 iterations shall be run as follows:

```
# Input: PT, IV, Key
for i = 1 to 1000:
        if i == 1:
                CT[1] = AES-CBC-Encrypt(Key, IV, PT)
                PT = IV
        else:
                CT[i] = AES-CBC-Encrypt(Key, PT)
                PT = CT[i-1]
```

The ciphertext computed in the 1000th iteration (i.e., CT[1000]) is the result for that trial. This result shall be compared to the result of running 1000 iterations with the same values using a known good implementation.

The evaluator shall test the decrypt functionality using the same test as for encrypt, exchanging CT and PT and replacing AES-CBC-Encrypt with AES-CBC-Decrypt.

AES-CCM Tests

The evaluator shall test the generation-encryption and decryption-verification functionality of AES-CCM for the following input parameter and tag lengths:

128 bit and 256 bit keys

Two payload lengths. One payload length shall be the shortest supported payload length, greater than or equal to zero bytes. The other payload length shall be the longest supported payload length, less than or equal to 32 bytes (256 bits).

Two or three associated data lengths. One associated data length shall be 0, if supported. One associated data length shall be the shortest supported payload length, greater than or equal to zero bytes. One associated data length shall be the longest supported payload length, less than or equal to 32 bytes (256 bits). If the implementation supports an associated data length of 2^{16} bytes, an associated data length of 2^{16} bytes shall be tested.

Nonce lengths. All supported nonce lengths between 7 and 13 bytes, inclusive, shall be tested.

Tag lengths. All supported tag lengths of 4, 6, 8, 10, 12, 14 and 16 bytes shall be tested.

To test the generation-encryption functionality of AES-CCM, the evaluator shall perform the following four tests:

Test 1. For EACH supported key and associated data length and ANY supported payload, nonce and tag length, the evaluator shall supply one key value, one nonce value and 10 pairs of associated data and payload values and obtain the resulting ciphertext.

Test 2. For EACH supported key and payload length and ANY supported associated data, nonce and tag length, the evaluator shall supply one key value, one nonce value and 10 pairs of associated data and payload values and obtain the resulting ciphertext.

Test 3. For EACH supported key and nonce length and ANY supported associated data, payload and tag length, the evaluator shall supply one key value and 10 associated data, payload and nonce value 3-tuples and obtain the resulting ciphertext.

Test 4. For EACH supported key and tag length and ANY supported associated data, payload and nonce length, the evaluator shall supply one key value, one nonce value and 10 pairs of associated data and payload values and obtain the resulting ciphertext.

To determine correctness in each of the above tests, the evaluator shall compare the ciphertext with the result of generation-encryption of the same inputs with a known good implementation.

To test the decryption-verification functionality of AES-CCM, for EACH combination of supported associated data length, payload length, nonce length and tag length, the evaluator shall supply a key value and 15 nonce, associated data and ciphertext 3-tuples and obtain either a FAIL result or a PASS result with the decrypted payload. The evaluator shall supply 10 tuples that should FAIL and 5 that should PASS per set of 15. Additionally, the evaluator shall use tests from the IEEE 802.11-02/362r6 document —Proposed Test vectors for IEEE 802.11 TGi , dated September 10, 2002, Section 2.1 AES-CCMP Encapsulation Example and Section 2.2 Additional AES CCMP Test Vectors to further verify the IEEE 802.11-2007 implementation of AES-CCMP.

AES-GCM Monte Carlo Test

The evaluator shall test the authenticated encrypt functionality of AES-GCM for each combination of the following input parameter lengths:

128 bit and 256 bit keys

Two plaintext lengths. One of the plaintext lengths shall be a non-zero integer multiple of 128 bits, if supported. The other plaintext length shall not be an integer multiple of 128 bits, if supported.

Three AAD lengths. One AAD length shall be 0, if supported. One AAD length shall be a non-zero integer multiple of 128 bits, if supported. One AAD length shall not be an integer multiple of 128 bits, if supported.

Two IV lengths. If 96 bit IV is supported, 96 bits shall be one of the two IV lengths tested.

The evaluator shall test the encrypt functionality using a set of 10 key, plaintext, AAD, and IV tuples for each combination of parameter lengths above and obtain the ciphertext value and tag that results from AES-GCM authenticated encrypt. Each supported tag length shall be tested at least once per set of 10. The IV value may be supplied by the evaluator or the implementation being tested, as long as it is known.

The evaluator shall test the decrypt functionality using a set of 10 key, ciphertext, tag, AAD, and IV 5-tuples for each combination of parameter lengths above and obtain a Pass/Fail result on authentication and the decrypted plaintext if Pass. The set shall include five tuples that Pass and five that Fail.

The results from each test may either be obtained by the evaluator directly or by supplying the inputs to the implementer and receiving the results in response. To determine correctness, the evaluator shall compare the resulting values to those obtained by submitting the same inputs to a known good implementation.

XTS-AES Monte Carlo Test

The evaluator shall test the encrypt functionality of XTS-AES for each combination of the following input parameter lengths:

256 bit (for AES-128) and 512 bit (for AES-256) keys

Three data unit (i.e., plaintext) lengths. One of the data unit lengths shall be a non- zero integer multiple of 128 bits, if supported. One of the data unit lengths shall be an integer multiple of 128 bits, if supported. The third data unit length shall be either the longest supported data unit length or 2^{16} bits, whichever is smaller.

using a set of 100 (key, plaintext and 128-bit random tweak value) 3-tuples and obtain the ciphertext that results from XTS-AES encrypt.

The evaluator may supply a data unit sequence number instead of the tweak value if the implementation supports it. The data unit sequence number is a base-10 number ranging between 0 and 255 that implementations convert to a tweak value internally.

The evaluator shall test the decrypt functionality of XTS-AES using the same test as for encrypt, replacing plaintext values with ciphertext values and XTS-AES encrypt with XTS- AES decrypt.

AES Key Wrap (AES-KW) and Key Wrap with Padding (AES-KWP)

Test The evaluator shall test the authenticated encryption functionality of AES-KW for EACH combination of the following input parameter lengths:

128 and 256 bit key encryption keys (KEKs)

Three plaintext lengths. One of the plaintext lengths shall be two semi-blocks (128 bits). One of the plaintext lengths shall be three semi-blocks (192 bits). The third data unit length shall be the longest supported plaintext length less than or equal to 64 semi-blocks (4096 bits).

using a set of 100 key and plaintext pairs and obtain the ciphertext that results from AES-KW authenticated encryption. To determine correctness, the evaluator shall use the AES-KW authenticated-encryption function of a known good implementation.

The evaluator shall test the authenticated-decryption functionality of AES-KW using the same test as for authenticated-encryption, replacing plaintext values with ciphertext values and AES-KW authenticated-encryption with AES-KW authenticated-decryption.

The evaluator shall test the authenticated-encryption functionality of AES-KWP using the same test as for AES-KW authenticated-encryption with the following change in the three plaintext lengths:

One plaintext length shall be one octet. One plaintext length shall be 20 octets (160 bits).

One plaintext length shall be the longest supported plaintext length less than or equal to 512 octets (4096 bits).

The evaluator shall test the authenticated-decryption functionality of AES-KWP using the same test as for AES-KWP authenticated-encryption, replacing plaintext values with ciphertext values and AES-KWP authenticated-encryption with AES-KWP authenticated- decryption.

5.2.2.1.12 Cryptographic Operation for Hashing (FCS_COP.1(HASH))

The evaluator checks the AGD documents to determine that any configuration that is required to be done to configure the functionality for the required hash sizes is present. The evaluator shall check that the association of the hash function with other TSF cryptographic functions (for example, the digital signature verification function) is documented in the TSS.

The TSF hashing functions can be implemented in one of two modes. The first mode is the byte-oriented mode. In this mode the TSF only hashes messages that are an integral number of bytes in length; i.e., the length (in bits) of the message to be hashed is divisible by 8. The second mode is the bit-oriented mode. In this mode the TSF hashes messages of arbitrary length. As there are different tests for each mode, an indication is given in the following sections for the bit-oriented vs. the byte-oriented testmacs.

The evaluator shall perform all of the following tests for each hash algorithm implemented by the TSF and used to satisfy the requirements of this PP.

Short Messages Test - Bit-oriented Mode

The evaluators devise an input set consisting of m+1 messages, where m is the block length of the hash algorithm. The length of the messages range sequentially from 0 to m bits. The message text shall be pseudorandomly generated. The evaluators compute the message digest for each of the messages and ensure that the correct result is produced when the messages are provided to the TSF.

Short Messages Test - Byte-oriented Mode

The evaluators devise an input set consisting of m/8+1 messages, where m is the block length of the hash algorithm. The length of the messages range sequentially from 0 to m/8 bytes, with each message being an integral number of bytes. The message text shall be pseudorandomly generated. The evaluators compute the message digest for each of the messages and ensure that the correct result is produced when the messages are provided to the TSF.

Selected Long Messages Test - Bit-oriented Mode

The evaluators devise an input set consisting of m messages, where m is the block length of the hash algorithm. The length of the ith message is 512 + 99*i, where $1 \leq i \leq m$. The message text shall be pseudorandomly generated. The evaluators compute the message digest for each of the messages and ensure that the correct result is produced when the messages are provided to the TSF.

Selected Long Messages Test - Byte-oriented Mode

The evaluators devise an input set consisting of m/8 messages, where m is the block length of the hash algorithm. The length of the ith message is 512 + 8*99*i, where $1 \leq i \leq m/8$. The message text shall be

pseudorandomly generated. The evaluators compute the message digest for each of the messages and ensure that the correct result is produced when the messages are provided to the TSF.

Pseudorandomly Generated Messages Test

This test is for byte-oriented implementations only. The evaluators randomly generate a seed that is n bits long, where n is the length of the message digest produced by the hash function to be tested. The evaluators then formulate a set of 100 messages and associated digests by following the algorithm provided in Figure 1 of [SHAVS]. The evaluators then ensure that the correct result is produced when the messages are provided to the TSF.

5.2.2.1.13 Cryptographic Operation for Signature Algorithms (FCS_COP.1(SIGN))

Key Generation:

Key Generation for RSA Signature Schemes

The evaluator shall verify the implementation of RSA Key Generation by the TOE using the Key Generation test. This test verifies the ability of the TSF to correctly produce values for the key components including the public verification exponent *e*, the private prime factors *p* and *q*, the public modulus *n* and the calculation of the private signature exponent *d*.

Key Pair generation specifies 5 ways (or methods) to generate the primes p and q. These include:

1) Random Primes:
    - Provable primes
    - Probable primes
2) Primes with Conditions:
    - Primes p1, p2, q1,q2, p and q shall all be provable primes
    - Primes p1, p2, q1, and q2 shall be provable primes and p and q shall be probable primes
    - Primes p1, p2, q1,q2, p and q shall all be probable primes

To test the key generation method for the Random Provable primes method and for all the Primes with Conditions methods, the evaluator must seed the TSF key generation routine with sufficient data to deterministically generate the RSA key pair. This includes the random seed(s), the public exponent of the RSA key, and the desired key length. For each key length supported, the evaluator shall have the TSF generate 25 key pairs. The evaluator shall verify the correctness of the TSF's implementation by comparing values generated by the TSF with those generated from a known good implementation.

ECDSA Key Generation Tests

FIPS 186-4 ECDSA Key Generation Test

For each supported NIST curve, i.e., P-256, P-284 and P-521, the evaluator shall require the implementation under test (IUT) to generate 10 private/public key pairs. The private key shall be generated using an approved random bit generator (RBG). To determine correctness, the evaluator shall

submit the generated key pairs to the public key verification (PKV) function of a known good implementation.

FIPS 186-4 Public Key Verification (PKV) Test

For each supported NIST curve, i.e., P-256, P-284 and P-521, the evaluator shall generate 10 private/public key pairs using the key generation function of a known good implementation and modify five of the public key values so that they are incorrect, leaving five values unchanged (i.e., correct). The evaluator shall obtain in response a set of 10 PASS/FAIL values.

ECDSA Algorithm Tests

ECDSA FIPS 186-4 Signature Generation Test

For each supported NIST curve (i.e., P-256, P-284 and P-521) and SHA function pair, the evaluator shall generate 10 1024-bit long messages and obtain for each message a public key and the resulting signature values R and S. To determine correctness, the evaluator shall use the signature verification function of a known good implementation.

ECDSA FIPS 186-4 Signature Verification Test

For each supported NIST curve (i.e., P-256, P-284 and P-521) and SHA function pair, the evaluator shall generate a set of 10 1024-bit message, public key and signature tuples and modify one of the values (message, public key or signature) in five of the 10 tuples. The evaluator shall obtain in response a set of 10 PASS/FAIL values.

RSA Signature Algorithm Tests

Signature Generation Test

The evaluator shall verify the implementation of RSA Signature Generation by the TOE using the Signature Generation Test. To conduct this test the evaluator must generate or obtain 10 messages from a trusted reference implementation for each modulus size/SHA combination supported by the TSF. The evaluator shall have the TOE use their private key and modulus value to sign these messages. The evaluator shall verify the correctness of the TSF‘s signature using a known good implementation and the associated public keys to verify the signatures.

Signature Verification Test

The evaluator shall perform the Signature Verification test to verify the ability of the TOE to recognize another party‘s valid and invalid signatures. The evaluator shall inject errors into the test vectors produced during the Signature Verification Test by introducing errors in some of the public keys e, messages, IR format, and/or signatures. The TOE attempts to verify the signatures and returns success or failure.

The evaluator shall use these test vectors to emulate the signature verification test using the corresponding parameters and verify that the TOE detects these errors.

5.2.2.1.14 Cryptographic Operation for Keyed Hash Algorithms (FCS_COP.1(HMAC))

The evaluator shall examine the TSS to ensure that it specifies the following values used by the HMAC function: key length, hash function used, block size, and output MAC length used.

For each of the supported parameter sets, the evaluator shall compose 15 sets of test data. Each set shall consist of a key and message data. The evaluator shall have the TSF generate HMAC tags for these sets of test data. The resulting MAC tags shall be compared to the result of generating HMAC tags with the same key and IV using a known good implementation.

5.2.2.1.15 Cryptographic Operation for Password Based Key Derivation (FCS_COP.1(PBKD))

The evaluator shall check that the TSS describes the method by which the password is first encoded and then fed to the SHA algorithm. The settings for the algorithm (padding, blocking, etc.) shall be described, and the evaluator shall verify that these are supported by the selections in this component as well as the selections concerning the hash function itself. The evaluator shall verify that the TSS contains a description of how the output of the hash function is used to form the submask that will be input into the function and is the same length as the DEK as specified in FCS_CKM_EXT.2.

For the NIST SP 800-132-based conditioning of the passphrase, the required assurance activities will be performed when doing the assurance activities for the appropriate requirements (FCS_COP.1.1(4)). If any manipulation of the key is performed in forming the submask that will be used to form the KEK, that process shall be described in the TSS.

No explicit testing of the formation of the submask from the input password is required.

5.2.2.1.16 Extended: Initialization Vector Generation (FCS_IV_EXT.1)

The evaluator shall examine the key hierarchy section of the TSS to ensure that the encryption of all keys is described and the formation of the IVs for each key encrypted by the same KEK meets FCS_IV_EXT.1.

5.2.2.1.17 Extended: Random Bit Generation (FCS_RBG_EXT.1)

Documentation shall be produced — and the evaluator shall perform the activities — in accordance with Annex E.

The evaluator shall verify that the API documentation provided according to Section 6.2.1 includes the security functions described in FCS_RBG_EXT.1.3.

The evaluator shall perform the following tests, depending on the standard to which the RBG conforms.

Implementations Conforming to FIP 140-2 Annex C

The reference for the tests contained in this section is The Random Number Generator Validation System (RNGVS). The evaluators shall conduct the following two tests. Note that the "expected values" are produced by a reference implementation of the algorithm that is known to be correct. Proof of correctness is left to each Scheme.

The evaluators shall perform a Variable Seed Test. The evaluators shall provide a set of 128 (Seed, DT) pairs to the TSF RBG function, each 128 bits. The evaluators shall also provide a key (of the length appropriate to the AES algorithm) that is constant for all 128 (Seed, DT) pairs. The DT value is incremented by 1 for each set. The seed values shall have no repeats within the set. The evaluators ensure that the values returned by the TSF match the expected values.

The evaluators shall perform a Monte Carlo Test. For this test, they supply an initial Seed and DT value to the TSF RBG function; each of these is 128 bits. The evaluators shall also provide a key (of the length appropriate to the AES algorithm) that is constant throughout the test. The evaluators then invoke the TSF RBG 10,000 times, with the DT value being incremented by 1 on each iteration, and the new seed for the subsequent iteration produced as specified in NIST-Recommended Random Number Generator Based on ANSI X9.31 Appendix A.2.4 Using the 3-Key Triple DES and AES Algorithms, Section 3. The evaluators ensure that the 10,000th value produced matches the expected value.

Implementations Conforming to NIST Special Publication 800-90A

The evaluator shall perform 15 trials for the RNG implementation. If the RNG is configurable, the evaluator shall perform 15 trials for each configuration. The evaluator shall also confirm that the operational guidance contains appropriate instructions for configuring the RNG functionality.

If the RNG has prediction resistance enabled, each trial consists of (1) instantiate DRBG, (2) generate the first block of random bits (3) generate a second block of random bits (4) uninstantiate. The evaluator verifies that the second block of random bits is the expected value. The evaluator shall generate eight input values for each trial. The first is a count (0 – 14). The next three are entropy input, nonce, and personalization string for the instantiate operation. The next two are additional input and entropy input for the first call to generate. The final two are additional input and entropy input for the second call to generate. These values are randomly generated. —generate one block of random bits means to generate random bits with number of returned bits equal to the Output Block Length (as defined in NIST SP800-90A).

If the RNG does not have prediction resistance, each trial consists of (1) instantiate DRBG, (2) generate the first block of random bits (3) reseed, (4) generate a second block of random bits (5) uninstantiate. The evaluator verifies that the second block of random bits is the expected value. The evaluator shall generate eight input values for each trial. The first is a count (0 – 14). The next three are entropy input, nonce, and personalization string for the instantiate operation. The fifth value is additional input to the first call to generate. The sixth and seventh are additional input and entropy input to the call to reseed. The final value is additional input to the second generate call.

The following paragraphs contain more information on some of the input values to be generated/selected by the evaluator.

- **Entropy input**: the length of the entropy input value must equal the seed length.
- **Nonce**: If a nonce is supported (CTR_DRBG with no Derivation Function does not use a nonce), the nonce bit length is one-half the seed length.

- **Personalization string**: The length of the personalization string must be <= seed length. If the implementation only supports one personalization string length, then the same length can be used for both values. If more than one string length is support, the evaluator shall use personalization strings of two different lengths. If the implementation does not use a personalization string, no value needs to be supplied.
- **Additional input**: the additional input bit lengths have the same defaults and restrictions as the personalization string lengths.

5.2.2.1.18 Extended: Cryptographic Algorithm Services (FCS_SRV_EXT.1)

The evaluator shall verify that the API documentation provided according to Section 6.2.1 includes the security functions (cryptographic algorithms) described in these requirements.

The evaluator shall write, or the developer shall provide access to, an application that requests cryptographic operations by the TSF. The evaluator shall verify that the results from the validation match the expected results according to the API documentation. This application may be used to assist in verifying the cryptographic operation assurance activities for the other algorithm services requirements.

5.2.2.1.19 Extended: Cryptographic Key Storage (FCS_STG_EXT.1)

The assurance activity for this component entails examination of the ST‘s TSS to determine that the TOE‘s implements the required secure key storage.

The evaluator shall review the AGD guidance to determine that it describes the steps needed to import or destroy keys/secrets. The evaluator shall also verify that the API documentation provided according to Section 6.2.1 includes the security functions (import, use, and destruction) described in these requirements. The API documentation shall include the method by which applications restrict access to their keys/secrets in order to meet FCS_STG_EXT.1.4.

The evaluator shall test the functionality of each security function:

Test 1: The evaluator shall import keys/secrets of each supported type according to the AGD guidance. The evaluator shall write, or the developer shall provide access to, an application that generates a key/secret of each supported type and calls the import functions. The evaluator shall verify that no errors occur during import.

Test 2: The evaluator shall write, or the developer shall provide access to, an application that uses an imported key/secret:

- For RSA, the secret shall be used to sign data.

In the future additional types will be required to be tested:

- For ECDSA, the secret shall be used to sign data
- For symmetric algorithms, the secret shall be used to encrypt data.
- For persistent secrets, the secret shall be compared to the imported secret.

The evaluator shall repeat this test with the application-imported keys/secrets and a different application‘s imported keys/secrets. The evaluator shall verify that the TOE requires approval before allowing the application to use the key/secret imported by the user or by a different application:

- The evaluator shall deny the approvals to verify that the application is not able to use the key/secret as described.
- The evaluator shall repeat the test, allowing the approvals to verify that the application is able to use the key/secret as described.

If the ST Author has selected “common application developer”, this test is performed by either using applications from different developers or appropriately (according to API documentation) not authorizing sharing.

Test 3: The evaluator shall destroy keys/secrets of each supported type according to the AGD guidance. The evaluator shall write, or the developer shall provide access to, an application that destroys an imported key/secret.

The evaluator shall repeat this test with the application-imported keys/secrets and a different application‘s imported keys/secrets. The evaluator shall verify that the TOE requires approval before allowing the application to destroy the key/secret imported by the administrator or by a different application:

- The evaluator shall deny the approvals and verify that the application is still able to use the key/secret as described.
- The evaluator shall repeat the test, allowing the approvals and verifying that the application is no longer able to use the key/secret as described.

If the ST Author has selected “common application developer”, this test is performed by either using applications from different developers or appropriately (according to API documentation) not authorizing sharing.

Assurance Activity Note: The following tests require the developer to provide access to a test platform that provides the evaluator with tools that are typically not found on consumer Mobile Device products.

Test 5: The evaluator shall enable encryption according to the AGD guidance. The evaluator shall use the test outlined for FCS_CKM_EXT.4, destroy keys/secrets according to the AGD guidance provided for FMT_SMF_EXT.1 and as defined in Test 1, Step 4 of the assurance activities specified following FCS_CKM_EXT.4.

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

The evaluator shall review the TSS to determine that the TSS includes key hierarchy description of the protection of each DEK for data-at-rest, of software-based key storage, and of KEK related to the protection of the DEKs and software-based key storage. This description must include a diagram illustrating the key hierarchy implemented by the TOE in order to demonstrate that the implementation meets FCS_STG_EXT.2. The description shall indicate how the functionality described by FCS_RBG_EXT.1

is invoked to generate DEKs (FCS_CKM_EXT.2), the key size (FCS_CKM_EXT.2 and FCS_CKM_EXT.3) for each key, how each KEK is formed (generated, derived, or combined according to FCS_CKM_EXT.3), the integrity protection method for each encrypted key (FCS_STG_EXT.3), and the IV generation for each key encrypted by the same KEK (FCS_IV_EXT.1). More detail for each task follows the corresponding requirement.

The evaluator shall examine the key hierarchy section of the TSS to ensure that each key (DEKs, software-based key storage, and KEKs) is encrypted by keys of equal or greater security strength using one of the selected modes.

The evaluator shall examine the key hierarchy description in the TSS section to verify that each DEK and software-stored key is encrypted according to FCS_STG_EXT.2.

5.2.2.1.21 Extended: Integrity of Encrypted Key Storage (FCS_STG_EXT.3)

The evaluator shall examine the key hierarchy description in the TSS section to verify that each encrypted key is integrity protected according to one of the options in FCS_STG_EXT.3.

5.2.2.1.22 Extended: EAP TLS Protocol (FCS_TLS_EXT.1)

The evaluator shall check the description of the implementation of this protocol in the TSS to ensure that the ciphersuites supported are specified. The evaluator shall check the TSS to ensure that the ciphersuites specified include those listed for this component. The evaluator shall also check the operational guidance to ensure that it contains instructions on configuring the TOE so that TLS conforms to the description in the TSS.

The evaluator shall check that the AGD guidance contains instructions for the administrator to configure the list of Certificate Authorities that are allowed to sign certificates or to configure the FQDN of the authentication server certificate that will be accepted by the TOE in the EAP-TLS exchange.

Additional tests may be added in the future to test compliance with RFC 5246. The evaluator shall also perform the following tests:

- *Test 1*: The evaluator shall establish a TLS connection using each of the ciphersuites specified by the requirement. This connection may be established as part of the establishment of a higher-level protocol, e.g., as part of an EAP session. It is sufficient to observe the successful negotiation of a ciphersuite to satisfy the intent of the test; it is not necessary to examine the characteristics of the encrypted traffic in an attempt to discern the ciphersuite being used (for example, that the cryptographic algorithm is 128-bit AES and not 256-bit AES).
- *Test 2*: The following test is repeated for each supported certificate signing algorithm supported. The evaluator shall attempt to establish the connection using a server with a authentication server certificate that contains the Server Authentication purpose in the extendedKeyUsage field and verify that a connection is established. The evaluator will then verify that the client rejects an otherwise valid server certificate that lacks the Server Authentication purpose in the extendedKeyUsage field and a connection is not established. Ideally, the two certificates should be identical except for the extendedKeyUsage field.

- *Test 3*: Following the guidance provided by the AGD guidance, a CA or an FQDN will be configured as “acceptable” for authentication server certificates and then the evaluator will start a wireless connection and verify that the wireless client is able to successfully connect. The evaluator will then configure the system such that an otherwise valid certificate is signed by a CA that is not allowed by the TOE or presents a FQDN that is not allowed by the TOE. Attempts to authenticate to an authentication server presenting such a certificate should result in the connection being refused. If the TOE supports both methods of limiting the acceptable authentication servers, the evaluator shall repeat this test twice, once with each method.
- *Test 4*: The evaluator shall configure the authentication server to send a certificate in the TLS connection that the does not match the server-selected ciphersuite (for example, send a ECDSA certificate while using the TLS_RSA_WITH_AES_128_CBC_SHA ciphersuite or send a RSA certificate while using one of the ECDSA ciphersuites.) The evaluator shall verify that the TOE disconnects after receiving the server‘s Certificate handshake message.
- *Test 5*: The evaluator shall setup a man-in-the-middle tool between the TOE and the authentication server and shall perform the following modifications to the traffic:
    - Modify at least one byte in the server‘s nonce in the Server Hello handshake message, and verify that the server denies the client‘s Finished handshake message.
    - Modify the server‘s selected ciphersuite in the Server Hello handshake message to be a ciphersuite not presented in the Client Hello handshake message. The evaluator shall verify that the client rejects the connection after receiving the Server Hello.
    - (*conditional*) If a DHE or ECDHE ciphersuite is supported, modify the signature block in the Server‘s KeyExchange handshake message, and verify that the client rejects the connection after receiving the Server KeyExchange.
    - Modify a byte in a CA field in the Server‘s Certificate Request handshake message. The modified CA field must not be the CA used to sign the client‘s certificate. The evaluator shall verify that the server rejects the connection after receiving the Client Finished handshake message.
    - Modify a byte in the Server Finished handshake message, and verify that the client sends a fatal alert upon receipt and does not send any application data.

5.2.2.1.23 Extended: TLS Protocol (FCS_TLS_EXT.2)

The evaluator shall check the description of the implementation of this protocol in the TSS to ensure that the ciphersuites supported are specified. The evaluator shall check the TSS to ensure that the ciphersuites specified include those listed for this component. The evaluator shall also check the operational guidance to ensure that it contains instructions on configuring the TOE so that TLS conforms to the description in the TSS.

The evaluator shall verify that the TSS describes how the DN in the certificate is compared to the expected DN. If the DN is not compared automatically to the Domain Name or IP address, the evaluator shall ensure that the AGD guidance includes configuration of the expected DN for the connection.

Additional tests may be added in the future to test compliance with RFC 5246. The evaluator shall also perform the following tests:

- *Test 1*: The evaluator shall establish a TLS connection using each of the ciphersuites specified by the requirement. This connection may be established as part of the establishment of a higher-level protocol, e.g., as part of an EAP session. It is sufficient to observe the successful negotiation of a ciphersuite to satisfy the intent of the test; it is not necessary to examine the characteristics of the encrypted traffic in an attempt to discern the ciphersuite being used (for example, that the cryptographic algorithm is 128-bit AES and not 256-bit AES).
- *Test 2*: The following test is repeated for each supported certificate signing algorithm supported. The evaluator shall attempt to establish the connection using a server with a server certificate that contains the Server Authentication purpose in the extendedKeyUsage field and verify that a connection is established. The evaluator will then verify that the client rejects an otherwise valid server certificate that lacks the Server Authentication purpose in the extendedKeyUsage field and a connection is not established. Ideally, the two certificates should be identical except for the extendedKeyUsage field.
- *Test 3*: The evaluator shall attempt a connection with a certificate where the DN matches either the configured expected DN or the Domain Name/IP address of the peer. The evaluator shall verify that the TSF is able to successfully connect. The evaluator shall attempt a connection with a certificate where the DN does not match either the configured expected DN or the Domain Name/IP address of the peer. The evaluator shall verify that the TSF is not able to successfully connect.
- *Test 4*: The evaluator shall configure the server to send a certificate in the TLS connection that the does not match the server-selected ciphersuite (for example, send a ECDSA certificate while using the TLS_RSA_WITH_AES_128_CBC_SHA ciphersuite or send a RSA certificate while using one of the ECDSA ciphersuites.) The evaluator shall verify that the TOE disconnects after receiving the server‘s Certificate handshake message.
- *Test 5*: The evaluator shall setup a man-in-the-middle tool between the TOE and the server and shall perform the following modifications to the traffic:
    - Modify at least one byte in the server‘s nonce in the Server Hello handshake message, and verify that the server denies the client‘s Finished handshake message.
    - Modify the server‘s selected ciphersuite in the Server Hello handshake message to be a ciphersuite not presented in the Client Hello handshake message. The evaluator shall verify that the client rejects the connection after receiving the Server Hello.
    - (*conditional*) If a DHE or ECDHE ciphersuite is supported, modify the signature block in the Server‘s KeyExchange handshake message, and verify that the client rejects the connection after receiving the Server KeyExchange.
    - Modify a byte in a CA field in the Server‘s Certificate Request handshake message. The modified CA field must not be the CA used to sign the client‘s certificate. The evaluator shall verify that the server rejects the connection after receiving the Client Finished handshake message.
    - Modify a byte in the Server Finished handshake message, and verify that the client sends a fatal alert upon receipt and does not send any application data.

5.2.2.1.24 Extended: HTTPS Protocol (FCS_HTTPS_EXT.1)

The evaluator shall check the TSS to ensure that it is clear on how HTTPS uses TLS to establish an administrative session, focusing on any client authentication required by the TLS protocol vs. security administrator authentication which may be done at a different level of the processing stack. Testing for this activity is done as part of the TLS testing; this may result in additional testing if the TLS tests are done at the TLS protocol level.

5.2.2.2 User Data Protection

5.2.2.2.1 Extended: Security Attribute Based Access Control (FDP_ACF_EXT.1)

The evaluator shall ensure the TSS lists all system services available for use by an application. The evaluator shall also ensure that the TSS describes how applications interface with these system services, and means by which these system services are protected by the TSF.

The TSS shall describe which of the following categories each system service falls in:

1) No applications are allowed access
2) Privileged applications are allowed access
3) Applications are allowed access by user authorization
4) All applications are allowed access

Privileged applications include any applications developed by the TSF developer. The TSS shall describe how privileges are granted to third-party applications. For both types of privileged applications, the TSS shall describe how and when the privileges are verified and how the TSF prevents unprivileged applications from accessing those services.

For any services for which the user may grant access, the evaluator shall ensure that the TSS identifies whether the user is prompted for authorization when the application is installed, or during runtime.

Assurance Activity Note: The following tests require the vendor to provide access to a test platform that provides the evaluator with tools that are typically not found on consumer Mobile Device products.

The evaluator shall write, or the developer shall provide, applications for the purposes of the following tests.

Test 1: For each system service to which no applications are allowed access, the evaluator shall attempt to access the system service with a test application and verify that the application is not able to access that system service.

Test 2: For each system service to which only privileged applications are allowed access, the evaluator shall attempt to access the system service with an unprivileged application and verify that the application is not able to access that system service. The evaluator shall attempt to access the system service with a privileged application and verify that the application can access the service.

Test 3: For each system service to which the user may grant access, the evaluator shall attempt to access the system service with a test application. The evaluator shall ensure that either the system blocks such

accesses or prompts for user authorization. The prompt for user authorization may occur at runtime or at installation time, and should be consistent with the behavior described in the TSS.

Test 4: For each system service listed in the TSS that is accessible by all applications, the evaluator shall test that an application can access that system service.

5.2.2.2.2 Extended: Data at Rest Protection (FDP_DAR_EXT.1)

The evaluator shall verify that the TSS section of the ST indicates which data is protected by the DAR implementation and what data is considered TSF data. The evaluator shall ensure that this data includes all protected data.

The evaluator shall review the AGD guidance to determine that the description of the configuration and use of the DAR protection does not require the user to perform any actions beyond configuration and providing the authentication credential. The evaluator shall also review the AGD guidance to determine that the configuration does not require the user to identify encryption on a per-file basis.

Assurance Activity Note: The following test require the developer to provide access to a test platform that provides the evaluator with tools that are typically not found on consumer Mobile Device products.

Test 1: The evaluator shall enable encryption according to the AGD guidance. The evaluator shall create user data (non-system) either by creating a file or by using an application. The evaluator shall use a tool provided by the developer to verify that this data is encrypted when the product is powered off, in conjunction with Test 1 for FIA_UAU_EXT.1.

5.2.2.2.3 Extended: Sensitive Data Encryption (FDP_DAR_EXT.2)

The evaluator shall verify that the TSS includes a description of which data stored by the TSF (such as by native applications) is treated as sensitive. This data may include all or some user or enterprise data and must be specific regarding the level of protection of email, contacts, calendar appointments, messages, and documents.

The evaluator shall examine the TSS to determine that it describes the mechanism that is provided for applications to use to mark data and keys as sensitive. This description shall also contain information reflecting how data and keys marked in this manner are distinguished from data and keys that are not (for instance, tagging, segregation in a "special" area of memory or container, etc.).

Test 1: The evaluator shall enable encryption of sensitive data and require user authentication according to the AGD guidance. The evaluator shall try to access and create sensitive data (as defined in the ST and either by creating a file or using an application to generate sensitive data) in order to verify that no other user interaction is required.

The evaluator shall review the TSS section of the ST to determine that the TSS includes a description of the process of receiving sensitive data while the device is in a locked state. The evaluator shall also verify that the description indicates if sensitive data that may be received in the locked state is treated differently than sensitive data that cannot be received in the locked state. The description shall include the key scheme for encrypting and storing the received data, which must involve an asymmetric key and

must prevent the sensitive data-at-rest from being decrypted by wiping all key material used to derive or encrypt the data (as described in the application note). The introduction to this section provides two different schemes that meet the requirements, but other solutions may address this requirement.

The evaluator shall perform the tests in FCS_CKM_EXT.4 for all key material no longer needed while in the locked state and shall ensure that keys for the asymmetric scheme are addressed in the tests performed when transitioning to the locked state.

The evaluator shall verify that the key hierarchy section of the TSS required for FCS_STG_EXT.2 includes the symmetric encryption keys (DEKs) used to encrypt sensitive data. The evaluator shall ensure that these DEKs are encrypted by a key encrypted with (or chain to a KEK encrypted with) the REK and password-derived KEK.

The evaluator shall verify that the TSS section of the ST that describes the asymmetric key scheme includes the protection of any private keys of the asymmetric pairs. The evaluator shall ensure that any private keys that are not wiped and are stored by the TSF are stored encrypted by a key encrypted with (or chain to a KEK encrypted with) the REK and password-derived KEK.

The evaluator shall verify that the TSS section of the ST that describes the asymmetric key scheme includes a description of the actions taken by the TSF for the purposes of DAR upon transitioning to the unlocked state. These actions shall minimally include decrypting all received data using the asymmetric key scheme and re-encrypting with the symmetric key scheme used to store data while the device is unlocked.

5.2.2.2.4 Extended: Certificate Data Storage (FDP_STG_EXT.1)

The evaluator shall ensure the TSS describes the Trust Anchor Database implemented that contain certificates used to meet the requirements of this PP. This description shall contain information pertaining to how certificates are loaded into the store, and how the store is protected from unauthorized access (for example, unix permissions) in accordance with the permissions established in FMT_SMF.1, FMT_MOF.1(1), and FMT_MOF.1(2).

5.2.2.3 Identification and Authentication

5.2.2.3.1 Extended: Authorization Failure Handling (FIA_AFL_EXT.1)

The evaluator shall ensure that the TSS describes that a value corresponding to the number of unsuccessful authentication attempts since the last successful authentication is kept for each user. The evaluator shall verify that the AGD guidance describes how the administrator configures the maximum number of unsuccessful authentication attempts and the remediation action to be performed when that maximum is met or surpassed.

Test 1: The evaluator shall configure according to the AGD guidance the device with a maximum number of unsuccessful authentication attempts and with a remediation action to be performed when that maximum is met or surpassed. The evaluator shall enter the locked state and enter incorrect passwords until the remediation action occurs. The evaluator shall verify that the number of password entries corresponds to the configured maximum and that the remediation action is implemented.

5.2.2.3.2 Extended: Bluetooth Authentication (FIA_BLT_EXT.1)

The evaluator shall ensure that the TSS describes how data transfer is prevented before the Bluetooth pairing is completed. The TSS shall specifically call out any supported OBEX data transfer mechanisms. The evaluator shall ensure that the OBEX transfers are only completed after the Bluetooth devices are paired.

5.2.2.3.3 Extended: PAE Authentication (FIA_PAE_EXT.1)

The evaluator shall perform the following tests:

- *Test 1*: The evaluator shall demonstrate that the TOE has no access to the test network. After successfully authenticating with an authentication server through a wireless access system, the evaluator shall demonstrate that the TOE does have access to the test network.
- *Test 2*: The evaluator shall demonstrate that the TOE has no access to the test network. The evaluator shall attempt to authenticate using an invalid client certificate, such that the EAP-TLS negotiation fails. This should result in the TOE still being unable to access the test network.
- *Test 3*: The evaluator shall demonstrate that the TOE has no access to the test network. The evaluator shall attempt to authenticate using an invalid authentication server certificate, such that the EAP-TLS negotiation fails. This should result in the TOE still being unable to access the test network.

5.2.2.3.4 Extended: Password Management (FIA_PMG_EXT.1)

The evaluator shall examine the operational guidance to determine that it provides guidance to security administrators on the composition of strong passwords, and that it provides instructions on setting the minimum password length. The evaluator shall also perform the following tests. Note that one or more of these tests can be performed with a single test case.

Test 1: The evaluator shall compose passwords that either meet the requirements, or fail to meet the requirements, in some way. For each password, the evaluator shall verify that the TOE supports the password. While the evaluator is not required (nor is it feasible) to test all possible compositions of passwords, the evaluator shall ensure that all characters, rule characteristics, and a minimum length listed in the requirement are supported, and justify the subset of those characters chosen for testing.

5.2.2.3.5 Extended: Authorization Throttling (FIA_TRT_EXT.1)

The evaluator shall verify that the TSS describes the method by which authentication attempts are not able to be automated. The evaluator shall ensure that the TSS describes either how the TSF disables authentication via external interfaces (other than the ordinary user interface) or how authentication attempts are delayed in order to slow automated entry and shall ensure that this delay totals at least 500 milliseconds over 10 attempts.

5.2.2.3.6 Protected Authorization Feedback (FIA_UAU.7)

The evaluator shall ensure that the TSS describes the means of obscuring the password entry. The evaluator shall verify that any configuration of this requirement is addressed in the AGD guidance and that the password is obscured by default.

Test: The evaluator shall enter passwords on the device, including at least the Password Authentication Factor at lockscreen, and verify that the password is not displayed on the device.

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

The evaluator shall verify that the TSS section of the ST describes the process for decrypting protected data and keys. The evaluator shall ensure that this process requires the user to enter a Password Authentication Factor and, in accordance with FCS_CKM_EXT.3, derives a KEK which is used to protect the software-based secure key storage and (optionally) DEK(s) for sensitive data, in accordance with FCS_STG_EXT.2.

The following tests may be performed in conjunction with FDP_DAR_EXT.1.

Assurance Activity Note: The following test require the developer to provide access to a test platform that provides the evaluator with tools that are typically not found on consumer Mobile Device products.

Test 1: The evaluator shall enable encryption of protected data and require user authentication according to the AGD guidance. The evaluator shall write, or the developer shall provide access to, an application that includes a unique string treated as protected data.

The evaluator shall reboot the device, use a tool provided by developer to search for the unique string amongst the application data, and verify that the unique string cannot be found. The evaluator shall enter the Password Authentication Factor to access full device functionality, use a tool provided by developer to search for the unique string amongst the application data, and verify that the unique string can be found.

Test 2: [conditional] The evaluator shall require user authentication according to the AGD guidance. The evaluator shall write, or the developer shall provide access to, an application that generates and stores a key in the software-based secure key storage.

The evaluator shall lock the device, use a tool provided by developer to search for the key amongst the application data, and verify that the key cannot be found. The evaluator shall enter the Password Authentication Factor to access full device functionality, use a tool provided by developer to search for the key amongst the application data, and verify that the unique string can be found.

Test 3: [conditional] The evaluator shall enable encryption of sensitive data and require user authentication according to the AGD guidance. The evaluator shall write, or the developer shall provide access to, an application that includes a unique string treated as sensitive data (this may be data or a key).

The evaluator shall lock the device, use a tool provided by developer to search for the unique string amongst the application data, and verify that the unique string cannot be found. The evaluator shall enter the Password Authentication Factor to access full device functionality, use a tool provided by developer to search for the unique string amongst the application data, and verify that the unique string can be found.

5.2.2.3.8 Extended: Timing of Authentication (FIA_UAU_EXT.2)

The evaluator shall verify that the TSS describes the actions allowed by unauthorized users in the locked state. The evaluator shall attempt to perform some actions not listed in the selection while the device is in the locked state and verify that those actions do not succeed.

5.2.2.3.9 Extended: Re-Authorizing (FIA_UAU_EXT.3)

Test 1: The evaluator shall configure the TSF to use the Password Authentication Factor according to the AGD guidance. The evaluator shall change Password Authentication Factor according to the AGD guidance and verify that the TSF requires the entry of the Password Authentication Factor before allowing the factor to be changed.

Test 2: The evaluator shall configure the TSF to transition to the locked state after a time of inactivity (FMT_SMF.1) according to the AGD guidance. The evaluator shall wait until the TSF locks and then verify that the TSF requires the entry of the Password Authentication Factor before transitioning to the unlocked state.

Test 3: The evaluator shall configure user-initiated locking according to the AGD guidance. The evaluator shall lock the TSF and then verify that the TSF requires the entry of the Password Authentication Factor before transitioning to the unlocked state.

5.2.2.3.10 Extended: Validation of Certificates (FIA_X509_EXT.1)

The evaluator shall ensure the TSS describes where the check of validity of the certificates takes place. The evaluator ensures the TSS also provides a description of the certificate path validation algorithm.

The tests described must be performed in conjunction with the other Certificate Services assurance activities, including the use cases in FIA_X509_EXT.2.1 and FIA_X509_EXT.3. The tests for the extendedKeyUsage rules are performed in conjunction with the uses that require those rules.

Test 1: The evaluator shall demonstrate that validating a certificate without a valid certification path results in the function (application validation, trusted channel setup, or trusted software update) failing. The evaluator shall then load a certificate or certificates needed to validate the certificate to be used in the function, and demonstrate that the function succeeds. The evaluator then shall delete one of the certificates, and show that the function fails.

Test 2: The evaluator shall demonstrate that validating an expired certificate results in the function failing.

Test 3: The evaluator shall test that the TOE can properly handle revoked certificates – conditional on whether CRL or OCSP is selected; if both are selected, and then a test is performed for each method. The evaluator has to only test one up in the trust chain (future revisions may require to ensure the validation is done up the entire chain). The evaluator shall ensure that a valid certificate is used, and that the validation function succeeds. The evaluator then attempts the test with a certificate that will be revoked (for each method chosen in the selection) to ensure when the certificate is no longer valid that the validation function fails.

Test 4: The evaluator shall construct a certificate path, such that the certificate of the CA issuing the TOE's certificate does not contain the basicConstraints extension. The validation of the certificate path fails.

Test 5: The evaluator shall construct a certificate path, such that the certificate of the CA issuing the TOE's certificate has the cA flag in the basicConstraints extension not set. The validation of the certificate path fails.

Test 6: The evaluator shall construct a certificate path, such that the certificate of the CA issuing the TOE's certificate has the cA flag in the basicConstraints extension set to TRUE. The validation of the certificate path succeeds.

5.2.2.3.11 Extended: X.509 Certificate Authentication (FIA_X509_EXT.2)

The evaluator shall check the TSS to ensure that it describes how the TOE chooses which certificates to use, and any necessary instructions in the administrative guidance for configuring the operating environment so that the TOE can use the certificates.

The evaluator shall examine the TSS to confirm that it describes the behavior of the TOE when a connection cannot be established during the validity check of a certificate used in establishing a trusted channel. If the requirement that the administrator is able to specify the default action, then the evaluator shall ensure that the operational guidance contains instructions on how this configuration action is performed.

The evaluator shall perform Test 1 for each function listed in FIA_X509_EXT.2.1 that requires the use of certificates:

Test 1: The evaluator shall demonstrate that using a certificate without a valid certification path results in the function failing. Using the administrative guidance, the evaluator shall then load a certificate or certificates needed to validate the certificate to be used in the function, and demonstrate that the function succeeds. The evaluator then shall delete one of the certificates, and show that the function fails.

Test 2: The evaluator shall demonstrate that using a valid certificate that requires certificate validation checking to be performed in at least some part by communicating with a non-TOE IT entity. The evaluator shall then manipulate the environment so that the TOE is unable to verify the validity of the certificate, and observe that the action selected in FIA_X509_EXT.2.2 is performed. If the selected action is administrator-configurable, then the evaluator shall follow the operational guidance to determine that all supported administrator-configurable options behave in their documented manner.

The assurance activity for **FIA_X509_EXT.2.4** ~~this requirement~~ is performed in conjunction with the assurance activity for FIA_X509_EXT.2.1 and FIA_X509_EXT.2.2.

For FIA_X509_EXT.2.5 the evaluator shall check to ensure that the operational guidance contains instructions on generating a Certificate Request Message. The evaluator shall also perform the following test.

Test 1: The evaluator shall use the operational guidance to cause the TOE to generate a certificate request message. The evaluator shall confirm that they are able to provide the public key, Common Name, Organization, Organizational Unit, and Country as input into this request. The evaluator shall capture the generated message and ensure that it conforms with the format specified by RFC 2986.

5.2.2.3.12 Extended: Request Validation of Certificates (FIA_X509_EXT.3)

The evaluator shall verify that the API documentation provided according to Section 6.2.1 includes the security function (certificate validation) described in this requirement. This documentation shall be clear as to which results indicate success and failure.

The evaluator shall write, or the developer shall provide access to, an application that requests certificate validation by the TSF. The evaluator shall verify that the results from the validation match the expected results according to the API documentation. This application may be used to verify that import, removal, modification, and validation are performed correctly according to the tests required by FDP_STG_EXT.1, FDP_ITC_EXT.1, FMT_SMF.1.1 function 14, and FIA_X509_EXT.1.

5.2.2.4 Security Management

5.2.2.4.1 Management of Security Functions Behavior by the User (FMT_MOF.1(USER))

The evaluation shall verify that the TSS describes those management functions which may only be performed by the user in conjunction with the TSS description for FMT_SMF.1.

5.2.2.4.2 Management of Security Functions Behavior by the Organization (FMT_MOF.1(ORG))

Test 1: The evaluator shall use the test environment to deploy policies to mobile devices.

Test 2: The evaluator shall create policies which collectively include all management functions which are controlled by the (enterprise) administrator and cannot be overridden by the user as defined in FMT_MOF.1.1(2). The evaluator shall apply these policies to devices, attempt to override each setting as the user, and ensure that the TSF does not permit it.

5.2.2.4.3 Specifications of Management Functions (FMT_SMF.1)

The following activities shall take place in the test environment described in the Assurance Activity for FPT_TUD_EXT.1.1, FPT_TUD_EXT.1.2, FPT_TUD_EXT.1.3, and FPT_TUD_EXT.1.4. The evaluator shall consult the AGD guidance to perform each of the following tests, iterating each test as necessary if both the user and administrator may perform the function. The following test numbers correspond to the function numbers.

Test 1: The evaluator shall exercise the TSF configuration as the administrator and perform positive and negative tests, with at least two assignments for each variable setting, for each of the following:

- minimum password length
- minimum password complexity
- maximum password lifetime

Test 2: The evaluator shall exercise the TSF configuration as the user and the administrator. The evaluator shall perform positive and negative tests, with at least two assignments for each variable setting, for each of the following.

- screen-lock enabled/disabled
- screen lock timeout
- number of authentication failures (may be combined with test for FIA_AFL.1)

Test 3: The evaluator shall exercise the TSF configuration to enable the VPN protection. These configuration actions must be used for the testing of the FDP_IFC.1.1 requirement.

Test 4: The evaluator shall exercise the TSF configuration as both the user and administrator to enable and disable the state of each radio (e.g. Wi-Fi, GPS, cellular, NFC, Bluetooth) listed by the ST author. For each radio, the evaluator shall use a spectrum analyzer and a RF- shielded environment to verify the existence of signals when the radio is enabled and the absence of signals when the radio is disabled. The evaluator shall verify the absence of signals during device reboot and casual usage.

Test 5: The evaluator shall exercise the TSF configuration as both the user and administrator to enable and disable the state of each audio or visual collection devices (e.g. camera, microphone) listed by the ST author. For each collection device, the evaluator shall disable the device and then attempt to use its functionality.

Test 6: The evaluator shall create a test environment consisting of a wireless access system and an authentication server for the purpose of tests 6 and 7. The evaluator shall specify the wireless network and wireless network settings according to the AGD guidance both as an administrator and as a user. The evaluator shall specify a value for each management function according to the configuration of the test network. Minimally, the evaluator shall test a WPA2 Enterprise network using EAP-TLS. The evaluator shall verify that the TSF can establish a connection to the network.

Test 7: The evaluator shall specify a wireless network with an incorrect value for WLAN authentication server and verify that the mobile device cannot connect to the WLAN. The evaluator shall repeat this test, setting incorrect values for the security type and authentication protocol individually and verify that the mobile device cannot connect to the WLAN.

Test 8 & 9: The evaluator shall use the test environment to instruct the TSF, as the administrator, to command the device to:

- transition to a locked state
- perform a wipe of all data

The evaluator must ensure that the device transitions to the locked state upon command. The evaluator must ensure that this management setup is used when conducting the assurance activities in FCS_CKM_EXT.5.

Test 10: The evaluator shall exercise the TSF configuration as the administrator to restrict particular applications, sources of applications, or application installation according to the AGD guidance. The evaluator shall attempt to install denied applications and ensure that this is not possible.

Test 11 & 12: The test of these functions is performed in association with FCS_STG_EXT.1.

Test 13: The evaluator shall review the AGD guidance to determine that it describes the steps needed to import, modify, or remove certificates in the Trust Anchor database. The evaluator shall import certificates according to the AGD guidance as the user or as the administrator. The evaluator shall verify that no errors occur during import.

Test 14: The evaluator shall remove an administrator-imported certificate and any other categories of certificates included in the assignment of function 15 from the Trust Anchor Database according to the AGD guidance as the user and as the administrator.

Test 15: The evaluator shall verify that user approval is required to enroll the device into management and includes a description of each type of management function that will be enforced.

Test 16: The evaluator shall attempt to remove applications according to the AGD guidance and verify that the TOE no longer permits users to access those applications or their associated data.

Test 17 & 18: The evaluator shall attempt to update the TSF system software (if updates are available) and install mobile applications and verify that updates correctly install and that the version numbers of the system software and of the mobile applications increase.

Test 19: [*conditional*] The evaluator shall exercise the TSF configuration to enable and disable data transfer capabilities over each externally accessible hardware ports (e.g. USB, SD card, HDMI) listed by the ST author. The evaluator shall use test equipment for the particular interface to ensure that no low-level signalling is occurring on all pins used for data transfer when they are disabled.

Test 20: [*conditional*] The evaluator shall attempt to disable each listed protocol in the assignment, which should include tethering uses. The evaluator shall verify that remote devices can no longer access the TOE or TOE resources using any disabled protocols.

Test 21: [*conditional*] The evaluator shall exercise the TSF configuration as both the user and administrator to enable and disable any developer mode. The evaluator shall test that developer mode access is not available when its configuration is disabled. The evaluator shall verify the developer mode remains disabled during device reboot.

Test 22, 23, & 24: [*conditional*] The evaluator shall exercise the TSF configuration as both the user and administrator to enable system-wide data-at-rest protection according to the AGD guidance. The evaluator shall ensure that all assurance activities for DAR (see Section 5.3.2) are conducted with the device in this configuration. The evaluator shall disable any “Forgot Password” feature and ensure that the device does not offer any password hints.

Test 25: [*conditional*]: The evaluator shall establish an APN for the test network, configure the private APN onto the device. The evaluator shall then send packets to the publically routable Internet (perhaps using a tool provided by the developer). The evaluator shall observe that these packets are reaching the APN termination point and not arriving via the carrier's internet access gateway. The evaluator shall repeat the test with a different or invalid APN on the device, and verify that the packets do not reach the APN termination point.

Test 26: [*conditional*] The evaluator shall disable the Discoverable mode and verify that no new Bluetooth peripherals can connect to the device. The evaluator shall disallow each Bluetooth version and attempt to connect a Bluetooth peripheral to the device. The evaluator shall verify with a Bluetooth protocol analysis tool that the TOE does not perform disabled versions or list the disabled versions as supported by the TOE during pairing negotiations with a Bluetooth peripheral. The evaluator shall, according to the selection, restrict which pairing mechanisms are allowed by the TOE (via Bluetooth profiles or particular pairing protocols). The evaluator shall verify with a Bluetooth protocol analysis tool that the TOE does not perform disabled pairing mechanism or list the disabled mechanism as supported by the TOE during pairing negotiations with a Bluetooth peripheral.

Test 27: [*conditional*] For each category of information listed in the AGD guidance, the evaluator shall verify that when that TSF is configured to limit the information according to the AGD, the information is no longer displayed in the locked state.

Test 28: [*conditional*] The evaluator shall attempt to wipe sensitive data resident on the device according to the administrator guidance. The evaluator shall verify that the data is no longer accessible by the user.

Test 29: [*conditional*] The evaluator shall configure the device to alert the administrator according to the administrator guidance (for example, by configuring a trigger that causes an alert to the MDM). The evaluator shall verify that the administrator receives an alert for the device.

Test 30: [*conditional*] The evaluator shall attempt to remove any Enterprise applications from the device by following the administrator guidance. The evaluator shall verify that the TOE no longer permits users to access those applications or their associated data.

Test 31: [*conditional*] The evaluator shall also verify that the API documentation provided according to Section 6.2.1[32] includes any security functions (import, modification, or destruction of the Trust Anchor Database) allowed by applications.

If applications may import certificates to the Trust Anchor Database. The evaluator shall write, or the developer shall provide access to, an application that imports a certificate into the Trust Anchor Database. The evaluator shall verify that the TOE requires approval before allowing the application to import the certificate:

[32] This is the plaintext storage requirement (**FPT_KST_EXT.1**).

- The evaluator shall deny the approvals to verify that the application is not able to import the certificate. Failure of import shall be tested by attempting to validate a certificate that chains to the certificate whose import was attempted (as described in the Assurance Activity for FIA_X509_EXT.1).
- The evaluator shall repeat the test, allowing the approval to verify that the application is able to import the certificate and that validation occurs.

If applications may remove certificates in the Trust Anchor Database, the evaluator shall write, or the developer shall provide access to, an application that removes certificates from the Trust Anchor Database. The evaluator shall verify that the TOE requires approval before allowing the application to remove the certificate:

- The evaluator shall deny the approvals to verify that the application is not able to remove the certificate. Failure of removal shall be tested by attempting to validate a certificate that chains to the certificate whose removal was attempted (as described in the Assurance Activity for FIA_X509_EXT.1).

The evaluator shall repeat the test, allowing the approval to verify that the application is able to remove/modify the certificate and that validation no longer occurs.

Test 32: [*conditional*] The test of this function is performed in conjunction with FIA_X509_EXT.2.2.

Test 33: [*conditional*] The evaluator shall attempt to disable all cellular voice functionality according to the administrator guidance. The evaluator shall then attempt to place a call on the TOE as the user and verify that the function fails. The evaluator shall also attempt to call the TOE and verify that the call cannot be completed.

Test 34: [*conditional*] The evaluator shall attempt to disable all device messaging functionality according to the administrator guidance. The evaluator shall then attempt to send a message on the TOE as the user and verify that the function fails. The evaluator shall also attempt to send a message to the TOE and verify that the message is not received.

Test 35: [*conditional*] The evaluator shall attempt to disable each cellular protocol according to the administrator guidance. The evaluator shall attempt to connect the device to a cellular network and, using network analysis tools, verify that the device does not allow negotiation of the disabled protocols.

Test 36: [*conditional*] The evaluator shall attempt to disable voice control functionality and shall verify that the TOE no longer performs any actions upon being given a voice command.

Test 37: [*conditional*] The evaluator shall attempt to read any device audit logs according to the administrator guidance and verify that the logs may be read. This test may be performed in conjunction with the assurance activity of FAU_GEN.1.

Test 38: [*conditional*] The test of this function is performed in conjunction with FPT_TUD_EXT.2.5.

Test 39 & 40: [*conditional*] The test of these functions is performed in conjunction with FCS_STG_EXT.1.

Test 41: [*conditional*] The test of this function is performed in conjunction with FTA_TAB.1.

5.2.2.4.4 Extended: Specification of Remediation Actions (FMT_SMF_EXT.1)

The evaluator shall use the test environment to iteratively configure the device to perform each remediation action in the selection upon unenrollment. The evaluator shall unenroll the device according to AGD guidance and verify that the remediation action configured is performed.

5.2.2.5 Protection of the TSF

5.2.2.5.1 Extended: Anti-Exploitation Services for Address Space Layout Randomization (FPT_AEX_EXT.1)

The evaluator shall ensure that the TSS section of the ST describes how the 8 bits are generated and provides a justification as to why those bits are unpredictable.

Assurance Activity Note: The following test require the developer to provide access to a test platform that provides the evaluator with tools that are typically not found on consumer Mobile Device products.

Test 1: The evaluator shall select 3 apps included with the TSF. These must include any web browser or mail client included with the TSF. For each of these apps, the evaluator will launch the same app on two separate mobile devices of the same type and compare all memory mapping locations. The evaluator must ensure that no memory mappings are placed in the same location on both devices.

If the rare (at most 1/256) chance occurs that two mappings are the same for a single app and not the same for the other two apps, the evaluator shall repeat the test with that app to verify that in the second test the mappings are different.

For FPT_AEX_EXT.1.3 and FPT_AEX_EXT.1.3, the evaluator shall ensure that the TSS section of the ST describes how the 4 bits are generated and provides a justification as to why those bits are unpredictable.

Assurance Activity Note: *The following test require the developer to provide access to a test platform that provides the evaluator with tools that are typically not found on consumer Mobile Device products.*

Test 1: The evaluator shall reboot the TOE at least five times. For each of these reboots, the evaluator shall examine memory mapping locations of the kernel. The evaluator must ensure that no memory mappings are placed in the same location on both devices.

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

The evaluator shall ensure that the TSS describes of the memory management unit (MMU), and ensures that this description documents the ability of the MMU to enforce read, write, and execute permissions on all pages of virtual memory.

For FPT_AEX_EXT.2.2, the evaluator shall ensure that the TSS describes of the memory management unit (MMU), and ensures that this description documents the ability of the MMU to enforce write XOR execute permissions.

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

The evaluator shall determine that the TSS contains a description of stack-based buffer overflow protections implemented in the TSF software which runs in the non-privileged execution mode of the application processor. The exact implementation of stack-based buffer overflow protection will vary by platform. Example implementations may be activated through compiler options such as "-fstack-protector-all", “-fstack-protector”, and ”GS” flags. The evaluator shall ensure that the TSS contains an inventory of TSF binaries and libraries, indicating those that implement stack-based buffer overflow protections as well as those that do not. The TSS must provide a rationale for those binaries and libraries that are not protected in this manner.

5.2.2.5.4 Extended: Domain Isolation (FPT_AEX_EXT.4)

The evaluator shall ensure that the TSS describes the mechanisms that are in place that prevents non-TSF software from modifying the TSF software or TSF data that governs the behavior of the TSF. These mechanisms could range from hardware-based means (e.g. “execution rings” and memory management functionality); to software-based means (e.g. boundary checking of inputs to APIs). The evaluator determines that the described mechanisms appear reasonable to protect the TSF from modification.

The evaluator shall ensure the TSS describes how the TSF ensures that the address spaces of applications are kept separate from one another.

Assurance Activity Note: The following tests require the vendor to provide access to a test platform that provides the evaluator with tools that are typically not found on consumer Mobile Device products. In addition, the vendor provides a list of files (e.g., system files, libraries, configuration files) that make up the TSF. This list could be organized by folders/directories (e.g., /usr/sbin, /etc), as well as individual files that may exist outside of the identified directories.

Test 1: The evaluator shall check the “permission settings” for each file in vendor provided list of files that make up the TSF and ensure the settings are appropriate for preventing writing by untrusted applications. The evaluator shall attempt to modify a file of their choosing to ensure the mechanism enforces the permission settings and prevents modification.

Test 2: The evaluator shall create and load an app onto the mobile device. This app shall attempt to traverse over all file systems and report any locations to which data can be written or overwritten. The evaluator must ensure that none of these locations are part of the OS software, device drivers, system and security configuration files, key material, or another application‘s image/data.

5.2.2.5.5 Extended: Plaintext Key Storage (FPT_KST_EXT.1)

The evaluator shall consult the TSS section of the ST in performing the assurance activities for this requirement.

In performing their review, the evaluator shall determine that the TSS contains a description of the activities that happen on power-up and password authentication relating to the decryption of DEKs, stored keys, and data.

The evaluator shall ensure that the description also covers how the cryptographic functions in the FCS requirements are being used to perform the encryption functions, including how the KEKs, DEKs, and stored keys are unwrapped, saved, and used by the TOE so as to prevent plaintext from being written to non-volatile storage. The evaluator shall ensure that the TSS describes, for each power-down scenario how the TOE ensures that all keys in non-volatile storage are wrapped with a KEK.

The evaluator shall ensure that the TSS describes how other functions available in the system (e.g., regeneration of the keys) ensure that no unencrypted key material is present in persistent storage.

The evaluator shall review the TSS to determine that it makes a case that key material is not written unencrypted to the persistent storage.

5.2.2.5.6 Extended: No Key Transmission (FPT_KST_EXT.2)

The evaluator shall consult the TSS section of the ST in performing the assurance activities for this requirement. The evaluator shall ensure that the TSS describes the cryptographic module boundary. The cryptographic module may very well be a particular kernel module, the Operating System, the Application Processor, or up to the entire Mobile Device.

In performing their review, the evaluator shall determine that the TSS contains a description of the activities that happen on power-up and password authentication relating to the decryption of DEKs, stored keys, and data.

The evaluator shall ensure that the TSS describes how other functions available in the system (e.g., regeneration of the keys) ensure that no unencrypted key material is transmitted outside the cryptographic module.

The evaluator shall review the TSS to determine that it makes a case that key material is not transmitted outside the cryptographic module.

5.2.2.5.7 Extended: No Plaintext Key Transport (FPT_KST_EXT.3)

The ST author will provide a statement of their policy for handling and protecting keys. The evaluator shall check to ensure the TSS describes a policy in line with not exporting either plaintext DEKs, KEKs, or keys stored in the secure key storage.

5.2.2.5.8 Extended: Self-Test Event Notification (FPT_NOT_EXT.1)

The evaluator shall verify that the TSS describes critical failures that may occur and the actions to be taken upon these critical failures.

Assurance Activity Note: The following test require the developer to provide access to a test platform that provides the evaluator with tools that are typically not found on consumer Mobile Device products.

Test 1: The evaluator shall use a tool provided by the developer to modify files and processes in the system that correspond to critical failures specified in the second list. The evaluator shall verify that creating these critical failures causes the device to take the remediation actions specified in the first list.

5.2.2.5.9 Reliable Time Stamps (FPT_STM.1)

The evaluator shall examine the TSS to ensure that it lists each security function that makes use of time. The TSS provides a description of how the time is maintained and considered reliable in the context of each of the time related functions. This documentation must identify whether the TSF uses GPS, a NTP server, or the carrier‘s network time as the primary time sources and whether any or all of these sources is configurable.

The evaluator examines the operational guidance to ensure it instructs the administrator how to set the time. If the TOE supports the use of an NTP server, the operational guidance instructs how a communication path is established between the TOE and the NTP server, and any configuration of the NTP client on the TOE to support this communication.

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

The evaluator shall examine the TSS to ensure that it specifies the self-tests that are performed at start-up. This description must include an outline of the test procedures conducted by the TSF (e.g., rather than saying "memory is tested", a description similar to "memory is tested by writing a value to each memory location and reading it back to ensure it is identical to what was written" shall be used). The TSS must include any error states that they TSF may enter when self tests fail, and the conditions and actions necessary to exit the error states and resume normal operation. The evaluator shall verify that the TSS indicates these self-tests are run at start-up automatically, and do not involve any inputs from or actions by the user or operator.

The evaluator shall inspect the list of self-tests in the TSS and verify that it includes algorithm self tests. The algorithm self tests will typically be conducted using known answer tests.

5.2.2.5.11 Extended: TSF Integrity Testing (FPT_TST_EXT.2)

The evaluator shall verify that the TSS section of the ST includes a description of the boot procedures of the software for the TSF‘s Application Processor. The evaluator shall ensure that before loading the bootloader for the operating system and the kernel, the bootloader and kernel software is cryptographically verified. The evaluator shall verify that the TSS contains a justification for the protection of the cryptographic key or hash, preventing it from being modified by unverified or unauthenticated software. The evaluator shall verify that the TSS contains a description of the protection afforded to the mechanism performing the cryptographic verification.

For FPT_TST_EXT.2.2, the evaluator shall verify that the TSS section of the ST includes a description of the boot procedures of the software for the TSF‘s application processor and baseband processor. The evaluator shall ensure that before loading any executable code, that code is cryptographically verified. The evaluator shall verify that the TSS contains a justification for the protection of the cryptographic keys or hashes, preventing them from being modified by unverified or unauthenticated software.

5.2.2.5.12 Extended: Trusted Update: TSF Version Query (FPT_TUD_EXT.1)

The evaluator shall establish a test environment consisting of the mobile device and any supporting software that demonstrates usage of the management functions. This can be test software from the developer, a reference implementation of management software from the developer, or other commercially available software. The evaluator shall set up the mobile device and the other software to exercise the management functions according to provided guidance documentation.

Test 1: Using the AGD guidance provided, the evaluator shall test that the administrator and user can query:

- the current version of the TSF operating system and any firmware that can be updated separately
- the hardware model of the TSF
- the current version of all installed mobile applications

The evaluator must review manufacturer documentation to ensure that the hardware model identifier is sufficient to identify the hardware which comprises the device.

5.2.2.5.13 Extended: Trusted Update Verification (FPT_TUD_EXT.2)

The evaluator shall verify that the TSS section of the ST describes the TSF software update mechanism for updating the system software. The evaluator shall verify that the description includes a digital signature verification of the software before installation and that installation fails if the verification fails. The evaluator shall verify that the TSS describes the method by which the digital signature is verified and that the public key used to verify the signature is either hardware-protected or is validated to chain to a public key in the Trust Anchor Database. If hardware-protection is selected, the evaluator shall verify that the method of hardware-protection is described and that the ST author has justified why the public key may not be modified by unauthorized parties.

[*conditional*] If the ST author indicates that the public key for software update digital signature verification, the evaluator shall verify that the update mechanism includes a certificate validation according to FIA_X509_EXT.1 and a check for the Code Signing purpose in the extendedKeyUsage.

The evaluator shall verify that the ST author has provided evidence that the following tests were performed:

Test 1: The tester shall try to install an update without the digital signature and shall verify that installation fails. The tester shall attempt to install an update with digital signature, and verify that installation succeeds.

Test 2: The tester shall digitally sign the update with a key disallowed by the device and verify that installation fails. The tester shall digitally sign the update with the allowed key and verify that installation succeeds.

Test 3: [conditional] The tester shall digitally sign the update with an invalid certificate and verify that update installation fails. The tester shall digitally sign the application with a certificate that does not have the Code Signing purpose and verify that application installation fails.

The evaluator shall verify that the TSS describes how mobile application software is verified at installation. The evaluator shall ensure that this method uses a digital signature.

Test 1: The evaluator shall write, or the developer shall provide access to, an application. The evaluator shall try to install this application without a digitally signature and shall verify that installation fails. The evaluator shall attempt to install a digitally signed application, and verify that installation succeeds.

For FPT_TUD_EXT.2.5, the evaluator shall verify that the TSS describes how mobile application software is verified at installation. The evaluator shall ensure that this method uses a digital signature by a code signing certificate.

Test 1: The evaluator shall write, or the developer shall provide access to, an application. The evaluator shall try to install this application without a digitally signature and shall verify that installation fails. The evaluator shall attempt to install an application digitally signed with an appropriate certificate, and verify that installation succeeds.

Test 2: The evaluator shall digitally sign the application with an invalid certificate and verify that application installation fails. The evaluator shall digitally sign the application with a certificate that does not have the Code Signing purpose and verify that application installation fails. This test may be performed in conjunction with the assurance activities for FIA_X509_EXT.1.

Test 3: The evaluator shall configure the device to limit the public keys that can sign application software according to the AGD guidance. The evaluator shall digitally sign the application with a certificate disallowed by the device or configuration and verify that application installation fails. The evaluator shall attempt to install an application digitally signed with an authorized certificate and verify that application installation succeeds.

For FPT_TUD_EXT.2.6, the evaluator shall verify that the TSS describes the mechanism that prevents the TSF from installing software updates that are an older version that the currently installed version.

Test 1: The evaluator shall attempt to install an earlier version of software and shall verify that the update fails.

Test 2: The evaluator shall attempt to install a current or later version and shall verify that the update succeeds.

5.2.2.6 TOE Access

5.2.2.6.1 Extended: TSF- and User-initiated Locked State (FTA_SSL_EXT.1)

The evaluator shall verify the TSS describes the actions performed upon transitioning to the locked state . The evaluation shall verify that the AGD guidance describes the method of setting the inactivity interval

and of commanding a lock. The evaluator shall verify that the TSS describes the information allowed to be displayed to unauthorized users.

Test 1: The evaluator shall configure the TSF to transition to the locked state after a time of inactivity (FMT_SMF.1) according to the AGD guidance. The evaluator shall wait until the TSF locks and verify that the display is cleared or overwritten and that the only actions allowed in the locked state are unlocking the session and those actions specified in FIA_UAU_EXT.2.

Test 2: The evaluator shall command the TSF to transition to the locked state according to the AGD guidance as both the user and the administrator. The evaluator shall wait until the TSF locks and verify that the display is cleared or overwritten and that the only actions allowed in the locked state are unlocking the session and those actions specified in FIA_UAU_EXT.2.

5.2.2.6.2 Extended: Wireless Network Access (FTA_WSE_EXT.1)

The assurance activity for this requirement is performed in conjunction with the assurance activity for FMT_SMF.1.

5.2.2.6.3 Default TOE Access Banners (FTA_TAB.1)

The TSS shall describe when the banner is displayed. The evaluator shall also perform the following test:

Test 1: The evaluator follows the operational guidance to configure a notice and consent warning message. The evaluator shall then start up or unlock the TSF. The evaluator shall verify that the notice and consent warning message is displayed in each instance described in the TSS.

5.2.2.7 Trusted Path/Channels

5.2.2.7.1 Extended: Trusted Channel Communication (FTP_ITC_EXT.1)

The evaluator shall verify that the API documentation provided according to Section 6.2.1 includes the security functions (trusted channel) described in these requirements. The evaluator shall write, or the developer shall provide access to, an application that requests trusted channel services by the TSF. The evaluator shall verify that the results from the trusted channel match the expected results according to the API documentation. This application may be used to assist in verifying the trusted channel assurance activities for the protocol requirements.

The evaluator shall examine the TSS to determine that it describes the details of the TOE connecting to an access point in terms of the cryptographic protocols specified in the requirement, along with TOE-specific options or procedures that might not be reflected in the specification. The evaluator shall also confirm that all protocols listed in the TSS are specified and included in the requirements in the ST. The evaluator shall confirm that the operational guidance contains instructions for establishing the connection to the access point. The evaluator shall also perform the following tests:

Test 1: The evaluators shall ensure that the TOE is able to initiate communications with an access point using the protocols specified in the requirement, setting up the connections as described in the operational guidance and ensuring that communication is successful.

Test 2: The evaluator shall ensure, for each communication channel with an authorized IT entity, the channel data is not sent in plaintext.

6 TOE Summary Specification (TSS)

This chapter describes the Windows 8.1 and Windows Phone security functions. The Security Functions (SFs) satisfy the security functional requirements of the Mobile Device Fundamentals protection profile. The TOE also includes additional relevant security functions which are also described in the following sections, as well as a mapping to the security functional requirements satisfied by the TOE.

Unless otherwise noted in this section, all statements apply to Windows 8.1 and Windows Phone.

6.1 Product Architecture

6.2 TOE Security Functions

This section presents the TOE Security Functions (TSFs) and a mapping of security functions to Security Functional Requirements (SFRs). The TOE performs the following security functions:

- Cryptographic Support
- User Data Protection
- Identification and Authentication
- Security Management
- Protection of the TSF
- TOE Access
- Trusted Path / Channels

6.2.1 Cryptographic Support

6.2.1.1 Cryptographic Algorithms and Operations

Cryptography API: Next Generation (CNG) API is designed to be extensible at many levels and agnostic to cryptographic algorithm suites. An important feature of CNG is its native implementation of the Suite B algorithms, including algorithms for AES (128, 192, 256 key sizes), the SHA-1 and SHA-2 family (SHA-256, SHA-384 and SHA-512) of hashing algorithms, elliptic curve Diffie Hellman (ECDH), and elliptical curve DSA (ECDSA) over the NIST-standard prime curves P-256, P-384, and P-521.

Protocols such as the Internet Key Exchange (IKE), and Transport Layer Security (TLS), make use of elliptic curve Diffie-Hellman (ECDH) included in Suite B as well as hashing functions.

Deterministic random bit generation (DRBG) is implemented in accordance with NIST Special Publication 800-90. Windows generates random bits by taking the output of a cascade of two SP800-90 AES-256 counter mode based DRBGs in kernel-mode and four cascaded SP800-90 AES-256 DRBGs in user-mode; programmatic callers can choose to obtain either 128 or 256 bits from the RBG which is seeded from the Windows entropy pool. The entropy pool is populated using the following values:

- An initial entropy value from a seed file provided to the Windows OS Loader at boot time (512 bits of entropy). [33]
- A calculated value based on the high-resolution CPU cycle counter which fires after every 1024 interrupts (a continuous source providing 16384 bits of entropy).
- Random values gathered periodically from the Trusted Platform Module (TPM), if one is available on the system (320 bits of entropy on boot, 384 bits thereafter).
- Random values gathered periodically by calling the RDRAND CPU instruction, if supported by the CPU (256 bits of entropy).

The main source of entropy in the system is the CPU cycle counter which tracks hardware interrupts. This is a sufficient health test; if the computer were not accumulating hardware and software interrupts it would not be running and therefore there would be no need for random bit generation. In the same manner, a failure of the TPM chip or processor would be a critical error that halts the computer. In addition, when the user chooses to follow the CC administrative guidance, which includes operating Windows in the FIPS validated mode, it will run FIPS 140 AES-256 Counter Mode DBRG Known Answer Tests (instantiate, generate) and Dual-EC DRBG Known Answer Tests (instantiate, generate) on start-up. Windows always runs the SP 800-90-mandated self-tests for AES-CTR-DRBG during a reseed and runs the Dual-EC reseed self-test when the user chooses to operate Windows in the FIPS validated mode. [34]

Each entropy source is independent of the other sources and does not depend on time. The CPU cycle counter inputs vary by environmental conditions such as data received on a network interface card, key presses on a keyboard, mouse movement and clicks, and touch input.

The TSF defends against tampering of the random number generation (RNG) / pseudorandom number generation (PRNG) sources by encapsulating its use in Kernel Security Device Driver. The interface for the Windows random number generator is BCryptGenRandom.

By default, the CNG provider for random number generation is the AES_CTR_DRBG, however CNG can be configured to use the Dual EC DRBG, which is no longer a FIPS approved algorithm. When Windows requires the use of a salt it uses the Windows RBG.

The encryption and decryption operations are performed by independent modules, known as Cryptographic Service Providers (CSPs) which are FIPS 140-2 Level 1 compliant. Windows generates symmetric keys (AES keys) using the FIPS Approved random number generator.

In addition to encryption and decryption services, the TSF provides other cryptographic operations such as hashing and digital signatures. Hashing is used by other FIPS Approved algorithms implemented in Windows (the hashed message authentication code, RSA, DSA, and EC DSA signature services, Diffie-Hellman and elliptic curve Diffie-Hellman key agreement, and the Dual EC random bit generator).

[33] The Windows OS Loader implements a SP 800-90 AES-CTR-DRBG and passes along 384 bits of entropy to the kernel for CNG to be use during initialization. This DBRG uses the same algorithms to obtain entropy from the CPU cycle counter, TPM, and RDRAND as described above.
[34] Running Windows in FIPS validated mode is required according to the administrative guidance.

The hash-based message authentication code functions (HMAC) are based on SHA-1, SHA-256, SHA-384, and SHA-512, have the following characteristics:

Table 6-1 HMAC Characteristics

HMAC Algorithm	Hash function Used	Block Size	Output MAC Length	Key Length / Key Size
HMAC-SHA-1	SHA-1	512 bits	20 bytes	The key size is 10-63 bytes when the key size is less than the block size and the key size is 65 to 1024 bytes when the key size is greater than the block size. The key size may also equal the block size. The key size is variable.
HMAC-SHA-256	SHA-256	512 bits	32 bytes	Same as HMAC-SHA-1
HMAC-SHA-384	SHA-384	1024 bits	48 bytes	The key size is 24-127 bytes when the key size is less than the block size and the key size is 129-1024 bytes when the key size is greater than the block size. The key size may also equal the block size. The key size is variable.
HMAC-SHA-512	SHA-512	1024 bits	64 bytes	The key size is 32-127 bytes when the key size is less than the block size and the key size is 129-1024 bytes when the key size is greater than the block size. The key size may also equal the block size. The key size is variable.

The HMAC function forms the basis for a FIPS Approved implementation of a password based key derivation function (PBKDF). Windows inputs the password as a text string without any optional padding or blocking into a HMAC 512 function. The hash functions supported by the Windows implementation of SP 800-132 are SHA-1, SHA-256, SHA-384 or SHA-512. The SHA-512 function is used by DPAPI (see **Protecting Data with DPAPI**).

Table 6-2 Cryptographic Algorithm Standards and Evaluation Methods

Cryptographic Operation	Standard	Evaluation Method
Encryption/Decryption	FIPS 197 AES For ECB, CBC, CFB8, CCM, and GCM modes	NIST CAVP #2848, #2832, #2853
Digital signature	FIPS 186-4 rDSA	NIST CAVP #1487, #1493, #1494, #1519
Digital signature	FIPS 186-4 DSA	NIST CAVP #855
Digital signature	FIPS 186-4 ECDSA	NIST CAVP #505
Hashing	FIPS 180-3 SHA-2	NIST CAVP #2373, #2396
Keyed-Hash Message Authentication Code	FIPS 198-2 HMAC	NIST CAVP #1773
Random number generation	NIST SP 800-90 CTR_DRBG	NIST CAVP #489 for CTR_DRBG

Key agreement	NIST SP 800-56A ECDH	NIST CAVP #47
IKEv1	SP800-135	NIST CVL #323
IKEv2	SP800-135	NIST CVL #323
TLS	SP800-135	NIST CVL #323

The TSF includes a key isolation service designed specifically to host secret and private keys in a protected process to mitigate tampering or access to sensitive key materials. The TSF performs a key error detection check on each transfer of key (internal and intermediate transfers). The TSF prevents archiving of expired (private) signature keys. The TSF destroys non-persistent cryptographic keys. The TSF overwrites each intermediate storage area for plaintext key/critical cryptographic security parameter (i.e., any storage, such as memory buffers, that is included in the path of such data). This overwriting is performed as follows:

- For non-volatile memories other than EEPROM and Flash, the overwrite is executed three or more times using a different alternating data pattern each time upon the transfer of the key/critical cryptographic security parameter to another location.
- For volatile memory and non-volatile EEPROM and Flash memories, the overwrite is a single direct overwrite consisting of a pseudo random pattern, followed by a read-verify upon the transfer of the key/critical cryptographic security parameter to another location.

Windows uses FIPS Approved algorithms to establish Wi-Fi sessions and can be configured to use TLS and IPsec ciphersuites that solely use FIPS Approved algorithm primitives. The following table describes the keys and secrets used for IPsec, TLS, and Wi-Fi; when these ephemeral keys or secrets are no longer needed for a network session, they are deleted as described above and in section **5.1.1.9**.

Table 6-3 Keys Used for IPsec, TLS, and Wi-Fi

Key	Description
Symmetric encryption/decryption keys	Keys used for AES (FIPS 197) encryption/decryption for IPsec ESP, TLS, Wi-Fi.
HMAC keys	Keys used for HMAC-SHA1, HMAC-SHA256, HMAC-SHA384, and HMAC-SHA512 (FIPS 198-1) as part of IPsec
Asymmetric ECDSA Public Keys	Keys used for the verification of ECDSA digital signatures (FIPS 186-4) for IPsec traffic and peer authentication.
Asymmetric ECDSA Private Keys	Keys used for the calculation of ECDSA digital signatures (FIPS 186-4) for IPsec traffic and peer authentication.
Asymmetric RSA Public Keys	Keys used for the verification of RSA digital signatures (FIPS 186-4) for IPsec, TLS, Wi-Fi and signed product updates.
Asymmetric RSA Private Keys	Keys used for the calculation of RSA digital signatures (FIPS 186-4) for IPsec, TLS, and Wi-Fi.
DH Private and Public values	Private and public values used for Diffie-Hellman key establishment for TLS.
ECDH Private and Public values	Private and public values used for EC Diffie-Hellman key

establishment for TLS.

6.2.1.2 Programming Interfaces

Modern Store Applications can use these interfaces to obtain random bits from the OS:

- CryptographicBuffer.GenerateRandom
- CryptographicBuffer.GenerateRandomNumber

And can use these interfaces to obtain other cryptographic services from the OS:

- CryptographicEngine.Encrypt
- CryptographicEngine.Decrypt
- HashAlgorithmProvider.CreateHash
- HashAlgorithmProvider.HashData
- CryptographicEngine.Sign
- CryptographicEngine.VerifySignature
- KeyDerivationParameters.BuildForPbkdf2
- AsymmetricKeyAlgorithmProvider.CreateKeyPair
- CryptographicEngine.Sign
- CryptographicEngine.SignAsync
- CryptographicEngine.SignHashedData
- CryptographicEngine.SignHashedDataAsync
- CryptographicEngine.VerifySignature
- CryptographicEngine.VerifySignatureWithHashInput
- CryptographicEngine.Encrypt
- CryptographicEngine.Decrypt

6.2.1.3 Trusted Platform Module

Computers that incorporate a TPM have the ability to both create cryptographic keys within the TPM and protect data stored outside TPM so that the data can be decrypted only by the TPM internal keys. This process, often called "sealing" or "binding", can help protect the data from disclosure, but more importantly associates the key with the TPM. Each TPM contains a master "sealing" key, called the Storage Root Key (SRK), which was generated by the Storage Primary Seed (SPS). Like other cryptographic data within the TPM, the private portion of a key created in a TPM is never exposed to any other component, software, process, or user.

A TPM 2.0 protection profile written by the Trusted Computing Group provides additional detail about the SPS and the SRK: “The TPM holds the Storage Primary Seed (SPS) and generates Storage Root Keys (SRK) from SPS. The SRK are roots of Protected Storage Hierarchies associated with a TPM.[35] The storage keys in these hierarchies are used for symmetric encryption and signing of other keys and data together

[35] Windows creates only one protected storage hierarchy, and that is used by BitLocker.

with their security attributes. The resulting encrypted file, which contains header information in addition to the data or the key, is called a BLOB (Binary Large Object) and is output by the TPM and can be loaded in the TPM when needed. The private keys generated on the TPM can be stored outside the TPM (encrypted) in a way that allows the TPM to use them later without ever exposing such keys in the clear outside the TPM. The TPM uses symmetric cryptographic algorithms to encrypt data and keys ….”[36]

The TPM also provides protections that prevent the export of TPM keys and cryptographic data, such as the SPS and SRK, and anti-hammering mechanisms to prevent guessing of a TPM password.

6.2.1.4 Encrypting the Device with BitLocker

The BitLocker Data Encryption Key (DEK), also known as the Full Volume Encryption Key (FVEK), which encrypts the device’s storage volume is 128 bits for Windows Phone. For other Windows editions the administrator can choose to use either a 128 bit or 256 FVEK, however the instruction in the administrative guidance is use a 256 bit FVEK. The Windows RBG generates the FVEK. The FVEK is ultimately protected by keys within the TPM, namely the Storage Root Key (SRK) and the Storage Primary Seed, the latter is the Root Encryption Key (REK), and protects the SRK. During initialization, the TPM also generates the 2048-bit RSA key pair that is used as the SRK; sealing operations by the SRK in turn protects the BitLocker intermediate keys which are used by Windows when Windows boots (or resumes from hibernation and so the REK is isolated from operating system and applications and thus preventing reading and exporting the plaintext representation of the REK.

The key hierarchy for BitLocker shows an AES 256 CCM function is used to encrypt the Volume Master Key (VMK), which is a KEK and the Full Volume Encryption Key (FVEK), which is a DEK. The FVEK encrypts disk blocks using AES CBC.

The other KEKs are always 256 bits, and so their key size will always the same or larger than the FVEK.

For Windows 8.1, the Windows OS Loader will prompt the user for the Enhanced PIN which is used to generate a set of intermediate keys, one of which is sealed by the TPM; the ultimate result is a key which decrypts the encrypted VMK, which in turn decrypts the encrypted FVEK, thus enabling the Windows Loader to read the Windows kernel, ntoskrnl.exe, and then transfer execution to the kernel.

For Windows Phone, the Windows OS Loader will use the TPM to seal the Intermediate Key, which is then used to decrypt the encrypted VMK, which in turn decrypts the encrypted FVEK, thus enabling the Windows Loader to read the Windows kernel, ntoskrnl.exe, from the primary partition and then transfer execution to the kernel.

The FVEK and intermediate keys are all generated by the Windows RBG or by combining intermediate keys as described in FCS_CKM_EXT.3.

The unencrypted VMKs are zeroized after they are (1) used to encrypt the FVEK and (2) encrypted by an intermediate key. The other keys are also zeroized from volatile memory in the process of generating the VMK. When Windows shuts down normally or goes into hibernation, Windows will zeroize the FVEK

[36] Draft *Protection Profile PC Client Specific TPM*, FCS_COP.1/AES, page 5.

as part of shutdown. In the event of a system crash, the BitLocker Crash Dump Filter will zeroize the FVEK in order to prevent the FVEK from being included in the crash dump file.

6.2.1.5 Key Storage

The Key Isolation Service in Windows hosts secret and private keys within a protected process in order to mitigate tampering or access to sensitive key materials, which can be private keys, secret keys, or other secret material that need to be persisted. The NTFS files that the Key Isolation Service uses to store keys are protected by the Discretionary Access Control security policy described in the *Windows 8, Server 2012 Security Target*. In the NTFS file the key data is further is protected by the Data Protection API (DPAPI), which is described further below. The NTFS files are stored in NTFS volumes which is protected by BitLocker full disk encryption. Please see **Data at Rest Protection** for more information on BitLocker full disk encryption.

The IT administrator can configure Certificate Profiles in a Mobile Device Management (MDM) server for importing keys to the enrolled Windows devices. Applications import keys/secrets into the secure key storage by using the CertificateEnrollmentManager.ImportPfxDataAsync API. In addition, on Windows 8.1 devices users and local administrators can use the Certificate MMC Snap-in to import keys from Personal Information Exchange (.pfx) files into the secure key storage.

Private keys are protected on disk using DPAPI and BitLocker encryption and access is restricted using the Windows Discretionary Access Control Policy. When a Windows Store Application is deleted the local private keys imported by that app are deleted. All private keys are destroyed when a wipe operation is performed on a device. Local administrators can also perform a wipe on their Windows device to destroy all the keys or secrets. The IT administrator can perform a wipe operation of the enrolled device to destroy the keys.

Windows can restrict access to the application imported key/secret in secure key storage to only the application that imported the key or secret by using the subject identity for the Discretionary Access Control security policy as described in the *Windows 8 Server 2012 Security Target*. Users and local administrators authorize applications at installation to access shared keys or secrets when an application declares the **sharedUserCertificates** capability to share the certificate with other Windows Store Applications for the user. The **sharedUserCertificates** capability is described further in **Restricting Access to System Services**.

Destruction of keys/secrets imported into the secure key storage by applications is conducted automatically by the modern application environment after the keys/secrets are no longer in use.

For the purposes of this Mobile Device evaluation, the cryptographic module is the combination of the operating system and the device running Windows. After the device is configured the only persisted keys which protect user data via BitLocker are the Storage Root Key held by the TPM (the REK), the encrypted VMK (a KEK), and the encrypted FVEK (the DEK). When the device is turned on, the TPM checks the integrity of the SRK as described above, and then the Windows OS Loader unwraps the VMK and FVEK after the user provides the correct authorization factors. When a user provides their password

during interactive logon, Windows will use the submask derived from the password to provide access to private keys and secrets protected by DPAPI.

No unencrypted BitLocker key material is transmitted outside the cryptographic module. The encrypted FVEK, VMK, and Intermediate Key are stored on disk as metadata on the storage volume, however the metadata is stored outside of the mounted NTFS volume and so these are never transmitted outside the device, which the boundary of the cryptographic module in this evaluation.

6.2.1.6 Protecting Data with DPAPI

The Windows RBG generates a DPAPI Master Secret which is used as input into an AES function along with an initialization vector and encryption key, both of which are based on the user's password, to generate the encrypted DPAPI Master Secret. The DPAPI Master Secret is a kind of DEK and the password-based encryption key, which protects the DPAPI Master Secret is a kind of KEK. Also note that the DPAPI Master Secret is ultimately protected by the REK. The password encryption key is generated from a PBKDF2 function takes a result of a one-way function computation of the user's password.[37]

Windows will also combine the DPAPI Master Secret along with a salt value which will be used as an encryption key to protect user data, such as a private key. Each user will have a separate encryption key.

The integrity of both the encrypted DPAPI Master Secret and the encryption key is ensured by calculating MAC values.

6.2.1.7 Networking

Windows has a native implementation of IEEE 802.11-2012 to provide secure wireless local area networking (Wi-Fi). Windows uses PRF-384 in WPA2 Wi-Fi sessions and generates AES 128-bit keys using the Windows RBG. Windows complies with the IEEE 802.11-2012 standard and interoperates with other devices that implement the standard. TOE devices have received WPA2 certification, both Enterprise and Personal, and Wi-Fi CERTIFIED Interoperability Certificates from the Wi-Fi Alliance:

- Surface 3 (the Marvell 8897 adapter is also certified)
- Lumia 635
- Lumia 830

Windows implements key wrapping and unwrapping according to the NIST SP 800-38F specification (the "KW" mode) and so unwraps the Wi-Fi Group Temporal Key (GTK) which was sent by the access point. Because the GTK was protected by AES Key Wrap when it was delivered in an EAPOL-Key frame, the GTK is not exposed to the network.

[37] Note that data protected by DPAPI is also encrypted by BitLocker when the data is persisted to disk, and so the AES256 encrypted data will be encrypted a second time using the BitLocker 128-bit or 256-bit FVEK.

6.2.1.8 Network Protocols

6.2.1.8.1 TLS and EAP TLS

Windows 8.1 and Windows Phone implement TLS to enable a trusted network path that is used for both EAP, for client and server authentication, as well as HTTPS/ HTTP/TLS.

The following table summarizes the TLS RFCs implemented in Windows:

Table 6-4 TLS RFCs Implemented by Windows

RFC #	Name	How Used
2246	The TLS Protocol Version 1.0	Specifies requirements for TLS 1.0.
3268	Advanced Encryption Standard (AES) Ciphersuites for Transport Layer Security (TLS)	Specifies additional ciphersuites implemented by Windows.
3546	Transport Layer Security (TLS) Extensions	Updates RFC 2246 with TLS 1.0 extensions implemented by Windows.
4346	The Transport Layer Security (TLS) Protocol Version 1.1	Specifies requirements for TLS 1.0.
4366	Transport Layer Security (TLS) Extensions	Obsoletes RFC 3546 Requirements for TLS 1.0 extensions implemented by Windows.
4492	Elliptic Curve Cryptography (ECC) Cipher Suites for Transport Layer Security (TLS)	Specifies additional ciphersuites implemented by Windows.
4681	TLS User Mapping Extension	Extends TLS to include a User Principal Name during the TLS handshake.
5246	The Transport Layer Security (TLS) Protocol Version 1.2	Oboletes RFCs 3268, 4346, and 4366. Specifies requirements for TLS 1.2.
5289	TLS Elliptic Curve Cipher Suites with SHA-256/384 and AES Galois Counter Mode (GCM)	Specifies additional ciphersuites implemented by Windows.
SSL3	The SSL Protocol Version 3	Specifies requirements for SSL3.

Exceptions from the protocols are described in these documents:

- MS-TLSP Transport Layer Security (TLS) Profile.docx
- RFC 2246 - The TLS Protocol Version 1.0.docx
- RFC 3268 - AES Ciphersuites for TLS.docx
- RFC 3546 Transport Layer Security (TLS) Extensions.docx
- RFC 4366 Transport Layer Security (TLS) Extensions.docx
- RFC 4492 - ECC Cipher Suites for TLS.docx
- RFC 4681 - TLS User Mapping Extension.docx
- RFC 5246 - The Transport Layer Security (TLS) Protocol, Version 1.2.docx
- RFC 5289 - TLS ECC Suites with SHA-256384 and AES GCM.docx
- Internet Draft - SSL3 SSL 3.0 Specification.docx

The Cipher Suites in Schannel article describes the complete set of TLS cipher suites implemented in Windows (reference: http://msdn.microsoft.com/en-us/library/windows/desktop/aa374757(v=vs.85).aspx), of which the following are used in the evaluated configuration:

- TLS_RSA_WITH_AES_128_CBC_SHA
- TLS_RSA_WITH_AES_256_CBC_SHA
- 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_ECDHE_ECDSA_WITH_AES_128_GCM_SHA256 as defined in RFC 5289
- TLS_ECDHE_ECDSA_WITH_AES_256_GCM_SHA384 as defined in RFC 5289
- TLS_ECDHE_ECDSA_WITH_AES_128_CBC_SHA256 as defined in RFC 6460
- TLS_ECDHE_ECDSA_WITH_AES_256_CBC_SHA384 as defined in RFC 6460.

Each Windows component that uses TLS checks that the identifying information in the certificate matches what is expected, the component should reject the connection, these checks include checking the expected Distinguished Name (DN), Subject Name (SN), or Subject Alternative Name (SAN) attributes along with the applicable extended key usages. The DN, and any Subject Alternative Name, in the certificate is checked against the identity of the remote computer's DNS entry or IP address to ensure that it matches as described at http://technet.microsoft.com/en-us/library/cc783349(v=WS.10).aspx, and in particular the "Server Certificate Message" section.

Windows implements HTTPS as described in RFC 2818 so that Windows Store and system applications executing on the TOE can securely connect to external servers using HTTPS.

6.2.1.8.2 IPsec

The Windows IPsec implementation conforms to RFC 4301, Security Architecture for the Internet Protocol. This is documented publicly in the Windows protocol documentation at section 7.5.1 IPsec Overview.[38] Windows implements both RFCS 2409, Internet Key Exchange (IKEv1), and RFC 4306, Internet Key Exchange version 2, (IKEv2).[39] User-mode applications, which include Windows Store Applications, can transparently use IPsec networking services; networking traffic is isolated to the Windows kernel and the IPsec, IPsec Policy Agent, and IKE and AuthIP Keying Module user-mode service processes.

6.2.1.9 SFR Mapping

The **Cryptographic Support** function satisfies the following SFRs:

- **FCS_CKM.1(ASYM KA), FCS_CKM.1(ASYM AU)**: See **Table 6-2 Cryptographic Algorithm Standards and Evaluation Methods**.
- **FCS_CKM.1(WLAN), FCS_CKM.2**: Windows has a native implementation of IEEE 802.11.

[38] Also available as [MS-WSO], *Windows System Overview*, page 43 for offline reading.
[39] [MS-IKEE], *Internet Key Exchange Protocol Extensions*, page 8.

- **FCS_CKM_EXT.1**: The Windows devices in this evaluation use a root key of trust which prevents exporting or tampering the REK.
- **FCS_CKM_EXT.2(128), FCS_CKM_EXT.2(256)**: All data encrypting keys are generated by the Windows RBG, which has an input of at least 256 bits of entropy. The Windows Phone and data encrypting key is 128 bits, the Windows 8.1 data encrypting key is 256 bits in the evaluated configuration.
- **FCS_CKM_EXT.3**: Key encrypting keys have a security strength of 256 bits which is least as strong as the 128 bit or 256 bit disk encrypting key.
- **FCS_CKM_EXT.4**: Windows overwrites critical cryptographic parameters immediately after that data is no longer needed.
- **FCS_CKM_EXT.5**: Windows will delete the authorization factor to prevent access to protected data; after a wipe command Windows will format the partition to prevent access to protected data.
- **FCS_CKM_EXT.6**: When Windows needs to generate a salt, it uses the Windows random bit generator
- **FCS_COP.1(SYM)**: See **Table 6-2 Cryptographic Algorithm Standards and Evaluation Methods**.
- **FCS_COP.1(HASH)**: See **Table 6-2 Cryptographic Algorithm Standards and Evaluation Methods**.
- **FCS_COP.1(SIGN)**: See **Table 6-2 Cryptographic Algorithm Standards and Evaluation Methods**.
- **FCS_COP.1(HMAC)**: See **Table 6-2 Cryptographic Algorithm Standards and Evaluation Methods**.
- **FCS_COP.1(PBKD)**: Windows implements a FIPS Approved implementation of NIST SP 800-132.
- **FCS_IV_EXT.1**: When it is necessary to generate initialization vectors, Windows follows the guidance in Table 11: References and IV Requirements for NIST-approved Cipher Modes.
- **FCS_RBG_EXT.1**: See **Table 6-2 Cryptographic Algorithm Standards and Evaluation Methods**.
- **FCS_SRV_EXT.1**: See Section **6.2.1.2 Programming Interfaces**.
- **FCS_STG_EXT.1**: Windows provides secure key storage for private (asymmetric) keys, secret (symmetric) keys, and other data deemed by an authorized subject to require secure storage.
- **FCS_STG_EXT.2**: All keys in Windows are ultimately protected by the TPM-based root of trust for the devices included in this evaluation.
- **FCS_STG_EXT.3**: Key encrypting keys are protected by AES- MAC (CCM) mode.
- **FCS_TLS_EXT.1, FCS_TLS_EXT.2, FCS_HTTPS_EXT.1**: Windows implements TLS 1.0, 1.1, and 1.2 to provide confidentiality and integrity to upper-layer protocols such as Extensible Authentication Protocol and HTTP.

6.2.2 User Data Protection

6.2.2.1 Restricting Access to System Services

Windows Store Apps that need programmatic access resources such as device peripherals must declare the capabilities they require as part of the package manifest for the application.[40] There are two types of capabilities, the first is for developers who are registered as having individual accounts in the Windows

[40] This section is based on http://msdn.microsoft.com/en-us/library/windows/apps/hh464936.aspx.

Store; the second kind is for developers who are registered as having company accounts in the Windows Store. Applications from developers that are registered as companies can have additional capabilities.

The general-use capabilities that apply to most application scenarios are:

Table 6-5 General Use Capabilities

Capability	Description
Music	The **musicLibrary** capability provides programmatic access to the user's Music, allowing the app to enumerate and access all files in the library without user interaction. This capability is typically used in jukebox apps that need to access the entire Music library.
Pictures	The **picturesLibrary** capability provides programmatic access to the user's Pictures, allowing the app to enumerate and access all files in the library without user interaction. This capability is typically used in photo playback apps that need to access the entire Pictures library.
Videos	The **videosLibrary** capability provides programmatic access to the user's Videos, allowing the app to enumerate and access all files in the library without user interaction. This capability is typically used in movie playback apps that need access to the entire Videos library.
Removable Storage	The **removableStorage** capability provides programmatic access to files on removable storage, such as USB keys and external hard drives, filtered to the file type associations declared in the package manifest. For example, if a DOC reader app declared a .doc file type association, it can open .doc files on the removable storage device, but not other types of files.
internetClient	Windows 8.1 behavior: Can receive incoming data from the internet. Cannot act as a server. No local network access.[41] Windows Phone behavior: Full local and internet access and can act as a server. Inbound access to critical ports is always blocked.
internetClientClientServer[42]	Windows 8.1 behavior: Can receive incoming data from the internet. Can act as a server. No local network access. Windows Phone behavior: Full local and internet access and can act as a server. Inbound access to critical ports is always blocked.
Home and work networks	The **privateNetworkClientServer** capability provides inbound and outbound access to home and work networks through the firewall. This capability is typically used for games that communicate across the local area network (LAN), and for apps that share data across a variety of local devices. On Windows, this capability does not provide access to the internet. On Windows Phone, this capability provides the same access as

[41] This a “least privilege” security measure because many Windows Store Applications need only to receive or send data to remote web services (e.g., social network sites or weather apps) and not communicate with other hosts on the local network.

[42] Most Windows Store Apps that have a web service component will use **internetClient**. Apps that enable peer-to-peer (P2P) scenarios where the app needs to listen for incoming network connections should use **internetClientServer**.

	internetClient or **internetClientClientServer**.
Appointments	The **appointments** capability provides access to the user's appointment store. This capability allows read access to appointments obtained from the synced network accounts and to other apps that write to the appointment store.
Contacts	The **contacts** capability provides access to the aggregated view of the contacts from various contacts stores. This capability gives the app limited access (network permitting rules apply) to contacts that were synced from various networks and the local contact store.

Device capabilities allow the Windows Store App to access peripheral and internal devices. Device capabilities are specified with the **DeviceCapability** element in the app package manifest.

Table 6-6 Device Capabilities

Capability	Description
Location	The **location** capability provides access to location functionality, which you get from dedicated hardware like a GPS sensor in the PC or is derived from available network info. Apps must handle the case where the user has disabled location services from the **Settings** charm.[43]
Microphone	The **microphone** capability provides access to the microphone's audio feed, which allows the app to record audio from connected microphones. Apps must handle the case where the user has disabled the microphone from the **Settings** charm.
Proximity	The **proximity** capability enables multiple devices in close proximity to communicate with one another. This capability is typically used in casual multi-player games and in apps that exchange information. Devices attempt to use the communication technology that provides the best possible connection, including Bluetooth, Wi-Fi, and the internet. This capability is used only to initiate communication between the devices.
Webcam	The **webcam** capability provides access to the video feed of a built-in camera or external webcam, which allows the app to capture photos and videos. On Windows, apps must handle the case where the user has disabled the camera from the **Settings** charm.
USB	The **usb** device capability enables access to APIs in the **Windows.Devices.Usb** namespace. This capability is used only Windows 8.1, not Windows Phone.
Human interface device (HID)	The **humaninterfacedevice** device capability enables access to APIs in the **Windows.Devices.HumanInterfaceDevice** namespace. This namespace enables the Windows Store App to access devices that support the Human Interface Device (HID) protocol.
Bluetooth GATT	The **bluetooth.genericAttributeProfile** device capability enables

[43] A charm is an admin tool available by opening the Windows Settings page by swiping from the left side of the screen.

	access to APIs in the Windows.Devices.Bluetooth.GenericAttributeProfile namespace. This namespace enables the Windows Store App to access Bluetooth LE devices through a collection of primary services, included services, characteristics, and descriptors.
Bluetooth RFCOMM	The **bluetooth.rfcomm** device capability enables access to APIs in the **Windows.Devices.Bluetooth.Rfcomm** namespace. This namespace supports the Basic Rate/Extended Data Rate (BR/EDR) transport and also enables the Windows Store App to access a device that implements Serial Port Profile (SPP).

The additional capabilities associated with Windows Store Applications which are from company accounts are highly restricted and require additional review before the App is published to the Windows Store.

Table 6-7 Special Use Capabilities

Capability	Description
Enterprise authentication	Windows domain credentials, which are domain username and password for a particular user, enable the user to log into remote resources using their credentials, and act as if a user provided their user name and password. The **enterpriseAuthentication** capability is typically used in line-of-business apps that connect to servers within an enterprise and is not needed for basic communications over the Internet. The Enterprise Authentication capability allows a Windows Store App to use the Credential Manager when prompted for domain credentials.
Shared User Certificates	The **sharedUserCertificates** capability enables a Windows Store Application to access software and hardware certificates, such as certificates stored on a smart card, the certificate is stored in the user's DPAPI profile location instead of the DPAPI profile associated with the Windows Store Application
Documents	The **documentsLibrary** capability provides programmatic access to the user's Documents, filtered to the file type associations declared in the package manifest, to support offline access to OneDrive. For example, if a DOC reader app declared a .doc file type association, it can open .doc files in Documents, but not other types of files.

As part of installing a Windows Store Application, the user is prompted to authorize the use of the capability by the App, after the App has been installed is it allowed to access the capability when running on behalf of the user. When an App requests to access a resource that is managed by a capability, the Windows App Container, checks if the App has been authorized access, according to the installed package manifest, and then provides mediated access to the resource. In addition to the application-level isolation, Windows also restricts access to hardware resources through the

discretionary access control security policy and kernel-mode / user-mode architecture described in the *Windows 8 Server 2012 Security Target*.

6.2.2.2 Data at Rest Protection

The entire storage volume is protected by BitLocker full disk encryption, this includes user data, Windows configuration (TSF) data, and all programs other than the BitLocker programs needed to unlock the drive. BitLocker in Windows 8 was evaluated against the NIAP Software Full Disk Encryption Protection Profile (certificate # 10540). Device Encryption, the term Windows Phone full disk encryption is the same as BitLocker, however the Software Full Disk Encryption protection profile included requirements for authorization factors which are not part of those two products; otherwise the implementations are the same, using AES CBC mode with 128-bit blocks for Windows Phone and an administrator-specified 128- or 256-bit blocks for Windows 8.1. The administrative guidance recommends using AES 256.

When the local administrator decides to wipe the device, or the IT administrator decides to wipe a phone using a MDM, Windows will delete the BitLocker metadata, which includes the authorization factors that unlock the device. Without the BitLocker metadata, the encrypted data on the storage volume is effectively wiped. The wiping of the BitLocker metadata from flash memory on Windows 8.1 is performed by first overwriting the metadata with zeros, then overwriting the data with ones and finally overwriting the data with random bytes, each step is followed by a read-verify. On Windows Phone 8.1 the metadata is deleted in the same manner as Windows 8.1. After deleting the metadata, Windows will reboot and install a fresh copy of the operating system from a recovery partition.

6.2.2.3 Protecting Sensitive User Data

Windows 8.1 and Windows Phone 8.1 can provide an additional layer of protection using the Encrypting File System which is a per-file encryption capability at an architectural layer above BitLocker (which encrypt disk blocks on the storage volume). For Windows 8.1, the user can choose which files to encrypt, Windows Phone 8.1 will automatically encrypt the user's enterprise mail folder which, in the context of this evaluation, is deemed to be sensitive data. The application chooses which keys and data to protect by using the CNG DPAPI and specifying the "local locked credentials" protection descriptor which uses an asymmetric encryption scheme to protect the user data.

When the screen is locked, Windows Phone will suspend all Windows Store Applications except those which are registered to display notifications on the lock screen as described in section **6.2.6**. These applications can use the CNG DPAPI to protect their data. When the screen has been unlocked, the application decrypts the sensitive data using CNG DPAPI. When the data is ultimately persisted to disk it is encrypted again using the BitLocker FVEK. To add more detail, received email content is encrypted by the public portion of a key pair. When the screen is unlocked, Windows will (1) retrieve an encrypted copy of the private portion of the key pair from the Windows registry (the private was deleted from volatile memory when the screen was locked), (2) decrypt the private using a symmetric encryption key associated with the user, and (3) then use the private key to decrypt the sensitive data.

The symmetric encryption key mentioned in the previous session is a Device User Credential Key (DUCK) which is a 256-bit value generated by the RBG. During initial logon or when changing credentials, Windows Phone will seal this key and a hash of the user's password. The DUCK then encrypts the private portion of the key pair described above. When the screen is locked both the DUCK and the private portion of the CNG DPAPI key pair are deleted from volatile memory.

6.2.2.4 Certificate Storage

The MDF PP defines the *Trust Anchor Database* as "[a] list of trusted root Certificate Authority certificates". In a Windows OS, these certificates are known as *trusted root certificates*, which are contained in certificate stores. Each user has their own certificate store and there is a certificate store for the computer account; access to a certificate store is managed by the discretionary access control policy in Windows such that only the authorized administrator, i.e., the user or the local administrator, can add or remove entries.[44] Certificates which are used by applications, for example, IPsec and TLS, are also placed in certificate stores for the user.

In addition to the standard certificate revocation processes, application certificates can be loaded by either using administrative tools such as **certutil.exe**, changes to the trusted root certificates can be made using Certificate Trust Lists.

6.2.2.5 VPN Client

The Windows IPsec VPN client can be configured by the device local administrator or the MDM IT administrator, when the device is enrolled. The administrator can also configure the IPsec VPN client that traffic is routed through the IPsec. The IPsec VPN is an end-to-end internetworking technology and so VPN sessions can be established over physical network protocols such as mobile broadband (ex. LTE) wireless LAN (Wi-Fi), or local area network.

The components responsible for routing IP traffic through the VPN client:

- The **IPv4 / IPv6 network stack** in the kernel processes ingoing and outgoing network traffic.
- The **IPsec** and **IKE and AuthIP Keying Modules** service which hosts the IKE and Authenticated Internet Protocol (AuthIP) keying modules. These keying modules are used for authentication and key exchange in Internet Protocol security (IPsec).
- The **Remote Access Service** device driver in the kernel, which is used primarily for VPN connections; known as the "RAS IPsec VPN" or "RAS VPN".
- The **IPsec Policy Agent** service which enforces IPsec policies.

6.2.2.6 SFR Mapping

The **User Data Protection** function satisfies the following SFRs:

- **FDP_ACF_EXT.1**: Through the use of capabilities that Windows Store Applications request during installation, Windows restricts system services to Apps.

[44] Refer to the Windows 8 Operating System Protection Profile evaluation for more information about the discretionary access control policy.

- **FDP_DAR_EXT.1(128), FDP_DAR_EXT.1(128)**: All user data and all Windows data is encrypted on the device.
- **FDP_DAR_EXT.2**: All sensitive data in the Windows Phone is encrypted when the screen is locked.
- **FDP_STG_EXT.1**: Windows provides a trusted and secure store for certificates.

6.2.3 Identification and Authentication

All logons are treated essentially in the same manner regardless of their source (e.g., interactive logon, network interface, internally initiated service logon) and start with an account name, domain name (which may be NULL; indicating the local system), and credentials that must be provided to the TSF.

The Local Security Authority component within Windows maintains a count of the consecutive failed logon attempts by security principals from their last successful authentication. When the number of consecutive failed logon attempts is larger than the policy for failed logon attempts, which ranges from 0 (never lockout the account) to 999, Windows 8.1 will lockout the user account and Windows Phone will wipe the user data from the device. Interactive logons are done on the secure desktop, which does not allow other programs to run, and therefore prevents automated password guessing. In addition, the Windows logon component enforces a one second delay between every failed logon with an increased delay after several consecutive logon failures.

The Windows implementation of Bluetooth follows the Bluetooth SIG Specification, including OBEX data transfer and OPP (object push profile). The OBEX specification, which Windows implements, prevents any transfer of user data until both Bluetooth devices have paired. When a Windows OS encounters an unpaired device, it does not transfer any data to the unpaired device.

6.2.3.1 Protecting User Data

Windows protects user data with BitLocker, which encrypts the entire device; the user's persistent keys and secrets additionally protected by DPAPI. At the most basic level, all data on stored on the device is encrypted by BitLocker using FIPS Approved symmetric encryption algorithms. During boot, Windows will derive disk encryption keys (DEK) and key encryption keys (KEK) based on the BitLocker authorization factors that unlock the device; the administrative guidance for Windows 8.1 includes the configuration for an additional BitLocker authorization factors which is a device password, technically known as the "Enhanced PIN", that includes uppercase and lowercase English letters, symbols on an EN-US keyboard, numbers, special characters and spaces. The system and user (protected) data remains encrypted in non-volatile storage, the file system device driver uses the BitLocker FVEK (a DEK) to decrypt the data as it is loaded into volatile storage. The only time user (protected) data is decrypted is after the user authenticates by providing their Enhanced PIN password, for Windows 8.1, and password, for Windows Phone. That password is used to derive the DPAPI secret (a KEK) which provides an additional layer of protection for certain user data, including keys.

6.2.3.2 X.509 Certificate Validation

Each Windows component that uses X.509 certificates is responsible for performing certificate validation, however all components use a common subcomponent, which validates certificates as

described in RFC 5280 including all applicable usage constraints such as Server Authentication for networking sessions and Code Signing when installing product updates. Each component that uses X.509 certificates will have a repository for public certificates and will select a certificate based on criteria such as entity name for the communication partner, any extended key usage constraints, and cryptographic algorithms associated with the certificate.

If certificate validation fails, or if Windows is not able to check the validation status for a certificate, Windows will not establish a trusted network channel (IPsec, TLS), however it will inform the user and seek their consent before establishing a HTTPS web browsing session. Certification validation for software installation and updates is described in section **6.2.5.6**.

Modern Store Applications can use these interfaces to check the validity of certificates:

- Certificate.BuildChainAsync
- CertificateChain. Validate

6.2.3.3 SFR Mapping

- **FIA_AFL_EXT.1**: After the number of consecutive failed authentication attempts for a user account has been surpassed, Windows Phone will wipe the device and Windows 8.1 and will lock out the user account.
- **FIA_BLT_EXT.1**: Windows require Bluetooth mutual authentication between the Windows device any the remote device prior to any data transfer over the Bluetooth connection.
- **FIA_PAE_EXT.1**: Windows conforms to IEEE 802.1X as a Port Access Entity acting in the Supplicant role.
- **FIA_PMG_EXT.1**: Windows devices support logon passwords at least 15 characters in length. Windows 8.1 and Windows Phone logon passwords can be composed from uppercase characters, lowercase characters, digits, and special characters to be used in passwords.
- **FIA_TRT_EXT.1**: Windows logon component enforces a one second delay between every failed logon.
- **FIA_UAU.7**: During an interactive logon, Windows echoes the users password with "*" characters to prevent disclosure of the user's password.
- **FIA_UAU_EXT.1**: The user must authenticate successfully during interactive logon and prior to decryption of any user data stored on the device.
- **FIA_UAU_EXT.2**: The only actions that an unauthorized user can take when a Windows device is locked is to bring up the authentication dialog, turn the device off, or place an emergency call.
- **FIA_UAU_EXT.3**: Windows requires that a user provide the correct password prior to changing their password and when unlocking their device.
- **FIA_X509_EXT.1, FIA_X509_EXT.3**: Windows validates X.509 certificates according to RFC 5280 and provides OCSP and CRL services for applications to check certificate revocation status.
- **FIA_X509_EXT.2**: Windows uses X.509 certificates for EAP-TLS exchanges, TLS, HTTPS, IPsec, code signing for system software updates, code signing for mobile applications, and code signing for integrity verification.

6.2.4 Security Management

The complete set of management functions are described in Security Management (FMT), the following table maps which activities can be done by the device user (who is considered to be a standard user in a Windows client OS), the device administrator (who is considered to be a local administrator), and invoked by a mobile device manager. A person who uses a Windows Phone is both a device user and the local administrator of the device (based on the Windows security model); a person who uses a Windows 8.1 device may either be a standard user or a local administrator depending on the kind of user account created for the person. In the terminology of the MDF PP, the device user and (device) local administrator correspond to the FMT_MOF.1(USER) requirement and the MDM Agent corresponds to the FMT_MOF.1(ORG) requirement because the latter refers to management capabilities after a device has been enrolled into a MDM.

Table 6-8 Mobile Device Management Capabilities

Activity	Device User	Device (Local) Administrator	MDM Agent
Configure password policy		Windows 8.1	Windows Phone
Configure session locking policy		Windows 8.1	Windows Phone
Enable/disable the VPN protection	√	√	
Enable/disable Wi-Fi, mobile broadband radios, Bluetooth	√	√	√
Enable/disable camera, microphone	√	√	
Specify wireless networks (SSIDs) to which the TSF may connect			√
Configure security policy for connecting to wireless networks			√
Transition to the locked state	√	√	
Full wipe of protected data	Windows Phone	√	
Configure application installation policy			√
Import keys/secrets into the secure key storage	√	√	
Destroy imported	Windows Phone	√	

Activity	Device User	Device (Local) Administrator	MDM Agent
keys/secrets and any other keys/secrets in the secure key storage			
Import X.509v3 certificates into the Trust Anchor Database			√
Remove imported X.509v3 certificates and any other X.509v3 certificates in the Trust Anchor Database			√
Enroll the TOE in management	√	√	
Remove applications			√
Update system software			√
Install applications			√
Enable/disable data transfer capabilities over USB port for Windows 8.1, Bluetooth	Windows Phone	√	
Enable/disable personal Hotspot connections, tethered connections	√	√	
Enable/disable wireless remote access connections except for personal Hotspot service, personal Hotspot connections, tethered connections			√
Enable data-at rest protection		Windows 8.1	Windows Phone
Enable removable media‘s data-at-rest protection	Windows 8.1	Windows 8.1	
Configure the Access Point Name and proxy used for communications between the cellular network and other	√	√	

Activity	Device User	Device (Local) Administrator	MDM Agent
networks			
Enable/disable display notification in the locked state	√	√	
Wipe sensitive data	Windows Phone	√	Windows Phone
Alert the administrator			√
Remove Enterprise applications			√
Approve import and removal by applications of X.509v3 certificates in the Trust Anchor Database	√	√	
Enable/disable cellular voice functionality	Windows Phone	Windows Phone	
Enable/disable device messaging capabilities	Windows Phone	Windows Phone	
Enable/disable the cellular protocols used to connect to cellular network base stations	Windows Phone	Windows Phone	
Configure the unlock banner	Windows Phone	√	
Enable/disable location services	√	√	

6.2.4.1 SFR Mapping

The **Security Management** function satisfies the following SFRs:

- **FMT_MOF.1(USER)**: Windows provides the user with the capability to administer the security functions described in the security target. The mappings to specific functions are described in each applicable section of the TOE Summary Specification.
- **FMT_MOF.1(ORG)**: Windows provides the authorized administrator with the capability to administer the security functions described in the security target when the device is enrolled. The mappings to specific functions are described in each applicable section of the TOE Summary Specification.
- **FMT_SMF.1**: Windows provides the management functions that are described by FMT_MOF.1(USER) and FMT_MOF.1(ORG).
- **FMT_SMF_EXT.1**: After unenrollment, Windows will remove enterprise applications and inform the administrator that the device is no longer enrolled.

6.2.5 Protection of the TSF

6.2.5.1 Separation and Domain Isolation

The TSF provides a security domain for its own protection and provides process isolation. The security domains used within and by the TSF consists of the following:

- Hardware
- Virtualization Partitions (Windows 8 only)
- Kernel-mode software
- Trusted user-mode processes
- User-mode Administrative tools process

The TSF hardware is managed by the TSF kernel-mode software and is not modifiable by untrusted subjects. The TSF kernel-mode software is protected from modification by hardware execution state and protection for both physical memory and memory allocated to a partition; an operating system image runs within a partition. The TSF hardware provides a software interrupt instruction that causes a state change from user mode to kernel mode within a partition. The TSF kernel-mode software is responsible for processing all interrupts, and determines whether or not a valid kernel-mode call is being made. In addition, the TSF memory protection features ensure that attempts to access kernel-mode memory from user mode results in a hardware exception, ensuring that kernel-mode memory cannot be directly accessed by software not executing in the kernel mode.

The TSF provides process isolation for all user-mode processes through private virtual address spaces (private per process page tables), execution context (registers, program counters), and security context (handle table and token). The data structures defining process address space, execution context and security context are all stored in protected kernel-mode memory. All security relevant privileges are considered to enforce TSF Protection.

User-mode administrator tools execute with the security context of the process running on behalf of the authorized administrator. Administrator processes are protected like other user-mode processes, by process isolation.

Like TSF processes, user processes also are provided a private address space and process context, and therefore are protected from each other. Additionally, the TSF has the added ability to protect memory pages using Data Execution Prevention (DEP) which marks memory pages in a process as non-executable unless the location explicitly contains executable code. When the processor is asked to execute instructions from a page marked as data, the processor will raise an exception for the OS to handle.

The TSF implements cryptographic mechanisms within a distinct user-mode process, where its services can be accessed by both kernel- and user-mode components, in order to isolate those functions from the rest of the TSF to limit exposure to possible errors while protecting those functions from potential tampering attempts.

Furthermore, the TSF includes a Code Integrity Verification feature, also known as Kernel-mode code signing (KMCS), whereby device drivers will be loaded only if they are digitally signed by either Microsoft or from a trusted root certificate authority recognized by Microsoft. KMCS uses public-key cryptography technology to verify the digital signature of each driver as it is loaded. When a driver tries to load, the TSF decrypts the hash included with the driver using the public key stored in the certificate. It then verifies that the hash matches the one that it computes based on the driver code using the FIPS - certified cryptographic libraries in the TSF. The authenticity of the certificate is also checked in the same way, but using the certificate authority's public key, which must be configured in and trusted by the TOE.

6.2.5.1.1 Supporting Hardware

The devices used in the evaluation have the following characteristics:

Device	Processor	Hardware Specifications
Surface 3	Intel Atom Z8700	SoC • Max freq.: 1.6GHz base, 2.4GHz Burst mode • L3 Cache: 2MB • Cores: Quad core (no Hyperthreading) • Gfx execution units / freq.: 16 / 600 (400MHz nominal) Memory • Memory support: 2 and 4 GB SKUs available using LPDDR3 1600 memory Storage • Storage: 64GB and 128GB SKUs available using eMMC storage GPU • Gen8 PL Arch. DX11 Wireless • Marvell 88W8897: 802.11a/b/g/n/ac 2x2 MIMO, Bluetooth 4.0
Lumia 635	Snapdragon™ 400 Quad-core 1.2GHz ARM® Cortex™ A7	http://infocenter.arm.com/help/index.jsp?topic=/com.arm.doc.ddi0464f/index.html (See Memory Management Unit)
Lumia 830	Snapdragon™ 400 Quad-core 1.2GHz ARM® Cortex™ A7	http://infocenter.arm.com/help/index.jsp?topic=/com.arm.doc.ddi0464f/index.html (See Memory Management Unit)

Table 6-9 Supporting Hardware

6.2.5.2 Protection from Implementation Weaknesses

Windows runs on processors that provide support for virtual memory and enforce restrictions to read, write, and execute pages of virtual and physical memory. Collectively, this is known as Data Execution Prevention (DEP). On Intel platforms, DEP is called NX (no execute), ARM platforms call DEP XN (execute never).

The Windows kernel, user-mode applications, and all Windows Store Applications implement Address Space Layout Randomization (ASLR) in order to load executable code at unpredictable base addresses. The base address is generated using a pseudo-random number generator that is seeded by high quality entropy sources when available which provides at least 8 random bits for memory mapping. [45]

The Windows runtime also provides stack buffer overrun protection capability that will terminate a process after Windows detects a potential buffer overrun on the thread's stack by checking canary values in the function prolog and epilog as well as reordering the stack. All Windows binaries and Windows Store Applications implement stack buffer overrun protection by being complied with the /GS option, which is used for all Windows binaries; and checking that all Windows Store Applications are compiled with buffer overrun protection before ingesting the Windows Store Application into the Windows Store.

To enable these protections using the Microsoft Visual Studio development environment, programs are complied with /DYNAMICBASE option for ASLR, and optionally with /HIGHENTROPYVA for 64-bit ASLR, or /NXCOMPAT:NO to opt out of software-based DEP, and /GS (switched on by default) for stack buffer overrun protection.

Windows Store Applications are compiled with the /APPCONTAINER option which builds the executable to run in a Windows appcontainer, to run with the user-mode protections described in this section.

6.2.5.3 Time Service

Each hardware platform supported by the TOE includes a real-time clock. The real-time clock is a device that can only be accessed using functions provided by the TSF and serves as the reference clock that maintains the system time. Specifically, the TSF provides functions that allow users, including the TSF itself, to query and set the clock, as well as functions to synchronize clocks within a domain. The ability to query the clock is unrestricted, while the ability to set the clock requires the SeSystemtimePrivilege. This privilege is only granted to authorized administrators to protect the integrity of the time service.

Synchronizing the clocks within a managed Windows deployment is critical for cross-machine communications and correlating activities which occur on multiple computers. Accuracy (which the NIAP OS PP describes as "reliable and monotonically increasing" is described in How the Windows Time Service Works. In addition this communications path can be protected using IPsec between the computers in the Active Directory domain.

How To Configure an Authoritative Time Server in Windows Server describes additional steps a domain administrator can take to explicitly specify the reference clock for the domain or an arbitrary NTP server. Alternatively, devices with a mobile broadband modem can synchronize to the carrier network's time source.

Windows capabilities that are included in the OS protection profile evaluation which use the centralized (i.e., reliable) time service are:

[45] The PRNG is seeded by the TPM RBG, the RDRAND instruction and other sources.

- Audit record generation
- Network expirations for authentication and data access
- Session timeout and screen locking
- X.509 certificate generation, revocation, and expiration

These capabilities use the interfaces described at http://msdn.microsoft.com/en-us/library/ms725473(v=vs.85).aspx. Public documentation about time functions in Windows is located at http://msdn.microsoft.com/en-us/library/ms724962(v=vs.85).aspx. This describes the different types of time services offered to developers.

6.2.5.4 Self-Tests

The Windows self-tests are a collection of tests which verify that the Windows is operating correctly. The self-tests are enabled when the administrator sets the "System Cryptography: Use FIPS compliant algorithms for encryption, hashing, and signing" policy; Windows will always run the self-tests described in this section.

The kernel-mode startup self-tests are:[46]

- AES-128 encrypt/decrypt EBC Known Answer Test
- AES-128 encrypt/decrypt CBC Known Answer Test
- AES-128 CMAC Known Answer Test
- AES-128 encrypt/decrypt CCM Known Answer Test
- AES-128 encrypt/decrypt GCM Known Answer Test
- RSA Known Answer Test
- ECDSA sign/verify test on P256 curve
- ECDH secret agreement Known Answer Test on P256 curve
- HMAC-SHA-1 Known Answer Test
- HMAC-SHA-256 and HMAC-SHA-512 Known Answer Tests
- SP800-56A concatenation KDF Known Answer Tests (same as Diffie-Hellman KAT)
- SP800-90 AES-256 counter mode DRBG Known Answer Tests (instantiate, generate and reseed)
- SP800-90 Dual-EC DRBG Known Answer Tests (instantiate, generate and reseed)

The Windows kernel-mode cryptographic module, the Kernel Mode Cryptographic Primitives Library, also performs pair-wise consistency checks upon each invocation of RSA, ECDH, and ECDSA key-pair generation and import as defined in FIPS 140-2. SP 800-56A conditional self-tests are also performed. A continuous RNG test (CRNGT) is used for the random number generators of this cryptographic module. All approved and non-approved RNGs have a CRNGT. The SP 800-90 DRBGs have health tests. A pair-wise consistency test is done for Diffie-Hellman.

The Kernel Mode Cryptographic Primitives Library is loaded into the kernel's memory early during the boot process. If there is a failure in any startup self-test, the Kernel Mode Cryptographic Primitives

[46] When the System Cryptography policy is set, Windows will always perform these self-tests however the evaluated configuration does not use the ECDH, HMAC, and SP800-56A algorithms.

Library DriverEntry function will fail to return the STATUS_SUCCESS status to its caller. The only way to recover from the failure of a startup self-test is to attempt to invoke DriverEntry again, which will rerun the self-tests, and will only succeed if the self-tests passes.

By thoroughly exercising the cryptographic functions, Windows will prevent situations where user data is not stored in an encrypted state.

All operations on the TSF ultimately involve the use of cryptography, and so the FIPS 140 health tests are sufficient to determine that Windows is operating correctly.

6.2.5.5 Windows Code Integrity

A Windows operating system verifies the integrity of Windows program code using the Secure Boot and Code Integrity capability in Windows.[47] On computers with a TPM (either discrete or firmware), such as those used in the Mobile Device evaluation, before Windows will unlock the operating system drive, it will verify the integrity of the early boot components, which include the Boot Loader, OS Loader, and OS Resume binaries, in order to prevent tampering and to ensure that the drive is in the same computer as when the OS was initialized.

The Secure Boot capability Windows checks that the file integrity of early boot components has not been compromised and ensures that the files have not been modified, which mitigates the risk of rootkits and viruses, and that the data elements that contribute to creating the composite keys, which will ultimately unlock the operating system drive, have not been compromised. Secure Boot collects these file measurements and seals them to the TPM. When Secure Boot starts in the preboot environment, it will compare the sealed values from the TPM and if those values do not match the calculated values, Secure Boot will lock the system (which prevents booting) and display a warning on the computer display.

After Secure Boot verifies the integrity of early-running kernel components, including Code Integrity, the Code Integrity capability provides measures code integrity for kernel-mode and user-mode programs. Kernel-mode code signing (KMCS) prevents kernel-mode device drivers, such as the BitLocker Drive Encryption Drivers (fvevol.sys), from loading unless they are published and digitally signed by developers who have been vetted by one of a handful of trusted certificate authorities (CAs). KMCS, using public-key cryptography technologies, requires that kernel-mode code include a digital signature generated by one of the trusted certificate authorities. When a kernel device driver tries to load, Windows decrypts the hash included with the driver using the public key stored in the certificate, then verifies that the hash matches the one computed with the code. The authenticity of the certificate is checked in the same way, but using the certificate authority's public key, which is trusted by Windows. The root public key of the certificate chain that verifies the signature must match one of the Microsoft's root public keys indicating that Microsoft is the publisher of the Windows image files. These Microsoft's root public keys are hardcoded in the Windows boot loader.

[47] In MDF PP terminology, Windows runs on the application processor.

The cryptography used by Secure Boot and Code Integrity is validated as part of the Windows FIPS 140 validation.

6.2.5.6 Windows and Application Updates

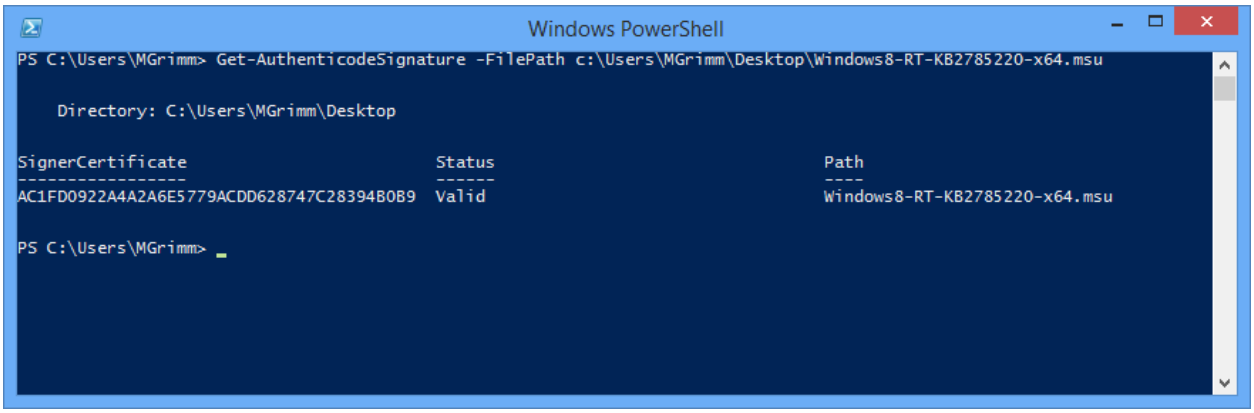
Updates to Windows are delivered as Microsoft Update Standalone Package files (.msu files) and are signed by Microsoft with two digital signatures, a SHA1 signature for legacy applications and a SHA256 signature for modern applications. The RSA SHA256 digital signature is signed by *Microsoft Corporation*, with a certification path through a Microsoft Code Signing certificate and ultimately the Microsoft Root Certification Authority. These certificates are checked by the Windows Trusted Installer prior to installing the update.

The Windows operating system will check that the certificate is valid and has not been revoked using a standard PKI CRL. Once the Trusted Installer determines that the package is valid, it will update Windows; otherwise the installation will abort and there will be an error message in the event log. Note that the Windows installer will not install an update if the files in the package have lower version numbers than the installed files.

The integrity of the Microsoft Code Signing certificate on the computer is protected by the storage root key within the TPM, and the validated integrity of the Windows binaries as a result of Secure Boot and Code Integrity.

Updates to Windows are delivered through the Windows Update capability, which is enabled by default, or the user can go to http://www.microsoft.com/security/default.aspx to search and obtain security updates on their own volition. Windows Phone users can go to Settings, and the Phone Update to search for updates.

A user can then check that the signature is valid either by viewing the digital signature details of the file from Windows Explorer or by using the Get-`AuthenticodeSignature` PowerShell Cmdlet. The following is an example of using PowerShell:



If the Get-`AuthenticodeSignature` PowerShell Cmdlet or Windows Explorer could not verify the signature, the status will be marked as invalid. This verification check uses the same functionality described above.

6.2.5.6.1 Windows Store Applications

In the same manner as checking the integrity of the Microsoft Update Packages and Windows executable code, Windows Store Applications and their installation packages are verified using a digital signature from *Microsoft Corporation* with the Code Signing usage.

6.2.5.7 SFR Mapping

The **TSF Protection** function satisfies the following SFRs:

- **FPT_AEX_EXT.1**: All Windows Store applications use address space layout randomization.
- **FPT_AEX_EXT.2**: The Intel and Qualcomm processors included in this evaluation enforce read, write, and execute permissions for physical memory.
- **FPT_AEX_EXT.3**: Windows binaries are compiled with stack overflow protection (compiled using the **/Gs** option for native applications). **Appendix D: TOE Binary List** contains a list of Windows binaries along with any exceptions which do not use stack overflow protection.
- **FPT_AEX_EXT.4**: The Windows kernel and user-mode system services protect themselves from modification by untrusted subject programs; moreover user-mode programs execute in separate virtual address spaces.
- **FPT_KST_EXT.1**: During normal operation, Windows does not store plaintext key material in non-volatile storage.
- **FPT_KST_EXT.2**: Plaintext keys are not exported from the FIPS-validated cryptographic modules.
- **FPT_KST_EXT.3**: Users cannot export plain text keys from Windows Store applications.
- **FPT_NOT_EXT.1**: Windows will fall into a non-operational state after a failure of the Windows FIPS 140 cryptographic self-tests and integrity failure for Windows system binaries.
- **FPT_STM.1**: The real-time clock in each Windows platform, in conjunction with periodic domain synchronization, for domain-joined devices, and time signals from the LTE network, provide a reliable source of time stamps for the TSF; changing the clock can be restricted to authorized administrators.
- **FPT_TST_EXT.1**: Windows runs a series of self-tests that confirm that essential cryptographic operations are performed correctly and halts if the self-tests fail. Those cryptographic functions are then used to check integrity of TOE executables.
- **FPT_TST_EXT.2**: Windows checks the integrity of the Windows boot loader, OS loader, kernel, and system binaries and all application executable code, i.e, Windows Store Applications and updates to Windows and Windows Store Applications.
- **FPT_TUD_EXT.1**: Windows provides a means to identify the current version of the Windows software, the hardware model, and installed applications.
- **FPT_TUD_EXT.2**: Windows has an update mechanism to deliver updated binaries and a means for a user to confirm that the digital signatures, which ensure the integrity of the update, are valid for both the operating system and Windows Store Applications.

6.2.6 TOE Access

6.2.6.1 Windows 8.1

Windows provides the ability for a user to lock their interactive logon session at their own volition or after a user-defined inactivity timeout. Windows also provides the ability for the administrator to specify the interval of inactivity after which the session will be locked. This policy will be applied to either the local machine or the computers within a domain using either local policy or group policy respectively. If both the administrator and a standard user specify an inactivity timeout period, Windows will lock the session when the shortest time period expires.

Once a user has a desktop session, they can invoke the session locking function by using the same key sequence used to invoke the trusted path (**Ctrl+Alt+Del**). This key sequence is captured by the TSF and cannot be intercepted or altered by any user process. The result of that key sequence is a menu of functions, one of which is to lock the workstation. The user can also lock their desktop session by going to the Start screen, selecting their logon name, and then choosing the "Lock" option.

Windows constantly monitors the mouse, keyboard, touch display, and the orientation sensor for inactivity in order to determine if they are inactive for the specified time period. After which, Windows will lock the workstation and execute the screen saver unless the user is streaming video such as a movie. Note that if the workstation was not locked manually, the TSF will lock the display and start the screen saver program if and when the inactivity period is exceeded, as well any notifications from applications which have registered to publish the application's badge or the badge with associated notification text to the locked screen.[48] The user has the option to not display any notifications, or choose one Windows Store Application to display notification text, and select other applications display their badge.

For Windows Phone the inbox Calendar, Mail, [SMS] Messaging, and Phone applications can generate notifications, and when selected to display notification text they will show the location and time of the upcoming and in-progress meeting, the sender and subject line of the last received email, the sender and text from the last received SMS message, and the last phone caller and caller notification respectively.

For Windows 8.1 the inbox Calendar, Weather, and Alarm applications can generate notifications, and when selected to display notification text they will show the location and time of the upcoming and in-progress meeting, the current weather conditions, and an expired alarm times. In addition, Mail application can be configured to display a badge but not notification text.

After the computer was locked, in order to unlock their session, the user either presses a key or swipes the display. The user must provide the **Ctrl+Alt+Del** key combination if the **Interactive Logon: Do not required CTRL+ALT+DEL** policy is set to disabled.[49] Either action will result in an authentication dialog.

[48] The badge is a logo which represents the Windows Store Application and the notification text can be items such as a count of unread messages or an appointment.
[49] This policy is defined under Local Policies / Security Options.

The user must then re-enter their authentication data, which has been cached by the local system from the initial logon, after which the user's display will be restored and the session will resume. Alternately, an authorized administrator can enter their administrator identity and password in the authentication dialog. If the TSF can successfully authenticate the administrator, the user will be logged off, rather than returning to the user's session, leaving the workstation ready to authenticate a new user.

As part of establishing the interactive logon session, Windows can be configured to display a logon banner, which is specified by the administrator, that the user must accept prior to establishing the session.

6.2.6.2 Windows Phone

The behavior for Windows Phone is similar to Windows 8 and Windows 8.1 after an administrator-specified period of inactivity has passed or when a user explicitly chooses to lock the device by pressing the "lock" button. When the device has transitioned to the lock state, it will display either a photo which was selected by the user along with any notifications, or a blank screen. When the phone transitions to "standby" and the Glance setting is enabled, Windows Phone will display that time and notifications received during the first fifteen idle minutes.

As part of establishing the interactive logon session, Windows can be configured to display a logon banner, which is specified by the administrator, that the user must accept prior to establishing the session.

6.2.6.3 SFR Mapping

The **TOE Access** function satisfies the following SFRs:

- **FTA_SSL_EXT.1**: Windows 8.1 and Windows Phone will transition to a locked state when there is an administrator-specified period of inactivity or when the user explicitly locks the device.
- **FTA_WSE_EXT.1**: An authorized administrator can specify which Wi-Fi networks to connect to, as specified in FMT_SMF.1.
- **FTA_TAB.1**: An authorized administrator can define and modify a banner that will be displayed prior to allowing a user to logon.

6.2.7 Trusted Path / Channels

Windows Store applications used the HttpClient interface to establish a secure HTTPS/TLS channel. Windows Store applications do not have access to low level interfaces to perform TLS, the HttpClient interface supports performing TLS in the context of an HTTPS connection by passing a HTTPS Uniform Resource Identifier (URI) to the HttpClient constructor. When a HTTPS URI is used then TLS will be used when establishing the HTTP connection. Mobile Device Managers use HTTPS/TLS: the mobile device authenticates against the MDM to check the identity of the MDM service, and the MDM authenticates the client to ensure the identity of the client device.

Third party VPN Windows Store applications use the Windows.Networking.Vpn interface to establish an IPsec VPN secure channel.

Windows implements IEEE 802.11-2012, IEEE 802.1X and EAP-TLS to provide authenticated wireless networking sessions when requested by the user.

6.2.7.1 The specific details for each protocol are described in section Networking

Windows has a native implementation of IEEE 802.11-2012 to provide secure wireless local area networking (Wi-Fi). Windows uses PRF-384 in WPA2 Wi-Fi sessions and generates AES 128-bit keys using the Windows RBG. Windows complies with the IEEE 802.11-2012 standard and interoperates with other devices that implement the standard. TOE devices have received WPA2 certification, both Enterprise and Personal, and Wi-Fi CERTIFIED Interoperability Certificates from the Wi-Fi Alliance:

- Surface 3 (the Marvell 8897 adapter is also certified)
- Lumia 635
- Lumia 830

Windows implements key wrapping and unwrapping according to the NIST SP 800-38F specification (the "KW" mode) and so unwraps the Wi-Fi Group Temporal Key (GTK) which was sent by the access point. Because the GTK was protected by AES Key Wrap when it was delivered in an EAPOL-Key frame, the GTK is not exposed to the network.

Network Protocols.

To summarize the **Trusted Path / Channel** function satisfies this SFR:

- **FPT_ITC_EXT.1**: Windows provides several trusted network channels that protect data in transit from disclosure, provide data integrity, and endpoint identification that is used by 802.11-2012, 802.1X, EAP-TLS, TLS, HTTPS and IPsec.

7 Protection Profile Conformance Claim

This section provides the protection profile conformance claim and supporting justifications and rationale.

7.1 Rationale for Conformance to Protection Profile

This Security Target is in strict compliance with the Protection Profile for Mobile Device Fundamentals, version 1.1, February 12, 2014 (MDF PP).

For all of the content incorporated from the protection profile, the corresponding rationale in that protection profile remains applicable to demonstrate the correspondence between the TOE security functional requirements and TOE security objectives.

The requirements in the Protection Profile for Mobile Device Fundamentals are assumed to represent a complete set of requirements that serve to address any interdependencies. Given that all of the functional requirements in the MDF PP have been copied into this security target, the dependency analysis for those requirements is not reproduced here.

8 Rationale for Modifications to the Security Requirements

This section provides a rationale that describes how the Security Target reproduced the security functional requirements and security assurance requirements from the protection profile.

8.1 Functional Requirements

This Security Target includes security functional requirements (SFRs) that can be mapped to SFRs found in the protection profile along with SFRs that describe additional features and capabilities. The mapping from protection profile SFRs to security target SFRs along with rationale for operations is presented in **Table 8-1 Rationale for Operations**. SFR operations left incomplete in the protection profile have been completed in this security target and are identified within each SFR in section 5.1 TOE Security Functional Requirements.

Table 8-1 Rationale for Operations

MDF PP Requirement	ST Requirement	Operation & Rationale
FCS_CKM.1(1)	FCS_CKM.1(ASYM KA)	A selection which is allowed by the PP.
FCS_CKM.1(2)	FCS_CKM.1(ASYM AU)	A selection which is allowed by the PP.
FCS_CKM.1(3)	FCS_CKM.1(WLAN)	Copied from the PP without changes.
FCS_CKM.2	FCS_CKM.2	Copied from the PP without changes.
FCS_CKM_EXT.1	FCS_CKM_EXT.1	A selection which is allowed by the PP.
FCS_CKM_EXT.2	FCS_CKM_EXT.2(128) FCS_CKM_EXT.2(256)	Iterated and made a selection which is allowed by the PP.
FCS_CKM_EXT.3	FCS_CKM_EXT.3	Two selections which are allowed by the PP.
FCS_CKM_EXT.4	FCS_CKM_EXT.4	A selection which is allowed by the PP.
FCS_CKM_EXT.5	FCS_CKM_EXT.5	A selection which is allowed by the PP.
FCS_CKM_EXT.6	FCS_CKM_EXT.6	Copied from the PP without changes.
FCS_COP.1(1)	FCS_COP.1(SYM)	Two selections which are allowed by the PP.
FCS_COP.1(2)	FCS_COP.1(HASH)	Two selections which are allowed by the PP.
FCS_COP.1(3)	FCS_COP.1(SIGN)	A selection which is allowed by the PP.
FCS_COP.1(4)	FCS_COP.1(HMAC)	Three selections which are allowed by the PP.
FCS_COP.1(5)	FCS_COP.1(PBKD)	Two selections which are allowed by the PP.
FCS_IV_EXT.1	FCS_IV_EXT.1	Copied from the PP without changes.
FCS_RBG_EXT.1	FCS_RBG_EXT.1	Three selections which are allowed by the PP.
FCS_SRV_EXT.1	FCS_SRV_EXT.1	A selection which is allowed by the

MDF PP Requirement	ST Requirement	Operation & Rationale
		PP and refinements to switch to the SFR labels used in the security target.
FCS_STG_EXT.1	FCS_STG_EXT.1	Five selections which are allowed by the PP.
FCS_STG_EXT.2	FCS_STG_EXT.2	Three selections which are allowed by the PP.
FCS_STG_EXT.3	FCS_STG_EXT.3	A selection which is allowed by the PP.
FCS_TLS_EXT.1	FCS_TLS_EXT.1	Three selections which are allowed by the PP.
FCS_TLS_EXT.2	FCS_TLS_EXT.2	Two selections which are allowed by the PP.
FCS_HTTPS_EXT.1	FCS_HTTPS_EXT.1	Copied from the PP without changes.
FDP_ACF_EXT.1	FDP_ACF_EXT.1	Copied from the PP without changes.
FDP_DAR_EXT.1	FDP_DAR_EXT.1(128) FDP_DAR_EXT.1(256)	Iterated and made a selection which is allowed by the PP.
FDP_DAR_EXT.2	FDP_DAR_EXT.2	Copied from the PP without changes.
FDP_STG_EXT.1	FDP_STG_EXT.1	Copied from the PP without changes.
FIA_AFL_EXT.1	FIA_AFL_EXT.1	Two assignments and a selection which is allowed by the PP.
FIA_BLT_EXT.1	FIA_BLT_EXT.1	Copied from the PP without changes.
FIA_PAE_EXT.1	FIA_PAE_EXT.1	Copied from the PP without changes.
FIA_PMG_EXT.1	FIA_PMG_EXT.1	An assignment and a selection which is allowed by the PP.
FIA_TRT_EXT.1	FIA_TRT_EXT.1	A selection which is allowed by the PP.
FIA_UAU.7	FIA_UAU.7	Copied from the PP without changes.
FIA_UAU_EXT.1	FIA_UAU_EXT.1	Copied from the PP without changes.
FIA_UAU_EXT.2	FIA_UAU_EXT.2	A selection which is allowed by the PP.
FIA_UAU_EXT.3	FIA_UAU_EXT.3	A selection which is allowed by the PP.
FIA_X509_EXT.1	FIA_X509_EXT.1	A selection which is allowed by the PP.
FIA_X509_EXT.2	FIA_X509_EXT.2	Four selections which are allowed by the PP.
FIA_X509_EXT.3	FIA_X509_EXT.3	Copied from the PP without changes.
FMT_MOF.1(1)	FMT_MOF.1(USER)	Multiple selections which are allowed by the PP.
FMT_MOF.1(2)	FMT_MOF.1(ORG)	Multiple selections which are allowed by the PP.
FMT_SMF.1	FMT_SMF.1	Multiple selections, assignments, and refinements which are allowed by the PP.
FMT_SMF_EXT.1	FMT_SMF_EXT.1	Three selections which are allowed by the PP.

MDF PP Requirement	ST Requirement	Operation & Rationale
FPT_AEX_EXT.1	FPT_AEX_EXT.1	Copied from the PP without changes.
FPT_AEX_EXT.2	FPT_AEX_EXT.2	Copied from the PP without changes.
FPT_AEX_EXT.3	FPT_AEX_EXT.3	Copied from the PP without changes.
FPT_AEX_EXT.4	FPT_AEX_EXT.4	Copied from the PP without changes.
FPT_KST_EXT.1(1)	FPT_KST_EXT.1	Copied from the PP without changes.
FPT_KST_EXT.2	FPT_KST_EXT.2	Copied from the PP without changes.
FPT_KST_EXT.3	FPT_KST_EXT.3	Copied from the PP without changes.
FPT_NOT_EXT.1	FPT_NOT_EXT.1	Two selections which are allowed by the PP.
FPT_STM.1	FPT_STM.1	Copied from the PP without changes.
FPT_TST_EXT.1	FPT_TST_EXT.1	Copied from the PP without changes.
FPT_TST_EXT.2	FPT_TST_EXT.2	Three selections which are allowed by the PP.
FPT_TUD_EXT.1	FPT_TUD_EXT.1	Copied from the PP without changes.
FPT_TUD_EXT.2	FPT_TUD_EXT.2	Three selections which are allowed by the PP.
FTA_SSL_EXT.1	FTA_SSL_EXT.1	Assignment allowed by the PP.
FTA_WSE_EXT.1	FTA_WSE_EXT.1	Copied from the PP without changes.
FTA_TAB.1	FTA_TAB.1	Copied from the PP without changes.
FTP_ITC_EXT.1	FTP_ITC_EXT.1	A selection and assignment which are allowed by the PP.

8.2 Security Assurance Requirements

The statement of security assurance requirements (SARs) found in section 5.2 TOE Security Assurance Requirements, is in strict conformance with the Protection Profile for Mobile Device Fundamentals.

8.3 Rationale for the TOE Summary Specification

This section, in conjunction with section 6, the TOE Summary Specification (TSS), provides evidence that the security functions are suitable to meet the TOE security requirements.

Each subsection in section 6, TOE Security Functions (TSFs), describes a Security Function (SF) of the TOE. Each description is followed with rationale that indicates which requirements are satisfied by aspects of the corresponding SF. The set of security functions work together to satisfy all of the functional requirements. Furthermore, all the security functions are necessary in order for the TSF to provide the required security functionality.

The set of security functions work together to provide all of the security requirements as indicated in **Table 8-2**. The security functions described in the TOE Summary Specification and listed in the tables below are all necessary for the required security functionality in the TSF.

Table 8-2 Requirement to Security Function Correspondence

Requirement	Audit	Cryptographic Protection	User Data Protection	I & A	Security Management	TSF Protection	Resource Utilization	TOE Access	Trusted Path / Channel
FCS_CKM.1(ASYM KA)		X							
FCS_CKM.1(ASYM AU)		X							
FCS_CKM.1(WLAN)		X							
FCS_CKM.2		X							
FCS_CKM_EXT.1		X							
FCS_CKM_EXT.2(128)		X							
FCS_CKM_EXT.2(256)		X							
FCS_CKM_EXT.3		X							
FCS_CKM_EXT.4		X							
FCS_CKM_EXT.5		X							
FCS_CKM_EXT.6		X							
FCS_COP.1(SYM)		X							
FCS_COP.1(HASH)		X							
FCS_COP.1(SIGN)		X							
FCS_COP.1(HMAC)		X							
FCS_COP.1(PBKD)		X							
FCS_IV_EXT.1		X							
FCS_RBG_EXT.1		X							
FCS_SRV_EXT.1		X							
FCS_STG_EXT.1		X							
FCS_STG_EXT.2		X							
FCS_STG_EXT.3		X							
FCS_TLS_EXT.1		X							
FCS_TLS_EXT.2		X							
FCS_HTTPS_EXT.1		X							
FDP_ACF_EXT.1			X						
FDP_DAR_EXT.1(128)			X						
FDP_DAR_EXT.1(256)			X						
FDP_DAR_EXT.2			X						
FDP_STG_EXT.1			X						
FIA_AFL_EXT.1				X					
FIA_BLT_EXT.1				X					
FIA_PAE_EXT.1				X					
FIA_PMG_EXT.1				X					
FIA_TRT_EXT.1				X					
FIA_UAU.7				X					

Requirement	Audit	Cryptographic Protection	User Data Protection	I & A	Security Management	TSF Protection	Resource Utilization	TOE Access	Trusted Path / Channel
FIA_UAU_EXT.1				X					
FIA_UAU_EXT.2				X					
FIA_UAU_EXT.3				X					
FIA_X509_EXT.1				X					
FIA_X509_EXT.2				X					
FIA_X509_EXT.3				X					
FMT_MOF.1(USER)					X				
FMT_MOF.1(ORG)					X				
FMT_SMF.1					X				
FMT_SMF_EXT.1					X				
FPT_AEX_EXT.1						X			
FPT_AEX_EXT.2						X			
FPT_AEX_EXT.3						X			
FPT_AEX_EXT.4						X			
FPT_KST_EXT.1						X			
FPT_KST_EXT.2						X			
FPT_KST_EXT.3						X			
FPT_NOT_EXT.1						X			
FPT_STM.1						X			
FPT_TST_EXT.1						X			
FPT_TST_EXT.2						X			
FPT_TUD_EXT.1						X			
FPT_TUD_EXT.2						X			
FTA_SSL_EXT.1								X	
FTA_WSE_EXT.1								X	
FTA_TAB.1								X	
FTP_ITC_EXT.1									X

9 Appendix A: List of Abbreviations

Abbreviation	Meaning
3DES	Triple DES
ACE	Access Control Entry
ACL	Access Control List
ACP	Access Control Policy
AD	Active Directory
ADAM	Active Directory Application Mode
AES	Advanced Encryption Standard
AGD	Administrator Guidance Document
AH	Authentication Header
ALPC	Advanced Local Process Communication
ANSI	American National Standards Institute
API	Application Programming Interface
APIC	Advanced Programmable Interrupt Controller
BTG	BitLocker To Go
CA	Certificate Authority
CBAC	Claims Basic Access Control, see DYN
CBC	Cipher Block Chaining
CC	Common Criteria
CD-ROM	Compact Disk Read Only Memory
CIFS	Common Internet File System
CIMCPP	Certificate Issuing and Management Components For Basic Robustness Environments Protection Profile, Version 1.0, April 27, 2009
CM	Configuration Management; Control Management
COM	Component Object Model
CP	Content Provider
CPU	Central Processing Unit
CRL	Certificate Revocation List
CryptoAPI	Cryptographic API
CSP	Cryptographic Service Provider
DAC	Discretionary Access Control
DACL	Discretionary Access Control List
DC	Domain Controller
DEP	Data Execution Prevention
DES	Data Encryption Standard
DH	Diffie-Hellman
DHCP	Dynamic Host Configuration Protocol
DFS	Distributed File System
DMA	Direct Memory Access
DNS	Domain Name System
DS	Directory Service
DSA	Digital Signature Algorithm

DYN	Dynamic Access Control
EAL	Evaluation Assurance Level
ECB	Electronic Code Book
EFS	Encrypting File System
ESP	Encapsulating Security Protocol
FEK	File Encryption Key
FIPS	Federal Information Processing Standard
FRS	File Replication Service
FSMO	Flexible Single Master Operation
FTP	File Transfer Protocol
FVE	Full Volume Encryption
GB	Gigabyte
GC	Global Catalog
GHz	Gigahertz
GPC	Group Policy Container
GPO	Group Policy Object
GPOSPP	US Government Protection Profile for General-Purpose Operating System in a Networked Environment
GPT	Group Policy Template
GPT	GUID Partition Table
GUI	Graphical User Interface
GUID	Globally Unique Identifiers
HTTP	Hypertext Transfer Protocol
HTTPS	Secure HTTP
I/O	Input / Output
I&A	Identification and Authentication
IA	Information Assurance
ICF	Internet Connection Firewall
ICMP	Internet Control Message Protocol
ICS	Internet Connection Sharing
ID	Identification
IDE	Integrated Drive Electronics
IETF	Internet Engineering Task Force
IFS	Installable File System
IIS	Internet Information Services
IKE	Internet Key Exchange
IP	Internet Protocol
IPv4	IP Version 4
IPv6	IP Version 6
IPC	Inter-process Communication
IPI	Inter-process Interrupt
IPsec	IP Security
ISAPI	Internet Server API
IT	Information Technology
KDC	Key Distribution Center
LAN	Local Area Network

LDAP	Lightweight Directory Access Protocol
LPC	Local Procedure Call
LSA	Local Security Authority
LSASS	LSA Subsystem Service
LUA	Least-privilege User Account
MAC	Message Authentication Code
MB	Megabyte
MMC	Microsoft Management Console
MSR	Model Specific Register
NAC	(Cisco) Network Admission Control
NAP	Network Access Protection
NAT	Network Address Translation
NIC	Network Interface Card
NIST	National Institute of Standards and Technology
NLB	Network Load Balancing
NMI	Non-maskable Interrupt
NTFS	New Technology File System
NTLM	New Technology LAN Manager
OS	Operating System
PAE	Physical Address Extension
PC/SC	Personal Computer/Smart Card
PIN	Personal Identification Number
PKCS	Public Key Certificate Standard
PKI	Public Key Infrastructure
PP	Protection Profile
RADIUS	Remote Authentication Dial In Service
RAID	Redundant Array of Independent Disks
RAM	Random Access Memory
RAS	Remote Access Service
RC4	Rivest's Cipher 4
RID	Relative Identifier
RNG	Random Number Generator
RPC	Remote Procedure Call
RSA	Rivest, Shamir and Adleman
RSASSA	RSA Signature Scheme with Appendix
SA	Security Association
SACL	System Access Control List
SAM	Security Assurance Measure
SAML	Security Assertion Markup Language
SAR	Security Assurance Requirement
SAS	Secure Attention Sequence
SD	Security Descriptor
SHA	Secure Hash Algorithm
SID	Security Identifier
SIP	Session Initiation Protocol
SIPI	Startup IPI

SF	Security Functions
SFP	Security Functional Policy
SFR	Security Functional Requirement
SMB	Server Message Block
SMI	System Management Interrupt
SMTP	Simple Mail Transport Protocol
SP	Service Pack
SPI	Security Parameters Index
SPI	Stateful Packet Inspection
SRM	Security Reference Monitor
SSL	Secure Sockets Layer
SSP	Security Support Providers
SSPI	Security Support Provider Interface
SPS	Storage Primary Seed
SRK	Storage Root Key
ST	Security Target
SYSVOL	System Volume
TCP	Transmission Control Protocol
TDI	Transport Driver Interface
TLS	Transport Layer Security
TOE	Target of Evaluation
TPM	Trusted Platform Module
TSC	TOE Scope of Control
TSF	TOE Security Functions
TSS	TOE Summary Specification
UART	Universal Asynchronous Receiver / Transmitter
UI	User Interface
UID	User Identifier
UNC	Universal Naming Convention
US	United States
UPN	User Principal Name
URL	Uniform Resource Locator
USB	Universal Serial Bus
USN	Update Sequence Number
v5	Version 5
VDS	Virtual Disk Service
VPN	Virtual Private Network
VSS	Volume Shadow Copy Service
WAN	Wide Area Network
WCF	Windows Communications Framework
WebDAV	Web Document Authoring and Versioning
WebSSO	Web Single Sign On
WDM	Windows Driver Model
WIF	Windows Identity Framework
WMI	Windows Management Instrumentation
WSC	Windows Security Center

WU	Windows Update
WSDL	Web Service Description Language
WWW	World-Wide Web
X64	A 64-bit instruction set architecture
X86	A 32-bit instruction set architecture

10 Appendix B: Interfaces

This section is a list of APIs used during testing of Windows 8.1 and Windows Phone 8.1.

API	Description
CryptographicBuffer.GenerateRandom	http://msdn.microsoft.com/en-us/library/windows/apps/xaml/windows.security.cryptography.cryptographicbuffer.generaterandom.aspx
CryptographicBuffer.GenerateRandomNumber	http://msdn.microsoft.com/en-us/library/windows/apps/xaml/windows.security.cryptography.cryptographicbuffer.generaterandomnumber.aspx
CryptographicEngine.Encrypt	http://msdn.microsoft.com/en-us/library/windows/apps/xaml/windows.security.cryptography.core.cryptographicengine.encrypt.aspx
CryptographicEngine.Decrypt	http://msdn.microsoft.com/en-us/library/windows/apps/xaml/windows.security.cryptography.core.cryptographicengine.decrypt.aspx
HashAlgorithmProvider.CreateHash	http://msdn.microsoft.com/en-us/library/windows/apps/xaml/windows.security.cryptography.core.hashalgorithmprovider.createhash.aspx
HashAlgorithmProvider.HashData	http://msdn.microsoft.com/en-us/library/windows/apps/xaml/windows.security.cryptography.core.hashalgorithmprovider.hashdata.aspx
CryptographicEngine.Sign	http://msdn.microsoft.com/en-us/library/windows/apps/xaml/windows.security.cryptography.core.cryptographicengine.sign.aspx
CryptographicEngine.VerifySignature	http://msdn.microsoft.com/en-us/library/windows/apps/xaml/windows.security.cryptography.core.cryptographicengine.verifysignature.aspx
KeyDerivationParameters.BuildForPbkdf2	http://msdn.microsoft.com/en-us/library/windows/apps/windows.security.cryptography.core.keyderivationparameters.buildforpbkdf2.aspx
AsymmetricKeyAlgorithmProvider.CreateKeyPair	http://msdn.microsoft.com/en-us/library/windows/apps/xaml/windows.security.cryptography.core.asymmetrickeyalgorithmprovider.createkeypair.aspx
CryptographicEngine.SignAsync	http://msdn.microsoft.com/en-us/library/windows/apps/xaml/windows.security.cryptography.core.cryptographicengine.signasync.aspx
CryptographicEngine.SignHashedData	http://msdn.microsoft.com/en-us/library/windows/apps/xaml/windows.security.cryptography.core.cryptographicengine.signhasheddata.aspx
CryptographicEngine.SignHashedDataAsync	http://msdn.microsoft.com/en-us/library/windows/apps/xaml/windows.security.cryptography.core.cryptographicengine.signhasheddataasync.aspx
CryptographicEngine.VerifySignatureWithHashInput	http://msdn.microsoft.com/en-us/library/windows/apps/xaml/windows.security.cryptography.core.cryptographicengine.verifysignaturewithhashinput.aspx
AsymmetricKeyAlgorithmPro	http://msdn.microsoft.com/en-

vider.ImportKeyPair	us/library/windows/apps/windows.security.cryptography.core.asymmetrickeyalgorithmprovider.importkeypair.aspx
CertificateEnrollmentManager.ImportPfxDataAsync	http://msdn.microsoft.com/en-us/library/windows/apps/windows.security.cryptography.certificates.certificateenrollmentmanager.importpfxdataasync.aspx
CmsDetachedSignature.GenerateSignatureAsync	http://msdn.microsoft.com/en-us/library/windows/apps/dn298272.aspx
CmsAttachedSignature.GenerateSignatureAsync	http://msdn.microsoft.com/en-us/library/windows/apps/dn298266.aspx
HttpClient	http://msdn.microsoft.com/en-us/library/windows/apps/windows.web.http.httpclient.aspx
Windows.Networking.Vpn	http://msdn.microsoft.com/en-us/library/windows/apps/windows.networking.vpn.aspxn
Certificate.BuildChainAsync	http://msdn.microsoft.com/en-us/library/windows/apps/windows.security.cryptography.certificates.certificate.buildchainasync.aspx
CertificateChain.Validate	http://msdn.microsoft.com/en-us/library/windows/apps/dn279161.aspx
Windows.Security.Cryptography.DataProtection	http://msdn.microsoft.com/en-us/library/windows/apps/xaml/windows.security.cryptography.dataprotection.aspx

11 Appendix C: Analysis of Special Publication 800-56A and 800-56B

11.1 Special Publication 800-56A

The source document is NIST Special Publication 800-56A, "*Recommendation for Pair-Wise Key Establishment Schemes Using Discrete Logarithm Cryptography*".

11.1.1 NIST SP 800-56A Sections

11.1.1.1 Sections 1 – 3

The first three (3) sections do not specify any "shall", "shall not", "should" or "should not" statements. For completeness, they are:

1. Introduction
2. Scope and Purpose
3. Definitions, Symbols and Abbreviations

11.1.1.2 Section 4 Key Establishment Schemes Overview

This section is merely a high-level explanation of what key establishment is. Section 4.1 contains the statement "shall" and is listed here for completeness.

4.1 Key Agreement Preparations by an Owner

"Shall not", "should", and "should not" Options Implemented by TOE

N/A

Rationale for Implementation of "shall not" or "should not"

N/A

Omission of Functionality Related to "shall" or "should"

N/A

4.2 Key Agreement Process
4.3 DLC-based Key Transport Process

11.1.1.3 Section 5 Cryptographic Elements

5.1 Cryptographic Hash Functions

"Shall not", "should", and "should not" Options Implemented by TOE

N/A

Rationale for Implementation of “shall not” or “should not”

N/A

Omission of Functionality Related to "shall" or “should”

N/A

5.2 Message Authentication Code (MAC) Algorithm

“Shall not”, “should”, and “should not” Options Implemented by TOE

N/A

Rationale for Implementation of “shall not” or “should not”

N/A

Omission of Functionality Related to "shall" or “should”

N/A

5.2.1 MacTag Computation

“Shall not”, “should”, and “should not” Options Implemented by TOE

N/A

Rationale for Implementation of “shall not” or “should not”

N/A

Omission of Functionality Related to "shall" or “should”

N/A

5.2.2 MacTag Checking

“Shall not”, “should”, and “should not” Options Implemented by TOE

N/A

Rationale for Implementation of “shall not” or “should not”

N/A

Omission of Functionality Related to "shall" or “should”

N/A

5.2.3 Implementation Validation Message

“Shall not”, “should”, and “should not” Options Implemented by TOE

N/A

Rationale for Implementation of “shall not” or “should not”

N/A

Omission of Functionality Related to "shall" or “should”

N/A

5.3 Random Number Generation

“Shall not”, “should”, and “should not” Options Implemented by TOE

N/A

Rationale for Implementation of “shall not” or “should not”

N/A

Omission of Functionality Related to "shall" or “should”

N/A

5.4 Nonces

“Shall not”, “should”, and “should not” Options Implemented by TOE

The TOE implements random nonces.

Rationale for Implementation of “shall not” or “should not”

N/A

Omission of Functionality Related to "shall" or “should”

N/A

5.5 Domain Parameters

“Shall not”, “should”, and “should not” Options Implemented by TOE

N/A

Rationale for Implementation of “shall not” or “should not”

N/A

Omission of Functionality Related to "shall" or “should”

N/A

5.5.1 Domain Parameter Generation

This is a section header.

5.5.1.1 FFC Domain Parameter Generation

"Shall not", "should", and "should not" Options Implemented by TOE

The "should" statement is:

"If the appropriate security strength does not have an FFC parameter set, then Elliptic Curve Cryptography **should** be used (see Section 5.5.1.2)."

The "should" statement only applies to user behavior, which is outside the scope of the TOE.

Rationale for Implementation of "shall not" or "should not"

N/A

Omission of Functionality Related to "shall" or "should"

N/A

5.5.1.2 ECC Domain Parameter Generation

"Shall not", "should", and "should not" Options Implemented by TOE

N/A

Rationale for Implementation of "shall not" or "should not"

N/A

Omission of Functionality Related to "shall" or "should"

N/A

5.5.2 Assurances of Domain Parameter Validity

"Shall not", "should", and "should not" Options Implemented by TOE

The "should" statement is:

"The application performing the key establishment on behalf of the party **should** determine whether or not to allow key establishment based upon the method(s) of assurance that was used."

The "should" statement only applies to an application, which is outside the scope of the TOE.

Rationale for Implementation of "shall not" or "should not"

N/A

Omission of Functionality Related to "shall" or "should"

N/A

5.5.3 Domain Parameter Management

"Shall not", "should", and "should not" Options Implemented by TOE

N/A

Rationale for Implementation of "shall not" or "should not"

N/A

Omission of Functionality Related to "shall" or “should”

N/A

5.6 Private and Public Keys

This is a section header with a brief statement as such.

5.6.1 Private/Public Key Pair Generation

This is a section header.

5.6.1.1 FFC Key Pair Generation

“Shall not”, “should”, and “should not” Options Implemented by TOE

N/A

Rationale for Implementation of “shall not” or “should not”

N/A

Omission of Functionality Related to "shall" or “should”

N/A

5.6.1.2 ECC Key Pair Generation

“Shall not”, “should”, and “should not” Options Implemented by TOE

N/A

Rationale for Implementation of “shall not” or “should not”

N/A

Omission of Functionality Related to "shall" or “should”

N/A

5.6.2 Assurances of the Arithmetic Validity of a Public Key

“Shall not”, “should”, and “should not” Options Implemented by TOE

The “should” statement is:

> “The application performing the key establishment on behalf of the owner and recipient **should** determine whether or not to allow key establishment based upon the method(s) of assurance that was used.”

The “should” statement only applies to an application, which is outside the scope of the TOE.

Rationale for Implementation of “shall not” or “should not”

N/A

Omission of Functionality Related to "shall" or “should”

N/A

5.6.2.1 Owner Assurances of Static Public Key Validity

“Shall not”, “should”, and “should not” Options Implemented by TOE

The “should” statement is:

“The application performing the key establishment on behalf of the owner **should** determine whether or not to allow key establishment based upon the method(s) of assurance that was used.”

The “should” statement only applies to an application, which is outside the scope of the TOE.

Rationale for Implementation of “shall not” or “should not”

N/A

Omission of Functionality Related to "shall" or “should”

N/A

5.6.2.2 Recipient Assurances of Static Public Key Validity

“Shall not”, “should”, and “should not” Options Implemented by TOE

The “should” statement is:

“The application performing the key establishment on behalf of the recipient **should** determine whether or not to allow key establishment based upon the method(s) of assurance that was used.”

The “should” statement only applies to an application, which is outside the scope of the TOE.

Rationale for Implementation of “shall not” or “should not”

N/A

Omission of Functionality Related to "shall" or “should”

N/A

5.6.2.3 Recipient Assurances of Ephemeral Public Key Validity

“Shall not”, “should”, and “should not” Options Implemented by TOE

The “should” statement is:

“The application performing the key establishment on behalf of the recipient **should** determine whether or not to allow key establishment based upon the method(s) of assurance that was used.”

The “should” statement only applies to an application, which is outside the scope of the TOE.

Rationale for Implementation of “shall not” or “should not”

N/A

Omission of Functionality Related to "shall" or “should”

N/A

5.6.2.4 FFC Full Public Key Validation Routine – Unimplemented

Note: Full public key validation is one of several options available for assurances of the arithmetic validity of public keys. Microsoft chose not to implement it in the TOE.

5.6.2.5 ECC Full Public Key Validation Routine – Unimplemented

Note: Full public key validation is one of several options available for assurances of the arithmetic validity of public keys. Microsoft chose not to implement it in the TOE.

5.6.2.6 ECC Partial Public Key Validation Routine

"Shall not", "should", and "should not" Options Implemented by TOE

N/A

Rationale for Implementation of "shall not" or "should not"

N/A

Omission of Functionality Related to "shall" or "should"

N/A

5.6.3 Assurances of the Possession of a Static Private Key

This section and all subsections concern Owner and Recipient user behavior, which is outside the scope of the TOE.

5.6.3.1 Owner Assurances of Possession of a Static Private Key
5.6.3.2 Recipient Assurance of Owner's Possession of a Static Private Key
5.6.3.2.1 Recipient Obtains Assurance through a Trusted Third Party
5.6.3.2.2 Recipient Obtains Assurance Directly from the Claimed Owner
5.6.4 Key Pair Management

This is a section header.

5.6.4.1 Common Requirements on Static and Ephemeral Key Pairs

"Shall not", "should", and "should not" Options Implemented by TOE

N/A

Rationale for Implementation of "shall not" or "should not"

N/A

Omission of Functionality Related to "shall" or "should"

N/A

5.6.4.2 Specific Requirements on Static Key Pairs

"Shall not", "should", and "should not" Options Implemented by TOE

The "should" statement is:

"The application performing the key establishment on behalf of the recipient **should** determine whether or not to allow key establishment based upon the method(s) of assurance that was used."

The "should" statement only applies to an application, which is outside the scope of the TOE.

Rationale for Implementation of "shall not" or "should not"

N/A

Omission of Functionality Related to "shall" or "should"

N/A

5.6.4.3 Specific Requirements on Ephemeral Key Pairs

"Shall not", "should", and "should not" Options Implemented by TOE

1. The first instance of the word "should" is: "An ephemeral key pair should be generated as close to its time of use as possible." The TOE implements this.
2. The second "should" statement is:

   "The application performing the key establishment on behalf of the recipient **should** determine whether or not to allow key establishment based upon the method(s) of assurance that was used."

   This second instance of the word "should" only applies to an application, which is outside the scope of the TOE.

Rationale for Implementation of "shall not" or "should not"

N/A

Omission of Functionality Related to "shall" or "should"

N/A

5.7 DLC Primitives

"Shall not", "should", and "should not" Options Implemented by TOE

N/A

Rationale for Implementation of "shall not" or "should not"

N/A

Omission of Functionality Related to "shall" or "should"

N/A

5.7.1 Diffie-Hellman Primitives

This is a section header.

5.7.1.1 Finite Field Cryptography Diffie-Hellman (FFC DH) Primitive

"Shall not", "should", and "should not" Options Implemented by TOE

N/A

Rationale for Implementation of "shall not" or "should not"

N/A

Omission of Functionality Related to "shall" or "should"

N/A

5.7.1.2 Elliptic Curve Cryptography Cofactor Diffie-Hellman (ECC CDH) Primitive

"Shall not", "should", and "should not" Options Implemented by TOE

N/A

Rationale for Implementation of "shall not" or "should not"

N/A

Omission of Functionality Related to "shall" or "should"

N/A

5.7.2 MQV Primitives -- Unimplemented

This section and all subsections (5.7.2 through 5.7.2.3.2) are MQV primitives. MQV is only one of several options available for key establishment schemes. Microsoft chose not to implement MQV primitives in the TOE.

5.7.2.1 Finite Field Cryptography MQV (FFC MQV) Primitive – Unimplemented
5.7.2.1.1 MQV2 Form of the FFC MQV Primitive – Unimplemented
5.7.2.1.2 MQV1 Form of the FFC MQV Primitive – Unimplemented
5.7.2.2 ECC MQV Associate Value Function – Unimplemented
5.7.2.3 Elliptic Curve Cryptography MQV (ECC MQV) Primitive – Unimplemented
5.7.2.3.1 Full MQV Form of the ECC MQV Primitive – Unimplemented
5.7.2.3.2 One-Pass Form of the ECC MQV Primitive – Unimplemented

5.8 Key Derivation Functions for Key Agreement Schemes

"Shall not", "should", and "should not" Options Implemented by TOE

N/A

Rationale for Implementation of “shall not” or “should not”

N/A

Omission of Functionality Related to "shall" or “should”

N/A

5.8.1 Concatenation Key Derivation Function (Approved Alternative 1)

“Shall not”, “should”, and “should not” Options Implemented by TOE

N/A

Rationale for Implementation of “shall not” or “should not”

N/A

Omission of Functionality Related to "shall" or “should”

N/A

5.8.2 ASN.1 Key Derivation Function (Approved Alternative 2) – Unimplemented

11.1.1.4 Section 6 Key Agreement

This section is an explanation of three (3) categories of key agreement schemes as detailed in sections 6.1, 6.2, and 6.3. Under each category, there are one or more subcategories that are classified by static keys usage. SP 800-56A does not mandate the implementation of all categories and subcategories. Microsoft chose to implement a subset of all possible key agreement schemes in the TOE.

“Shall not”, “should”, and “should not” Options Implemented by TOE

N/A

Rationale for Implementation of “shall not” or “should not”

N/A

Omission of Functionality Related to "shall" or “should”

The “should” statement is:

“Key confirmation may be added to many of these schemes to provide assurance that the participants share the same keying material; see Section 8 for details on key confirmation. Each party **should** have such assurance.”

Microsoft chose not to implement the option of key confirmation in the TOE.

6.1 Schemes Using Two Ephemeral Key Pairs, C(2)

This section is a header with a short explanation of the subcategories.

6.1.1 Each Party Has a Static Key Pair and Generates an Ephemeral Key Pair, C(2, 2) – Unimplemented

This section and all subsections (6.1.1 through 6.1.1.5) are optional. Microsoft chose not to implement them in the TOE.

6.1.1.1 dhHybrid1, C(2, 2, FFC DH) – Unimplemented
6.1.1.2 Full Unified Model, C(2, 2, ECC CDH) – Unimplemented
6.1.1.3 MQV2, C(2, 2, FFC MQV) – Unimplemented
6.1.1.4 Full MQV, C(2, 2, ECC MQV) – Unimplemented
6.1.1.5 Rationale for Choosing a C(2, 2) Scheme – Unimplemented

6.1.2 Each Party Generates an Ephemeral Key Pair; No Static Keys are Used, C(2, 0)

"Shall not", "should", and "should not" Options Implemented by TOE

N/A

Rationale for Implementation of "shall not" or "should not"

N/A

Omission of Functionality Related to "shall" or "should"

N/A

6.1.2.1 dhEphem, C(2, 0, FFC DH)

"Shall not", "should", and "should not" Options Implemented by TOE

N/A

Rationale for Implementation of "shall not" or "should not"

N/A

Omission of Functionality Related to "shall" or "should"

N/A

6.1.2.2 Ephemeral Unified Model, C(2, 0, ECC CDH)

"Shall not", "should", and "should not" Options Implemented by TOE

N/A

Rationale for Implementation of "shall not" or "should not"

N/A

Omission of Functionality Related to "shall" or "should"

N/A

6.1.2.3 Rationale for Choosing a C(2, 0) Scheme

This section only explains the rationale.

6.2 Schemes Using One Ephemeral Key Pair, C(1)

This section is a header with a short explanation of the subcategories.

6.2.1 Initiator Has a Static Key Pair and Generates an Ephemeral Key Pair; Responder Has a Static Key Pair, C(1, 2) – Unimplemented

This section and all subsections (6.2.1 through 6.2.1.5) are optional. Microsoft chose not to implement them in the TOE.

6.2.1.1 dhHybridOneFlow, C(1, 2, FFC DH) – Unimplemented

6.2.1.2 One-Pass Unified Model, C(1, 2, ECC CDH) – Unimplemented

6.2.1.3 MQV1, C(1, 2, FFC MQV) – Unimplemented

6.2.1.4 One-Pass MQV, C(1, 2, ECC MQV) – Unimplemented

6.2.1.5 Rationale for Choosing a C(1, 2) Scheme – Unimplemented

6.2.2 Initiator Generates Only an Ephemeral Key Pair; Responder Has Only a Static Key Pair, C(1, 1)

"Shall not", "should", and "should not" Options Implemented by TOE

N/A

Rationale for Implementation of "shall not" or "should not"

N/A

Omission of Functionality Related to "shall" or "should"

N/A

6.2.2.1 dhOneFlow, C(1, 1, FFC DH)

"Shall not", "should", and "should not" Options Implemented by TOE

N/A

Rationale for Implementation of "shall not" or "should not"

N/A

Omission of Functionality Related to "shall" or "should"

N/A

6.2.2.2 One-Pass Diffie-Hellman, C(1, 1, ECC CDH)

"Shall not", "should", and "should not" Options Implemented by TOE

N/A

Rationale for Implementation of "shall not" or "should not"

N/A

Omission of Functionality Related to "shall" or “should”

N/A

6.2.2.3 Rationale in Choosing a C(1, 1) Scheme

This section only explains the rationale.

6.3 Scheme Using No Ephemeral Key Pairs, C(0, 2)

“Shall not”, “should”, and “should not” Options Implemented by TOE

N/A

Rationale for Implementation of “shall not” or “should not”

N/A

Omission of Functionality Related to "shall" or “should”

N/A

6.3.1 dhStatic, C(0, 2, FFC DH)

“Shall not”, “should”, and “should not” Options Implemented by TOE

N/A

Rationale for Implementation of “shall not” or “should not”

N/A

Omission of Functionality Related to "shall" or “should”

N/A

6.3.2 Static Unified Model, C(0, 2, ECC CDH)

“Shall not”, “should”, and “should not” Options Implemented by TOE

N/A

Rationale for Implementation of “shall not” or “should not”

N/A

Omission of Functionality Related to "shall" or “should”

N/A

6.3.3 Rationale in Choosing a C(0, 2) Scheme

This section only explains the rationale.

11.1.1.5 Section 7 DLC-Based Key Transport

This section was not selected in the ST.

11.1.1.6 Section 8 Key Confirmation

As allowed in Section 6 Key Agreement, Microsoft chose not to implement optional key confirmation in the TOE.

11.1.1.7 Section 9 Key Recovery

"Shall not", "should", and "should not" Options Implemented by TOE

N/A

Rationale for Implementation of "shall not" or "should not"

N/A

Omission of Functionality Related to "shall" or "should"

N/A

11.1.1.8 Section 10 Implementation Validation

The TOE shall be proven to comply with the "shall" statements in this section as evidenced by NIST CMVP FIPS 140-2 validation certificates when they are published on the NIST CMVP Validated FIPS 140-1 and FIPS 140-2 Cryptographic Modules website:

http://csrc.nist.gov/groups/STM/cmvp/documents/140-1/140val-all.htm

"Shall not", "should", and "should not" Options Implemented by TOE

N/A

Rationale for Implementation of "shall not" or "should not"

N/A

Omission of Functionality Related to "shall" or "should"

N/A

11.1.1.9 Appendices A, D, and E (Informative)

These appendices are informative and are included here for completeness.

11.1.1.10 Appendix B: Rationale for Including Identifiers in the KDF Input

This section is explanatory rationale and is included here for completeness.

11.1.1.11 Appendix C: Data Conversions (Normative)

"Shall not", "should", and "should not" Options Implemented by TOE

N/A

Rationale for Implementation of "shall not" or "should not"

N/A

Omission of Functionality Related to "shall" or "should"

N/A

11.1.2 Exceptions

11.1.2.1 TOE-Specific Extensions

There are not any TOE-specific extensions that may impact the security requirements the TOE is to enforce.

11.1.2.2 Additional Processing

There is no processing that is not included in the documents that may impact the security requirements the TOE is to enforce.

11.1.2.3 Alternative Implementations

There are no alternative implementations allowed by the documents that may impact the security requirements the TOE is to enforce.

11.2 Special Publication 800-56B

The source document is NIST Special Publication 800-56, "*Recommendation for Pair-Wise Key Establishment Schemes Using Using Integer Factorization Cryptography*".

11.2.1 NIST SP 800-56B Sections

This standard describes requirements and procedures for key establishment schemes using an asymmetric-based key agreement and key transport scheme based on the RSA algorithm. The FCS_CKM.1(ASYM KA) security functional requirement is applicable only to the generation of the RSA key pair that is subsequently used by key establishment operations. Therefore only the SHALL, SHOULD, SHALL NOT and SHOULD NOT directives that are related to sections of this standard specifying requirements on the actual RSA key generation and associated cryptographic primitives used for RSA key generation are relevant in the assurance activity for FCS_CKM.1(ASYM KA). All other sections in this standard that are not relevant to actual RSA key generation are noted as such.

11.2.1.1 Sections 1 – 3

The first three (3) sections do not specify any relevant requirements. For completeness, they are:

1. Introduction
2. Scope and Purpose
3. Definitions, Symbols and Abbreviations

11.2.1.2 Section 4 Key Establishment Schemes Overview

This section is associated with key establishment processes that are based on using a generated RSA key pair and is not relevant to the actual RSA key pair generation.

4.1 Key Establishment Preparations by an Owner
4.2 Key Agreement Process
4.3 IFC-based Key Transport Process

11.2.1.3 Section 5 Cryptographic Elements

This section describes cryptographic elements associated with RSA key pair generation or using a generated RSA key pair.

5.1 Cryptographic Hash Functions

"Shall not", "should", and "should not" Options Implemented by TOE

N/A

Rationale for Implementation of "shall not" or "should not"

N/A

Omission of Functionality Related to "shall" or "should"

N/A

5.2 Message Authentication Code (MAC) Algorithm

"Shall not", "should", and "should not" Options Implemented by TOE

N/A

Rationale for Implementation of "shall not" or "should not"

N/A

Omission of Functionality Related to "shall" or "should"

N/A

5.3 Random Bit Generation

"Shall not", "should", and "should not" Options Implemented by TOE

N/A

Rationale for Implementation of “shall not” or “should not”

N/A

Omission of Functionality Related to "shall" or “should”

N/A

5.4 Prime Number Generators

“Shall not”, “should”, and “should not” Options Implemented by TOE

N/A

Rationale for Implementation of “shall not” or “should not”

N/A

Omission of Functionality Related to "shall" or “should”

N/A

5.5 Primality Testing Methods

“Shall not”, “should”, and “should not” Options Implemented by TOE

N/A

Rationale for Implementation of “shall not” or “should not”

N/A

Omission of Functionality Related to "shall" or “should”

N/A

5.6 Nonces

“Shall not”, “should”, and “should not” Options Implemented by TOE

The TOE implements random nonces.

Rationale for Implementation of “shall not” or “should not”

N/A

Omission of Functionality Related to "shall" or “should”

N/A

5.7 Symmetric Key-Wrapping Algorithms

This section describes symmetric key-wrapping algorithms that is not relevant to the generation of RSA key pairs and hence is not relevant in assurance activity for FCS_CKM.1(ASYM KA).

5.8 Mask Generation Function (MGF)

This section describes a mechanism for use with the RSA-OAEP based schemes associated with key transport operations that use RSA key pairs and is is not relevant to the actual generation of those key pairs and hence is not relevant in assurance activity for FCS_CKM.1(ASYM KA).

5.8.1 Concatenation Key Derivation Function (Approved Alternative 1)
5.8.2 ASN.1 Key Derivation Function (Approved Alternative 2)

5.9 Key Derivation Functions for Key Establishment Schemes

This section describes a mechanism for deriving shared keying material from a shared secret between entities that use generated RSA key pairs and is not relevant to the actual generation of those key pairs and hence is not relevant in assurance activity for FCS_CKM.1(ASYM KA).

5.9.1 Concatenation Key Derivation Function (Approved Alternative 1)
5.9.2 ASN.1 Key Derivation Function (Approved Alternative 2)

11.2.1.4 Section 6 RSA Key Pairs

This section describes RSA key pair generation, some of which are relevant to RSA key generation and some of which are not relevant. All non-relevant sub-sections are included for completeness.

6.1 General Requirements

"Shall not", "should", and "should not" Options Implemented by TOE

From item (7):
"The owner of the key pair (or agents trusted to act on behalf of the owner) **should** determine that the methods used for obtaining these assurances are sufficient and appropriate to meet the security requirements of the owner's intended application(s)."

The "should" statement only applies to user/application behavior and is not relevant to RSA key pair generation.

From item (8):
The recipient of a public key (or agents trusted to act on behalf of the recipient) **should** determine which method(s) for obtaining these assurances are sufficient and appropriate to meet the security requirements of the owner's intended application(s). The application performing the key establishment on behalf of the recipient **should** determine whether or not to

allow the key establishment, based upon the method(s) used to obtain this assurance. Such knowledge may be explicitly provided to the application in some manner, or may be implicitly provided by the operation of the application itself.

The “should” statements only apply to user/application behavior and are not relevant to RSA key pair generation.

From item (9):
The recipient of a public key (or agents trusted to act on behalf of the recipient) **should** determine that the method used for obtaining this assurance is sufficient and appropriate to meet the security requirements of the recipient’s intended application(s).

The “should” statements only apply to user/application behavior and is not relevant to RSA key pair generation.

Rationale for Implementation of “shall not” or “should not”

N/A

Omission of Functionality Related to "shall" or “should”

Use of “should” statement is described immediately above.

6.2 Criteria for RSA Key Pairs for Key Establishment

This section does not specify any “shall”, “shall not”, “should” or “should not” statements.

6.2.1 Definition of a Key Pair

“Shall not”, “should”, and “should not” Options Implemented by TOE

N/A

Rationale for Implementation of “shall not” or “should not”

N/A

Omission of Functionality Related to "shall" or “should”

N/A

6.2.2 Formats

“Shall not”, “should”, and “should not” Options Implemented by TOE

N/A

Rationale for Implementation of “shall not” or “should not”

N/A

Omission of Functionality Related to "shall" or “should”

N/A

6.2.3 Parameter Length Sets

"Shall not", "should", and "should not" Options Implemented by TOE

N/A

Rationale for Implementation of "shall not" or "should not"

N/A

Omission of Functionality Related to "shall" or "should"

N/A

6.3 RSA Key Pair Generators

"Shall not", "should", and "should not" Options Implemented by TOE

N/A

Rationale for Implementation of "shall not" or "should not"

N/A

Omission of Functionality Related to "shall" or "should"

N/A

6.3.1 RSAKPG1 Family: RSA Key Pair Generation with a Fixed Public Exponent

This section and its subsections do not specify any "shall", "shall not", "should" or "should not" statements.

6.3.1.1 rsakpg1-basic
6.3.1.2 rsakpg1-prime-factor
6.3.1.3 rsakpg1-crt

6.3.2 RSAKPG2 Family: RSA Key Pair Generation with a Random Public Exponent

This section and its subsections do not specify any "shall", "shall not", "should" or "should not" statements.

6.3.2 RSAKPG2 Family: RSA Key Pair Generation with a Random Public Exponent
6.3.2.1 rsakpg2-basic
6.3.2.2 rsakpg2-prime-factor
6.3.2.3 rsakpg2-crt

6.4 Assurances of Validity

This section and its subsections describe assurance of RSA key pair validity that only applies to the owner or recipient of a RSA key pair, which is outside the scope of the TOE.

6.4.1 Assurance of Key Pair Validity
6.4.1.1 General Method for Obtaining Owner Assurance of Key Pair Validity
6.4.1.2 RSAKPV1 Family: RSA Key Pair Validation with a Fixed Exponent
6.4.1.2.1 rsakpv1-basic
6.4.1.2.2 rsakpv1-prime-factor
6.4.1.3 RSAKPV2 Family: RSA Key Pair Validation with a Random Exponent
6.4.1.3.1 rsakpv2-basic
6.4.1.3.2 rsakpv2-prime-factor
6.4.2 Recipient Assurances of Public Key Validity
6.4.2.1 General Method for Obtaining Assurance of Public Key Validity
6.4.2.2 Partial Public Key Validation for RSA

6.5 Assurances of Private Key Possession

This section and its subsections describe owner assurance of private key possession by applications, which is outside the scope of the TOE.

6.5.1 Owner Assurance of Private Key Possession
6.5.2 Recipient Assurance of Owner's Possession of a Private Key
6.5.2.1 Recipient Indirectly Obtains Assurance of Possession Using a Trusted Third Party
6.5.2.2 Recipient Obtains Assurance of Possession Directly from the Claimed Owner

6.6 Key Confirmation

This section and its subsections describe an application process applied by provider and recipient entities that uses their respective RSA key pairs to confirm they have a shared secret, which is outside the scope of the TOE.

6.6.1 Unilateral Key Confirmation for Key Establishment Schemes
6.6.2 Bilateral Key Confirmation for Key Establishment Schemes

6.7 Authentication

This section and its subsections do not specify any "shall", "shall not", "should" or "should not" statements.

11.2.1.5 Section 7 IFC Primitives and Operations

This section and its subsection are concerned with applications establishing keying material for a secret shared between two entities using a RSA key pair, which is outside the scope of the TOE.

7.1 Encryption and Decryption Primitives

7.1.1 RSAEP
7.1.2 RSADP
7.2 Encryption and Decryption Operations
7.2.1 RSA Secret Value Encapsulation (RSASVE)
7.2.1.1 RSASVE Components
7.2.1.2 RSASVE Generate Operation
7.2.1.3 RSASVE Recovery Operation
7.2.2 RSA with Optimal Asymmetric Encryption Padding (RSA-OAEP)
7.2.2.1 RSA-OAEP Components
7.2.2.2 RSA-OAEP Encryption Operation
7.2.2.3 RSA-OAEP Decryption Operation
7.2.3 RSA-based Key-Encapsulation Mechanism with a Key-Wrapping Scheme (RSA-KEM-KWS)
7.2.3.1 RSA-KEM-KWS Components
7.2.3.2 RSA-KEM-KWS Encryption Operation
7.2.3.3 RSA-KEM-KWS Decryption Operation

11.2.1.6 Section 8 Key Agreement Schemes

This section and its subsection are concerned with applications deriving keys based on a secret shared between two entities that was established using a RSA key pair, which is outside the scope of the TOE.

8.1 Common Components for Key Agreement
8.2 The KAS1 Family
8.2.1 KAS1 Family Prerequisites
8.2.2 KAS1-basic
8.2.3 KAS1 Key Confirmation
8.2.3.1 KAS1 Key Confirmation Components
8.2.3.2 KAS1-responder-confirmation
8.2.4 KAS1 Security Properties
8.3 The KAS2 Family
8.3.1 KAS2 Family Prerequisites
8.3.2 KAS2-basic
8.3.3 KAS2 Key Confirmation
8.3.3.1 KAS2 Key Confirmation Components
8.3.3.2 KAS2-responder-confirmation
8.3.3.3 KAS2-initiator-confirmation

8.3.3.4 KAS2-bilateral-confirmation
8.3.4 KAS2 Security Properties

11.2.1.7 Section 9 IFC based Key Transport Schemes

This section and its subsection are concerned with transferring keying material between sender and receiver entities using a RSA key pair, which is outside the scope of the TOE.

9.1 Additional Input
9.2 KTS-OAEP Family: Key Transport Using RSA-OAEP
9.2.1 KTS-OAEP Family Prerequisites
9.2.2 Common components
9.2.3 KTS-OAEP-basic
9.2.4 KTS-OAEP Key Confirmation
9.2.4.1 KTS-OAEP Common Components for Key Confirmation
9.2.4.2 KTS-OAEP-receiver-confirmation
9.2.5 KTS-OAEP Security Properties
9.3 KTS-KEM-KWS Family: Key Transport using RSA-KEM-KWS
9.3.1 KTS-KEM-KWS Family Prerequisites
9.3.2 Common Components of the KTS-KEM-KWS Schemes
9.3.3 KTS-KEM-KWS-basic
9.3.4 KTS-KEM-KWS Key Confirmation
9.3.4.1 KTS-KEM-KWS Common Components for Key Confirmation
9.3.4.2 KTS-KEM-KWS-receiver-confirmation
9.3.5 KTS-KEM-KWS Security Properties

11.2.1.8 Section 10 Key Recovery

This section and its subsections do not specify any “shall”, “shall not”, “should” or “should not” statements.

11.2.1.9 Section 11 Implementation Validation

The TOE shall be proven to comply with the “shall” statements in this section as evidenced by NIST CMVP FIPS 140-2 validation certificates when they are published on the NIST CMVP Validated FIPS 140-1 and FIPS 140-2 Cryptographic Modules website:

http://csrc.nist.gov/groups/STM/cmvp/documents/140-1/140val-all.htm

“Shall not”, “should”, and “should not” Options Implemented by TOE

N/A

Rationale for Implementation of “shall not” or “should not”

N/A

Omission of Functionality Related to "shall" or “should”

N/A

11.2.1.10 Appendix A: Summary of Differences between this Recommendation and ANS X9.44 (Informative)

This section and its subsections do not specify any “shall”, “shall not”, “should” or “should not” statements.

11.2.1.11 Appendix B: Data Conversions (Normative)

“Shall not”, “should”, and “should not” Options Implemented by TOE

N/A

Rationale for Implementation of “shall not” or “should not”

N/A

Omission of Functionality Related to "shall" or “should”

N/A

11.2.1.12 Appendix C: Prime Factor Recovery (Normative)

This section and its subsections do not specify any “shall”, “shall not”, “should” or “should not” statements.

11.2.1.13 Appendix D: References (Informative)

This section and its subsections do not specify any “shall”, “shall not”, “should” or “should not” statements.

12 Appendix D: TOE Binary List

Please send mail to wincc@microsoft.com if you would like a list of the Windows binaries included in this evaluation.