Microsoft Windows

Common Criteria Evaluation

Microsoft Windows 10 (Creators Update)

Security Target

Document Information	
Version Number	0.06
Updated On	June 14, 2018

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.

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,
- Describes the organization of the security target

1.1 ST Reference

ST Title: Microsoft Windows 10 (Creators Update) Security Target

ST Version: version 0.06, June 14, 2018

1.2 TOE Reference

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

- Microsoft Windows 10 Home Edition (Creators Update) (32-bit version)
- Microsoft Windows 10 Pro Edition (Creators Update) (64-bit versions)
- Microsoft Windows 10 Enterprise Edition (Creators Update) (64-bit versions)
- Microsoft Windows 10 S Edition (Creators Update) (64-bit versions)

TOE Versions:

- Windows 10: build 10.0.15063 (also known as version 1703)

The following security updates must be applied for:

- Windows 10, all critical updates as of August 9, 2017

1.3 TOE Overview

The TOE includes the Windows 10 operating system, and those applications necessary to manage, support and configure the operating system. Windows 10 can be delivered preinstalled on a new computer or downloaded from the Microsoft website.

1.3.1 TOE Types

Windows 10 editions, collectively called “Windows”, 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 expands 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.

1.3.2 TOE Usage

Windows 10 is suited for business desktops, notebook, and convertible computers. It is the workstation product and while it can be used by itself, it is designed to serve as a client within Windows domains.

Windows provides an interactive User Interface (UI), as well as a network interface. The TOE includes a set of Windows 10 systems that can be connected via their network interfaces and organized into domains and forests. A domain is a logical collection of Windows systems that allows the administration and application of a common security policy and the use of a common accounts database. One or more domains combine to comprise a forest. Windows supports single-domain and multiple-domain (i.e., forest) configurations as well as federation between forests and external authentication services.

Each domain must include at least one designated server known as a Domain Controller (DC) to manage the domain. The TOE allows for multiple DCs that replicate TOE user and machine account as well as group policy management data among themselves to provide for higher availability.

Each Windows system, whether it is a DC server, non-DC server, or workstation, provides a subset of the TSFs. The TSF subset for Windows 10 can consist of the security functions from a single system, for a stand-alone system, or the collection of security functions from an entire network of systems, for a domain configuration.

1.3.3 TOE Security Services

This section summarizes the security services provided by the TOE:

- **Security Audit:** Windows has the ability to collect audit data, review audit logs, protect audit logs from overflow, and restrict access to audit logs. Audit information generated by the system includes the date and time of the event, the user identity that caused the event to be generated, and other event specific data. Authorized administrators can review audit logs and have the ability to search and sort audit records. Authorized Administrators can also configure the audit system to include or exclude potentially auditable events to be audited based on a wide range of characteristics. In the context of this evaluation, the protection profile requirements cover generating audit events, selecting which events should be audited, and providing secure storage for audit event entries.
- **Cryptographic Support:** Windows provides FIPS-140-2 CAVP 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,[1] 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

[1] This option is not included in the Windows Common Criteria evaluation.

certificates generated and used by Windows authenticate users and machines as well as protect both user and system data in transit.

- **User Data Protection**: In the context of this evaluation Windows protects user data and provides virtual private networking capabilities.
- **Identification and Authentication** Each Windows user must be identified and authenticated based on administrator-defined policy prior to performing any TSF-mediated functions. An interactive user invokes a trusted path in order to protect his I&A information. Windows maintains databases of accounts including their identities, authentication information, group associations, and privilege and logon rights associations. Windows account policy functions include the ability to define the minimum password length, the number of failed logon attempts, the duration of lockout, and password age.
- **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.
- **TOE Access:** Windows allows an authorized administrator to configure the system to display a logon banner before the logon dialog.
- **Trusted Path for Communications**: Windows uses HTTPS, DTLS, and TLS to provide a trusted path for communications.

- **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.

1.3.4 Non-TOE Hardware, Software, Firmware in the Evaluation

Non-TOE Hardware Identification: The following real and virtualized hardware platforms, corresponding firmware, and components are included in the evaluated configuration:

- Microsoft Surface Laptop
- Microsoft Surface Pro
- Microsoft Surface Book
- Microsoft Surface Pro 4
- Microsoft Surface Pro 3
- Microsoft Surface Studio
- Microsoft Surface Hub

- Microsoft Windows Server 2016 Hyper-V
- HP Pro x2 612 G2 Detachable PC with LTE
- Dell 5285
- Dell Inspiron 660s
- Panasonic Toughbook

1.4 TOE Description

The TOE includes the Windows 10 operating system, supporting hardware, and those applications necessary to manage, support and configure the operating system.

1.4.1 Evaluated Configurations

The TOE includes four product variants of Windows 10 (build 10.0.15063):

- Microsoft Windows 10 Home Edition (Creators Update) (32-bit version)
- Microsoft Windows 10 Pro Edition (Creators Update) (64-bit versions)
- Microsoft Windows 10 Enterprise Edition (Creators Update) (64-bit versions)
- Microsoft Windows 10-S Edition (Creators Update) (64-bit versions)

Within this security target, when specifically referring to a type of TSF (for example, a domain controller), the TSF type will be explicitly stated. Otherwise, the term TSF refers to the total of all TSFs within the TOE.

1.4.2 Security Environment and TOE Boundary

The TOE includes both physical and logical boundaries. Its operational environment is a networked environment.

1.4.2.1 Logical Boundaries

Conceptually the Windows TOE can be thought of as a collection of the following security services which the security target describes with increasing detail in the remainder of this document:

- Security Audit
- Cryptographic Support
- User Data Protection
- Identification and Authentication
- Security Management
- Protection of the TOE Security Functions
- Access to the TOE
- Trusted Path and Channels

These services are primarily provided by Windows components:

- 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.

- 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 **Windows Trusted Installer** which installs updates to the Windows operating system.
- The **Local Security Authority Subsystem** which identifies and authenticates users prior to log on and generates events for the security audit log.
- FIPS-Approved **cryptographic algorithms** to protect user and system data.
- The **Key Isolation Service** which protects secret and private keys.
- **Local and remote administrative interfaces** for security management.
- **Windows Explorer** for Windows 10 which can be used to manage the OS and check the integrity of Windows files and updates.

1.4.2.2 Physical Boundaries

Each instance of the general purpose OS TOE runs on a tablet, convertible, workstation or server computer. The TOE executes on processors from Intel (x86 and x64) or AMD (x86 and x64) along with peripherals for input/output (keyboard, mouse, display, and network).

The TOE was tested on the following physical and virtual computer platforms:

- Microsoft Surface Laptop
- Microsoft Surface Pro
- Microsoft Surface Book
- Microsoft Surface Pro 4
- Microsoft Surface Pro 3
- Microsoft Surface Studio
- Microsoft Surface Hub
- Windows Server 2016 Hyper-V
- HP Pro x2 612 G2 Detachable PC with LTE
- Dell 5285
- Dell Inspiron 660s
- Panasonic Toughbook

The TOE does not include any hardware or network infrastructure components between the computers that comprise the distributed TOE. The security target assumes that any network connections, equipment, peripherals and cables are appropriately protected in the TOE security environment.

The Windows 10 operating system must be pre-installed on a computer by an OEM, installed by the end-user, by an organization’s IT administrator, or updated from a previous Windows 10 version downloaded from Windows Update. Consumers can download Windows 10 fromhttps://www.microsoft.com/en-us/software-download/windows10. The obtained file is in .iso format. Enterprises typically obtain Windows using volume licensing programs and subscriptions such as these for Windows 10.

Windows is pre-installed on all Windows Surface computers.

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

- *Windows 10 (Creators Update) GP OS Operational Guidance* along with all the documents referenced therein.

The administrator and user must follow the instructions in the *Windows 10 (Creators Update) GP OS Operational Guidance* in order to configure and remain in the evaluated configuration.

1.5 Product Description

In addition to core **operating system** capabilities described in the previous section, Windows 10 can also be categorized as the following types of Information Assurance (IA) or IA-enabled IT products, these capabilities leverage functionality included in this General Purpose OS evaluation as well as capabilities which fall outside the scope of the GP OS PP:

- Windows is a **Network Management** and **Desktop Management** product to support security infrastructure. Group Policy, which is part of the Windows 10 TOE, provide the centralized network management in Windows networks and desktops.
- Windows is a **Single Sign-On** product (using password or certificate) for Windows networks to defend the computing environment. Windows 10 support single sign on to the TOE.
- Windows is a **Firewall** product with the capability to filter network traffic based upon source and destination addresses, ports, applications, user or machine identity, and protocols.

Related operating system capabilities are covered in other evaluations:

- Windows 10 is a mobile device operating system currently undergoing evaluation against the Mobile Device Fundamentals protection profile 2.0 (see https://www.niap-ccevs.org/CCEVS_Products/in_eval.cfm).
- Windows 10 is currently undergoing evaluation against the IPsec VPN Client protection profile 1.4 (see https://www.niap-ccevs.org/CCEVS_Products/in_eval.cfm).

1.6 Conventions, Terminology, Acronyms

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

1.6.1 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.

1.6.2 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[2] and the disclosure and modification of data[3].
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[4], 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.

[2] Hardware and software

[3] Stored or communicated

[4] According to a defined metric

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 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 2016 [Standard] and Windows Server 2016 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.

1.6.3 Acronyms

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

1.7 ST Overview and Organization

This security target contains the following additional sections:

- **CC Conformance Claims (Section 2)**: Formal conformance claims which are examined during the evaluation.
- **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 ***General Purpose Operating Systems Protection Profile***.
- **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 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 5, April 2017, extended (Part 2 extended)
- Common Criteria for Information Technology Security Evaluation Part 3: Security assurance requirements Version 3.1, Revision 5 April 2017, (Part 3 extended)
- General Purpose Operating Systems Protection Profile, Version 4.1, March 9, 2016 (GP OS PP)

The security functional requirements and assurance activities have been modified with the following NIAP Technical Decisions:

- Technical Decision 104 for FMT_MOF_SMF.1
- Technical Decision 107 for FCS_CKM.1
- Technical Decision 163 for FCS_TLSC_EXT.1
- Technical Decision 208 for FTP_TRP.1
- Technical Decision 239 for FCS_CKM.4
- Technical Decision 244 for FCS_TLSC_EXT.2
- Technical Decision 246 for FIA_UAU.5

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 5, April 2017.

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 10. The assumptions, threats, and policies are copied from the General Purpose Operating Systems Protection Profile, Version 4.1, March 9, 2016 ("GP OS PP").

3.1 Threats to Security

Table 1 presents known or presumed threats to protected resources that are addressed by Windows based on conformance to the General Purpose Operating Systems Protection Profile.

Table 1 GP OS PP Threats Addressed by Windows

Threat	Description
T.NETWORK_ATTACK	An attacker is positioned on a communications channel or elsewhere on the network infrastructure. Attackers may engage in communications with applications and services running on or part of the OS with the intent of compromise. Engagement may consist of altering existing legitimate communications.
T.NETWORK_EAVESDROP	An attacker is positioned on a communications channel or elsewhere on the network infrastructure. Attackers may monitor and gain access to data exchanged between applications and services that are running on or part of the OS.
T.LOCAL_ATTACK	An attacker may compromise applications running on the OS. The compromised application may provide maliciously formatted input to the OS through a variety of channels including unprivileged system calls and messaging via the file system.
T.LIMITED_PHYSICAL_ACCESS	An attacker may attempt to access data on the OS while having a limited amount of time with the physical device.

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 2** describes organizational security policies which are necessary for conformance to the protection profile.

Table 2 Organizational Security Policies

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

3.3 Secure Usage Assumptions

Table 3 describes the core security aspects of the environment in which Windows 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 protection profile:

Table 3 Secure Usage Assumptions

Assumption	Description
A.PLATFORM	The OS relies upon a trustworthy computing platform for its execution. This underlying platform is out of scope of this PP.
A.PROPER_USER	The user of the OS is not willfully negligent or hostile, and uses the software in compliance with the applied enterprise security policy. At the same time, malicious software could act as the user, so requirements which confine malicious subjects are still in scope.
A.PROPER_ADMIN	The administrator of the OS is not careless, willfully negligent or hostile, and administers the OS within compliance of the applied enterprise security policy.

4 Security Objectives

This section defines the security objectives of Windows 10 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 describes the security objectives for Windows which are needed to comply with the GP OS PP.

Table 4 Security Objectives for the TOE

Security Objective	Source
O.ACCOUNTABILITY	Conformant OSs ensure that information exists that allows administrators to discover unintentional issues with the configuration and operation of the operating system and discover its cause. Gathering event information and immediately transmitting it to another system can also enable incident response in the event of system compromise.
O.INTEGRITY	Conformant OSs ensure the integrity of their update packages. OSs are seldom if ever shipped without errors, and the ability to deploy patches and updates with integrity is critical to enterprise network security. Conformant OSs provide execution environment based mitigations that increase the cost to attackers by adding complexity to the task of compromising systems.
O.MANAGEMENT	To facilitate management by users and the enterprise, conformant OSes provide consistent and supported interfaces for their security relevant configuration and maintenance. This includes the deployment of applications and application updates through the use of platform supported deployment mechanisms and formats, as well as providing mechanisms for configuration and application execution control.
O.PROTECTED_STORAGE	To address the issue of loss of confidentiality of credentials in the event of loss of physical control of the storage medium, conformant OSs provide data-at-rest protection for credentials. Conformant OSes also provide access controls which allow users to keep their files private from other users of the same system.
O.PROTECTED_COMMS	To address both passive (eavesdropping) and active (packet modification) network attack threats, conformant OSs provide mechanisms to create trusted channels for CSP and sensitive data. Both CSP and sensitive data should not be exposed outside of the platform.

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 5** describes the security objectives for the operational environment as specified in the protection profile.

Table 5 Security Objectives for the Operational Environment

Environment Objective	Description
OE.PLATFORM	The OS relies on being installed on trusted hardware.
OE.PROPER_USER	The user of the OS is not willfully negligent or hostile, and uses the software within compliance of the applied enterprise security policy. Standard user accounts are provisioned in accordance with the least privilege model. Users requiring higher levels of access should have a separate account dedicated for that use.
OE.PROPER_ADMIN	The administrator of the OS is not careless, willfully negligent or hostile, and administers the OS within compliance of the applied enterprise policy.

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 General Purpose Operating Systems Protection Profile, Version 4.1, March 9, 2016 (GP OS PP), the Common Criteria, or are defined in the following section.

Conventions:

Where requirements are drawn from the protection profile, 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 6 TOE Security Functional Requirements

Requirement Class	Requirement Component
Security Audit (FAU)	Audit Data Generation (FAU_GEN.1)
Cryptographic Support (FCS)	Cryptographic Key Generation for (FCS_CKM.1(1))
	Cryptographic Key Establishment (FCS_CKM.2(1)
	Cryptographic Key Destruction (FCS_CKM.4)
	Cryptographic Operation for Data Encryption/Decryption (FCS_COP.1(SYM))
	Cryptographic Operation for Hashing (FCS_COP.1(HASH))
	Cryptographic Operation for Signing (FCS_COP.1(SIGN))

	Cryptographic Operation for Keyed Hash Algorithms (FCS_COP.1(HMAC))
	Random Bit Generation (FCS_RBG_EXT.1)
	Storage of Sensitive Data (FCS_STO_EXT.1)
	TLS Client Protocol (FCS_TLSC_EXT.1)
	TLS Client Protocol (FCS_TLSC_EXT.2)
	TLS Client Protocol (FCS_TLSC_EXT.3)
	TLS Client Protocol (FCS_TLSC_EXT.4)
	DTLS Implementation (FCS_DTLS_EXT.1)
User Data Protection (FDP)	Access Controls for Protecting User Data (FDP_ACF_EXT.1)
	Information Flow Control (FDP_IFC_EXT.1)
Identification & Authentication (FIA)	Authorization Failure Handling (FIA_AFL.1)
	Multiple Authentication Mechanisms (FIA_UAU.5)
	X.509 Certification Validation (FIA_X509_EXT.1)
	X.509 Certificate Authentication (FIA_X509_EXT.2)
Security Management (FMT)	Management of Security Functions Behavior (FMT_MOF_EXT.1)
	Specification of Management Functions (FMT_SMF_EXT.1)
Protection of the TSF (FPT)	Access Controls (FPT_ACF_EXT.1)
	Address Space Layout Randomization (FPT_ASLR_EXT.1)
	Stack Buffer Overflow Protection (FPT_SBOP_EXT.1)
	Software Restriction Policies (FPT_SRP_EXT.1)
	Boot Integrity (FPT_TST_EXT.1)
	Trusted Update (FPT_TUD_EXT.1)
	Trusted Update for Application Software (FPT_TUD_EXT.2)
TOE Access (FTA)	Default TOE Access Banners (FTA_TAB.1)
Trusted Path/Channels (FTP)	Trusted Path (FTP_TRP.1)
	Trusted Channel Communication (FTP_ITC_EXT.1(TLS))
	Trusted Channel Communication (FTP_ITC_EXT.1(DTLS))

5.1.1 Security Audit (FAU)

5.1.1.1 Audit Data Generation (FAU_GEN.1)

FAU_GEN.1.1 The OS shall be able to generate an audit record of the following auditable events:

a. Start-up and shutdown of the audit functions;

b. All auditable events for the not specified level of audit; and

c.

- Authentication events (Success/Failure);
- Use of privileged/special rights events (Successful and unsuccessful security, audit, and configuration changes);
- Privilege or role escalation events (Success/Failure);

[

- ***File and object events (Successful and unsuccessful attempts to create, access, delete, modify, modify permissions),***
- ***User and Group management events (Successful and unsuccessful add, delete, modify, disable),***
- ***Audit and log data access events (Success/Failure),***

- ***Cryptographic verification of software (Success/Failure),***
- ***Program initiations (Success/Failure e.g. due to software restriction policy),***[5]
- ***System reboot, restart, and shutdown events (Success/Failure),***
- ***Kernel module loading and unloading events (Success/Failure),***
- ***Administrator or root-level access events (Success/Failure),***
- ***[Lock and Unlock a user account].***

]

FAU_GEN.1.2 The OS shall record within each audit record at least the following information:

a. Date and time of the event, type of event, subject identity (if applicable), and outcome (success or failure) of the event; and
b. For each audit event type, based on the auditable event definitions of the functional components included in the PP/ST [**none**].

5.1.2 Cryptographic Support (FCS)

5.1.2.1 Cryptographic Key Generation (FCS_CKM.1(1))

Application Note: FCS_CKM.1.1 corresponds to FCS_CKM.1.1(1) in the GP OS protection profile.

FCS_CKM.1.1 The OS shall generate asymmetric cryptographic keys in accordance with a specified cryptographic key generation algorithm [

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

].

5.1.2.2 Cryptographic Key Establishment (FCS_CKM.2(1))

Application Note: FCS_CKM.2.1 corresponds to FCS_CKM.2.1(1) in the GP OS protection profile.

FCS_CKM.2.1 The OS shall implement functionality to perform cryptographic key establishment in accordance with a specified cryptographic key establishment method:

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

[5] For Windows 10 and Windows 10 S only.

Pair-Wise Key Establishment Schemes Using Discrete Logarithm Cryptography"

].

5.1.2.3 Cryptographic Key Destruction (FCS_CKM.4)[6]

FCS_CKM.4.1 The TSF shall destroy cryptographic keys in accordance with a specified cryptographic key destruction method [

- ***For volatile memory, the destruction shall be executed by a [single overwrite consisting of [zeroes]];***

].

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

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

FCS_COP.1.1(SYM) The OS shall perform encryption/decryption services for data in accordance with a specified cryptographic algorithm:[

- ***AES-XTS (as defined in NIST SP 800-38E),***
- ***AES-CBC (as defined in NIST SP 800-38A)***]

and [

- ***AES Key Wrap (KW) (as defined in NIST SP 800-38F),***
- ***AES-GCM (as defined in NIST SP 800-38D),***
- ***AES-CCM (as defined in NIST SP 800-38C)***

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

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

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

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

5.1.2.6 Cryptographic Operation for Signing (FCS_COP.1(SIGN))

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

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

[

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

[6] This protection profile requirement was replaced as part of NIAP Technical Decision 239.

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

].

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

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

FCS_COP.1.1(HMAC) The OS shall perform keyed-hash message authentication services in accordance with a specified cryptographic algorithm [

- ***SHA-1,***
- ***SHA-256,***
- ***SHA-384,***
- ***SHA-512***

] with key sizes [**128 and 256 bits**] and message digest sizes [***160 bits, 256 bits, 384 bits, 512 bits***] that meet the following: FIPS Pub 198-1 *The Keyed-Hash Message Authentication Code* and FIPS Pub 180-4 *Secure Hash Standard*.

5.1.2.8 Random Bit Generation (FCS_RBG_EXT.1)

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

FCS_RBG_EXT.1.2 The deterministic RBG used by the OS shall be seeded by an entropy source that accumulates entropy from a ***[software-based noise source, platform-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.

5.1.2.9 Storage of Sensitive Data (FCS_STO_EXT.1)

FCS_STO_EXT.1.1 The OS shall implement functionality to encrypt sensitive data stored in non-volatile storage and provide interfaces to applications to invoke this functionality.

5.1.2.10 TLS Client Protocol (FCS_TLSC_EXT.1)

FCS_TLSC_EXT.1.1 The OS shall implement TLS 1.2 (RFC 5246) supporting the following ciphersuites:

Mandatory ciphersuites:

- TLS_RSA_WITH_AES_128_CBC_SHA as defined in RFC 5246

Optional ciphersuites: [

- ***TLS_RSA_WITH_AES_256_CBC_SHA as defined in RFC 5246***
- ***TLS_ECDHE_ECDSA_WITH_AES_128_CBC_SHA as defined in RFC 4492***
- ***TLS_ECDHE_ECDSA_WITH_AES_256_CBC_SHA as defined in RFC 4492***
- ***TLS_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_CBC_SHA256 as defined in RFC 5289***
- ***TLS_ECDHE_ECDSA_WITH_AES_256_CBC_SHA384 as defined in RFC 5289***
- ***TLS_ECDHE_ECDSA_WITH_AES_128_GCM_SHA256 as defined in RFC 5289***
- ***TLS_ECDHE_ECDSA_WITH_AES_256_GCM_SHA384 as defined in RFC 5289***

].

FCS_TLSC_EXT.1.2 The OS shall verify that the presented identifier matches the reference identifier according to RFC 6125.

FCS_TLSC_EXT.1.3 The OS shall only establish a trusted channel if the peer certificate is valid.

5.1.2.11 TLS Client Protocol (FCS_TLSC_EXT.2)[7]

FCS_TLSC_EXT.2.1 The OS shall present the Supported Elliptic Curves Extension in the Client Hello with the following NIST curves: [***secp256r1, secp384r1, secp521r1***] ~~and no other curves~~.

5.1.2.12 TLS Client Protocol (FCS_TLSC_EXT.3)

FCS_TLSC_EXT.3.1 The OS shall present the signature_algorithms extension in the Client Hello with the supported_signature_algorithms value containing the following hash algorithms [***SHA256, SHA384, SHA512***] and no other hash algorithms.

5.1.2.13 TLS Client Protocol (FCS_TLSC_EXT.4)

FCS_TLSC_EXT.4.1 The OS shall support mutual authentication using X.509v3 certificates.

5.1.2.14 DTLS Implementation (FCS_DTLS_EXT.1)

FCS_DTLS_EXT.1.1 The OS shall implement the DTLS protocol in accordance with [***DTLS 1.0 (RFC 4347), DTLS 1.2 (RFC 6347)***].

FCS_DTLS_EXT.1.2 The OS shall implement the requirements in TLS (FCS_TLSC_EXT.1) for the DTLS implementation, except where variations are allowed according to DTLS 1.2 (RFC 6347).

5.1.3 User Data Protection (FDP)

5.1.3.1 Access Controls for Protecting User Data (FDP_ACF_EXT.1)

FDP_ACF_EXT.1.1 The OS shall implement access controls which can prohibit unprivileged users from accessing files and directories owned by other users.

5.1.3.2 Information Flow Control (FDP_IFC_EXT.1)

FDP_IFC_EXT.1.1 The OS shall [

[7] This protection profile requirement was replaced as part of NIAP Technical Decision 244.

- ***Provide an interface which allows a VPN client to protect all IP traffic using IPsec***

] with the exception of IP traffic required to establish the VPN connection.

5.1.4 Identification and Authentication (FIA)

5.1.4.1 Authentication Failure Handling (FIA_AFL.1)

FIA_AFL.1.1 The OS shall detect when [***an administrator configurable positive integer within a [range of 1 - 999]***

] unsuccessful authentication attempts for [

- ***authentication based on user name and password,***

] occur related to [**the last successful authentication by that user**].

FIA_AFL.1.2 When the defined number of unsuccessful authentication attempts for an account has been met, the OS shall: [***Account Lockout***].

5.1.4.2 Multiple Authentication Mechanisms (FIA_UAU.5)

FIA_UAU.5.1 The OS shall provide the following authentication mechanisms:

[

- ***Authentication based on user name and password,***
- ***authentication based on user name and a PIN that releases an asymmetric key stored in OE-protected storage***[8]

] to support user authentication.

FIA_UAU.5.2 The OS shall authenticate any user's claimed identity according to the [**authentication based on username and password is performed for TOE-originated requests and with credentials stored by the OS for Windows Hello, smart card and virtual smart card**].

5.1.4.3 X.509 Certification Validation (FIA_X509_EXT.1)

FIA_X509_EXT.1.1 The OS shall implement functionality to validate certificates in accordance with the following rules:

- RFC 5280 certificate validation and certificate path validation.
- The certificate path must terminate with a trusted CA certificate.
- The OS 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 OS 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, an OCSP TLS Status Request Extension (i.e., OCSP stapling) as specified in RFC 6066***].
- The OS shall validate the extendedKeyUsage field according to the following rules:

[8] PIN-based authentication for Windows 10 and Windows 10 S only, smart card authentication for Windows 10 and Surface Hub only.

- ○ Certificates used for trusted updates and executable code integrity verification shall have the Code Signing purpose (id-kp 3 with OID 1.3.6.1.5.5.7.3.3) in the extendedKeyUsage field.
- ○ Server certificates presented for TLS shall have the Server Authentication purpose (id-kp 1 with OID 1.3.6.1.5.5.7.3.1) in the extendedKeyUsage field.
- ○ Client certificates presented for TLS shall have the Client Authentication purpose (id-kp 2 with OID 1.3.6.1.5.5.7.3.2) in the extendedKeyUsage field.
- ○ S/MIME certificates presented for email encryption and signature shall have the Email Protection purpose (id-kp 4 with OID 1.3.6.1.5.5.7.3.4) in the extendedKeyUsage field.
- ○ OCSP certificates presented for OCSP responses shall have the OCSP Signing purpose (id-kp 9 with OID 1.3.6.1.5.5.7.3.9) in the extendedKeyUsage field.

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

5.1.4.4 X.509 Certificate Authentication (FIA_X509_EXT.2)

FIA_X509_EXT.2.1 The OS shall use X.509v3 certificates as defined by RFC 5280 to support authentication for TLS and [***HTTPS***] connections.

5.1.5 Security Management (FMT)

5.1.5.1 Management of Security Functions Behavior (FMT_MOF_EXT.1)

FMT_MOF_EXT.1.1 The TSF shall restrict the ability to perform the function indicated in column 3 of the “Management Functions” table in FMT_SMF_EXT.1.1 to the administrator.

5.1.5.2 Management of Security Functions Behavior (FMT_SMF_EXT.1)

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

Management Function	FMT_SMF_EXT.1	FMT_MOF_EXT.1
Enable/disable screen lock	M	O
Configure screen lock inactivity timeout	M	O
Configure local audit storage capacity	M	O
Configure minimum password Length	O	O
~~Configure minimum number of special characters in password~~	~~O~~	~~O~~
~~Configure minimum number of numeric characters in password~~	~~O~~	~~O~~
~~Configure minimum number of uppercase characters in password~~	~~O~~	~~O~~
~~Configure minimum number of lowercase characters in password~~	~~O~~	~~O~~

Configure remote connection inactivity timeout	O	O
~~Enable/disable unauthenticated logon~~	~~O~~	~~M~~
Configure lockout policy for unsuccessful authentication attempts through [***timeouts between attempts, limiting number of attempts during a time period***]	O	O
Configure host-based firewall	O	O
Configure name/address of directory server to bind with[9]	O	O
Configure name/address of remote management server from which to receive management settings	O	O
~~Configure name/address of audit/logging server to which to send audit/logging records~~	~~O~~	~~O~~
Configure audit rules	O	O
Configure name/address of network time server	O	O
Enable/disable automatic software update	O	O
Configure Wi-Fi interface	O	O
Enable/disable Bluetooth interface	O	O
Configure USB interfaces	O	O
Enable/disable [**local area network interface**]	O	O
[**none**]	O	O

Application Note: The intent of this requirement is to ensure that the security target is populated with the management functions which are provided by Windows. This enables developers of compliance checklists, including those provided as operational user guidance, to leverage this table by providing enterprise specific values for each evaluated item.

5.1.6 Protection of the TSF (FPT)

5.1.6.1 Access Controls (FPT_ACF_EXT.1)

FPT_ACF_EXT.1.1 The OS shall implement access controls which prohibit unprivileged users from modifying:

- Kernel and its drivers/modules

[9] For Windows 10 Pro, Windows 10 Enterprise, and Surface Hub.

- Security audit logs
- Shared libraries
- System executables
- System configuration files
- [**none**]

FPT_ACF_EXT.1.2 The OS shall implement access controls which prohibit unprivileged users from reading:

- Security audit logs
- System-wide credential repositories
- [**none**]

5.1.6.2 Address Space Layout Randomization (FPT_ASLR_EXT.1)

FPT_ASLR_EXT.1.1 The OS shall always randomize process address space memory locations except for [**none**].

5.1.6.3 Stack Buffer Overflow Protection (FPT_SBOP_EXT.1)

FPT_SBOP_EXT.1.1 The OS shall be compiled with stack-based buffer overflow protections enabled.

5.1.6.4 Software Restriction Policies (FPT_SRP_EXT.1)[10]

FPT_SRP_EXT.1.1 The OS shall restrict execution to only programs which match an administrator-specified [

- ***File path,***
- ***File digital signature,***
- ***Version,***[11]
- ***Hash***

].

5.1.6.5 Boot Integrity (FPT_TST_EXT.1)

FPT_TST_EXT.1.1 The OS shall verify the integrity of the bootchain up through the OS kernel and [***operating system executable code and application executable code***] prior to its execution through the use of [***a digital signature using a hardware-protected asymmetric key, a hardware-protected hash***].[12]

5.1.6.6 Trusted Update (FPT_TUD_EXT.1)

FPT_TUD_EXT.1.1 The OS shall provide the ability to check for updates to the OS software itself.

[10] For Windows 10 Home, Windows 10 Pro, Windows 10 S editions only.

[11] Windows 10 Enterprise can restrict program execution based on a version using AppLocker and Device Guard; Windows 10 Pro and Windows 10 Home editions cannot.

[12] Windows can also run on computers that do not have a TPM, which is the mechanism that provides the hardware-based protection for boot integrity.

FPT_TUD_EXT.1.2 The OS shall cryptographically verify updates to itself using a digital signature prior to installation using schemes specified in FCS_COP.1(**SIGN**).

5.1.6.7 Trusted Update for Application Software (FPT_TUD_EXT.2)

FPT_TUD_EXT.2.1 The OS shall provide the ability to check for updates to application software.

FPT_TUD_EXT.2.2 The OS shall cryptographically verify the integrity of updates to applications using a digital signature specified by FCS_COP.1(**SIGN**) prior to installation.

5.1.7 TOE Access (FTA)

5.1.7.1 Default TOE Access Banners (FTA_TAB.1)

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

5.1.8 Trusted Path / Channels (FTP)

5.1.8.1 Trusted Path (FTP_TRP.1)

FTP_TRP.1.1 The OS shall provide a communications path between itself and [***local, remote***] users that is logically distinct from other communications paths and provides assured identification of its endpoints and protection of the communicated data from modification and disclosure.[13]

FTP_TRP.1.2 The OS shall permit [***the TSF, local users, remote users***] to initiate communication via the trusted path.

FTP_TRP.1.3 The OS shall require use of the trusted path for all remote administrative actions.

5.1.8.2 Trusted Channel Communication (FTP_ITC_EXT.1(TLS))

FTP_ITC_EXT.1.1(TLS) The OS shall use [

- ***TLS as conforming to FCS_TLSC_EXT.1***

] to provide a trusted communications channel between itself and authorized IT entities supporting the following capabilities: [***authentication server, [CRL checking, web traffic]***] 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.

5.1.8.3 Trusted Channel Communication (FTP_ITC_EXT.1(DTLS))

FTP_ITC_EXT.1.1(DTLS) The OS shall use [

- ***DTLS as conforming to FCS_DTLS_EXT.1,***

] to provide a trusted communications channel between itself and authorized IT entities supporting the following capabilities: [***web traffic, datagram-based application protocols***] that is logically distinct from other

[13] This protection profile requirement was modified as part of NIAP Technical Decision 208.

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.

5.2 TOE Security Assurance Requirements

5.2.1 CC Part 3 Assurance Requirements

The following table is the collection of CC Part 3 assurance requirements from the General Purpose Operating Systems Protection Profile.

Requirement Class	Requirement Component
Security Target (ASE)	ST Introduction (ASE_INT.1)
	Conformance Claims (ASE_CCL.1)
	Security Objectives (ASE_OBJ.1)
	Extended Components Definition (ASE_ECD.1)
	Stated Security Requirements (ASE_REQ.1)
	Security Problem Definition (ASE_SPD.1)
	TOE Summary Specification (ASE_TSS.1)
Design (ADV)	Basic Functional Specification (ADV_FSP.1)
Guidance (AGD)	Operational User Guidance (AGD_OPE.1)
	Preparative Procedures (AGD_PRE.1)
Lifecycle (ALC)	Labeling of the TOE (ALC_CMC.1)
	TOE CM Coverage (ALC_CMS.1)
	Timely Security Updates (ALC_TSU_EXT.1)
Testing (ATE)	Independent Testing – Conformance (ATE_IND.1)
Vulnerability Assessment (AVA	Vulnerability Survey (AVA_VAN.1)

Table 7 TOE Security Assurance Requirements

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**.**

ALC_TSU_EXT.1.2D The developer shall provide a description in the TSS of how users are notified when updates change security properties or the configuration of the product**.**

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.

ALC_TSU_EXT.1.2C The description shall include the mechanisms publicly available for reporting security issues pertaining to the TOE. 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).

Evaluator action elements:

ALC_TSU_EXT.1.1E The evaluator shall confirm that the information provided meets all the requirements for content and presentation of evidence.

Assurance Activity: The evaluator will verify that the TSS contains a description of the timely security update process used by the developer to create and deploy security updates. The evaluator will verify that this description addresses the entire application. The evaluator will also verify that, in addition to the OS developer’s process, any third-party processes are also addressed in the description. The evaluator will also verify that each mechanism for deployment of security updates is described.

The evaluator will verify that, for each deployment mechanism described for the update process, the TSS lists a time between public disclosure of a vulnerability and public availability of the security update to the OS patching this vulnerability, to include any third-party or carrier delays in deployment. The evaluator will verify that this time is expressed in a number or range of days.

The evaluator will verify that this description includes the publicly available mechanisms (including either an email address or website) for reporting security issues related to the OS. The evaluator shall verify that the description of this mechanism includes a method for protecting the report either using a public key for encrypting email or a trusted channel for a website.

5.2.2 General Purpose OS 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 Security Audit (FAU)

FAU_GEN.1.1

The evaluator shall check the administrative guide and ensure that it lists all of the auditable events. The evaluator shall check to make sure that every audit event type selected in the ST is included.

The evaluator shall test the OS's ability to correctly generate audit records by having the TOE generate audit records for the events listed in the ST. This should include all instance types of an event specified. When verifying the test results, the evaluator shall ensure the audit records generated during testing match the format specified in the administrative guide, and that the fields in each audit record have the proper entries.

FAU_GEN.1.2

The evaluator shall check the administrative guide and ensure that it provides a format for audit records. Each audit record format type must be covered, along with a brief description of each field. The evaluator shall ensure that the fields contains the information required.

The evaluator shall test the OS's ability to correctly generate audit records by having the TOE generate audit records for the events listed in the ST. The evaluator shall ensure the audit records generated during testing match the format specified in the administrative guide, and that the fields in each audit record provide the required information.

5.2.2.2 Cryptographic Support (FCS)

5.2.2.2.1 Cryptographic Key Generation (FCS_CKM.1(1))

The evaluator will ensure that the TSS identifies the key sizes supported by the OS. If the ST specifies more than one scheme, the evaluator will examine the TSS to verify that it identifies the usage for each scheme.

The evaluator will verify that the AGD guidance instructs the administrator how to configure the OS to use the selected key generation scheme(s) and key size(s) for all uses defined in this PP.

Assurance Activity Note: The following tests may require the vendor to furnish a developer environment and developer tools that are typically not available to end-users of the OS.

Key Generation for FIPS PUB 186-4 RSA Schemes

The evaluator will verify the implementation of RSA Key Generation by the OS 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 will verify the correctness of the TSF's implementation by comparing values generated by the TSF with those generated from a known good implementation.

If possible, the Random Probable primes method should also be verified against a known good implementation as described above. Otherwise, the evaluator will have the TSF generate 10 keys pairs for each supported key length nlen and verify:

- $n = p \cdot q$,
- p and q are probably prime according to Miller-Rabin tests,
- GCD(p-1,e) = 1,
- GCD(q-1,e) = 1,
- $2^{16} \le e \le 2^{256}$ and e is an odd integer,
- $p\text{-}q| > 2^{nlen/2 - 100}$,
- $p \ge 2^{nlen/2 - 1/2}$,

- $q \geq 2^{nlen/2 - 1/2}$,
- $2^{(nlen/2)} < d < LCM(p-1,q-1)$,
- $e \cdot d = 1 \bmod LCM(p-1,q-1)$.

Key Generation for ANSI X9.31-1998 RSA Schemes

If the TSF implements the ANSI X9.31-1998 scheme, the evaluator will 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 will ensure that the TSS contains the following information:

- The TSS shall list all sections of the standard to which the OS 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 OS 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 OS;
- For each applicable section of Appendix B, any omission of functionality related to "shall" or “should” statements shall be described.

Key Generation for Elliptic Curve Cryptography (ECC)

FIPS 186-4 ECC Key Generation Test

For each supported NIST curve, i.e., P-256, P-384 and P-521, the evaluator will 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 will 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-384 and P-521, the evaluator will 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 will obtain in response a set of 10 PASS/FAIL values.

5.2.2.2.2 Cryptographic Key Establishment (FCS_CKM.2(1))

The evaluator will ensure that the supported key establishment schemes correspond to the key generation schemes identified in FCS_CKM.1.1. If the ST specifies more than one scheme, the evaluator will examine the TSS to verify that it identifies the usage for each scheme.

The evaluator will verify that the AGD guidance instructs the administrator how to configure the OS to use the selected key establishment scheme(s).

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 factory products.

Key Establishment Schemes

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

SP800-56A Key Establishment Schemes

The evaluator will verify the OS'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 the OS has implemented the components of the key agreement scheme according to the specifications in the Recommendation. These components include the calculation of the discrete logarithm cryptography (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 will 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 MAC data and the calculation of MAC tag.

Function Test

The Function test verifies the ability of the OS 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 OS's 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 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 will obtain the DKM, the corresponding OS'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 OS id fields.

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

The evaluator will 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 OS shall perform the above for each implemented approved MAC algorithm.

Validity Test

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

> The evaluator will inject an error in some of the test vectors to test that the OS 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 MAC'd, or the generated MAC tag. If the OS 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 OS’s static private key to assure the OS 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 OS shall use these modified test vectors to emulate the key agreement scheme using the corresponding parameters. The evaluator will compare the OS’s results with the results using a known good implementation verifying that the OS detects these errors.

SP800-56B Key Establishment Schemes

The evaluator will verify that the TSS describes whether the OS acts as a sender, a recipient, or both for RSA-based key establishment schemes.

If the OS acts as a sender, the following assurance activity shall be performed to ensure the proper operation of every OS supported combination of RSA-based key establishment scheme:

> To conduct this test the evaluator will generate or obtain test vectors from a known good implementation of the OS's supported schemes. For each combination of supported key establishment scheme and its options (with or without key confirmation if supported, for each supported key confirmation MAC function if key confirmation is supported, and for each supported mask generation function if KTS-OAEP is supported), the tester shall generate 10 sets of test vectors. Each test vector shall include the RSA public key, the plaintext keying material, any additional input parameters if applicable, the MAC key and MAC tag if key confirmation is incorporated, and the outputted ciphertext. For each test vector, the evaluator shall perform a key establishment encryption operation on the OS with the same inputs (in cases where key confirmation is incorporated, the test shall use the MAC key from the test vector instead of the randomly generated MAC key used in normal operation) and ensure that the outputted ciphertext is equivalent to the ciphertext in the test vector.

If the OS acts as a receiver, the following assurance activities shall be performed to ensure the proper operation of every OS supported combination of RSA-based key establishment scheme:

> To conduct this test the evaluator will generate or obtain test vectors from a known good implementation of the OS's supported schemes. For each combination of supported key establishment scheme and its options (with our without key confirmation if supported, for each supported key confirmation MAC function if key confirmation is supported, and for each supported mask generation function if KTS-OAEP is supported), the tester shall generate 10 sets of test vectors. Each test vector shall include the RSA private key, the plaintext keying material, any additional input parameters if applicable, the MAC tag in cases where key confirmation is incorporated, and the outputted ciphertext. For each test vector, the evaluator will perform the key establishment decryption operation on the OS and ensure that the outputted plaintext

keying material is equivalent to the plaintext keying material in the test vector. In cases where key confirmation is incorporated, the evaluator will perform the key confirmation steps and ensure that the outputted MAC tag is equivalent to the MAC tag in the test vector.

The evaluator will ensure that the TSS describes how the OS handles decryption errors. In accordance with NIST Special Publication 800-56B, the OS must not reveal the particular error that occurred, either through the contents of any outputted or logged error message or through timing variations. If KTS-OAEP is supported, the evaluator will create separate contrived ciphertext values that trigger each of the three decryption error checks described in NIST Special Publication 800-56B section 7.2.2.3, ensure that each decryption attempt results in an error, and ensure that any outputted or logged error message is identical for each. If KTS-KEM-KWS is supported, the evaluator will create separate contrived ciphertext values that trigger each of the three decryption error checks described in NIST Special Publication 800-56B section 7.2.3.3, ensure that each decryption attempt results in an error, and ensure that any outputted or logged error message is identical for each.

5.2.2.2.3 Cryptographic Key Destruction (FCS_CKM.4)[14]

The evaluator examines the TSS to ensure it describes how the keys are managed in volatile memory. This description includes details of how each identified key is introduced into volatile memory (e.g. by derivation from user input, or by unwrapping a wrapped key stored in non-volatile memory) and how they are overwritten.

The evaluator shall check to ensure the TSS lists each type of key that is stored in in non-volatile memory, and identifies how the TOE interacts with the underlying platform to manage keys (e.g., store, retrieve, destroy). The description includes details on the method of how the TOE interacts with the platform, including an identification and description of the interfaces it uses to manage keys (e.g., file system APIs, platform key store APIs).

The evaluator examines the interface description for each different media type to ensure that the interface supports the selection(s) and description in the TSS.

If the ST makes use of the open assignment and fills in the type of pattern that is used, the evaluator examines the TSS to ensure it describes how that pattern is obtained and used. The evaluator shall verify that the pattern does not contain any CSPs.

The evaluator shall check that the TSS identifies any configurations or circumstances that may not strictly conform to the key destruction requirement.

There are a variety of concerns that may prevent or delay key destruction in some cases. The evaluator shall check that the guidance documentation identifies configurations or circumstances that may not strictly conform to the key destruction requirement, and that this description is consistent with the relevant parts of the TSS and any other relevant Required Supplementary Information. The evaluator shall check that the guidance documentation provides guidance on situations where key destruction may be delayed at the physical layer and how such situations can be avoided or mitigated if possible.

[14] This protection profile assurance activity was replaced as part of NIAP Technical Decision 239.

Some examples of what is expected to be in the documentation are provided here.

When the TOE does not have full access to the physical memory, it is possible that the storage may be implementing wear-leveling and garbage collection. This may create additional copies of the key that are logically inaccessible but persist physically. In this case, to mitigate this the drive should support the TRIM command and implements garbage collection to destroy these persistent copies when not actively engaged in other tasks.

Drive vendors implement garbage collection in a variety of different ways, as such there is a variable amount of time until data is truly removed from these solutions. There is a risk that data may persist for a longer amount of time if it is contained in a block with other data not ready for erasure. To reduce this risk, the operating system and file system of the OE should support TRIM, instructing the non-volatile memory to erase copies via garbage collection upon their deletion. If a RAID array is being used, only set-ups that support TRIM are utilized. If the drive is connected via PCI-Express, the operating system supports TRIM over that channel.

The drive should be healthy and contains minimal corrupted data and should be end of lifed before a significant amount of damage to drive health occurs, this minimizes the risk that small amounts of potentially recoverable data may remain in damaged areas of the drive.

- **Test 1**: Applied to each key held as in volatile memory and subject to destruction by overwrite by the TOE (whether or not the value is subsequently encrypted for storage in volatile or non-volatile memory). In the case where the only selection made for the destruction method key was removal of power, then this test is unnecessary. The evaluator shall:

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

  Steps 1-6 ensure that the complete key does not exist anywhere in volatile memory. If a copy is found, then the test fails.

- **Test 2**: Applied to each key help in non-volatile memory and subject to destruction by the TOE. The evaluator shall use special tools (as needed), provided by the TOE developer if necessary, to ensure the tests function as intended.

  1. Identify the purpose of the key and what access should fail when it is deleted. (e.g. the data encryption key being deleted would cause data decryption to fail.)
  2. Cause the TOE to clear the key.
  3. Have the TOE attempt the functionality that the cleared key would be necessary for.

  The test succeeds if step 3 fails.

- **Test 3**: Applied to each key held in non-volatile memory and subject to destruction by overwrite by the TOE. The evaluator shall use a tool that provides a logical view of the media (e.g., MBR file system):
  1. Record the value of the key in the TOE subject to clearing.
  2. Cause the TOE to perform a normal cryptographic processing with the key from Step #1.
  3. Cause the TOE to clear the key.
  4. Search the logical view that the key was stored in for instances of the known key value from Step #1. If a copy is found, then the test fails.
- **Test 4**: Applied to each key held as non-volatile memory and subject to destruction by overwrite by the TOE. The evaluator shall use a tool that provides a logical view of the media:
  1. Record the logical storage location of the key in the TOE subject to clearing.
  2. Cause the TOE to perform a normal cryptographic processing with the key from Step #1.
  3. Cause the TOE to clear the key.
  4. Read the logical storage location in Step #1 of non-volatile memory to ensure the appropriate pattern is utilized.

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

The evaluator will verify that the AGD documents contains instructions required to configure the OS to use the required modes and key sizes. The evaluator will execute all instructions as specified to configure the OS to the appropriate state. The evaluator will perform all of the following tests for each algorithm implemented by the OS and used to satisfy the requirements of this PP:

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 will 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 will supply a set of 10 plaintext values and obtain the ciphertext value that results from AESCBC 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 will 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 will 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 will perform the same test as for encrypt, using an all-zero ciphertext value as input and AESCBC decryption.

- **KAT-3**. To test the encrypt functionality of AES-CBC, the evaluator will 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 will 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 will 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 will 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 will test the encrypt functionality by encrypting an i-block message where 1 < i ≤ 10. The evaluator will 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 will also test the decrypt functionality for each mode by decrypting an i-block message where 1 < i ≤10. The evaluator will 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 will 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 will 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-GCM Monte Carlo Tests

The evaluator will 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 will 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 will 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 will compare the resulting values to those obtained by submitting the same inputs to a known good implementation.

AES-CCM Tests

The evaluator will 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 216 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 will 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 will 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 will 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 will 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 will 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 will 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 will supply 10 tuples that should FAIL and 5 that should PASS per set of 15.

Additionally, the evaluator will 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 AESCCMP 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 Test

The evaluator will 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 will 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 will 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 will compare the resulting values to those obtained by submitting the same inputs to a known good implementation.

XTS-AES Test

The evaluator will 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 nonzero 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 216 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 will 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 XTSAES decrypt.

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

The evaluator will 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 will use the AES-KW authenticated-encryption function of a known good implementation.

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

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

The evaluator will check that the association of the hash function with other application 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 will perform all of the following tests for each hash algorithm implemented by the TSF and used to satisfy the requirements of this PP.

The following tests require the developer to provide access to a test application that provides the evaluator with tools that are typically not found in the production application.

- **Test 1**: Short Messages Test (Bit oriented Mode) - The evaluator will generate 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 evaluator will 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.

- **Test 2**: Short Messages Test (Byte oriented Mode) - The evaluator will generate 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 evaluator will 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.
- **Test 3**: Selected Long Messages Test (Bit oriented Mode) - The evaluator will generate an input set consisting of m messages, where m is the block length of the hash algorithm. The length of the ith message is $512 + 99 \cdot i$, where $1 \le i \le m$. The message text shall be pseudorandomly generated. The evaluator will 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.
- **Test 4**: Selected Long Messages Test (Byte oriented Mode) - The evaluator will generate 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 $512 + 8 \cdot 99 \cdot i$, where $1 \le i \le m/8$. The message text shall be pseudorandomly generated. The evaluator will 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.
- **Test 5**: Pseudorandomly Generated Messages Test This test is for byte-oriented implementations only. The evaluator will 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 evaluator will then formulate a set of 100 messages and associated digests by following the algorithm provided in Figure 1 of [SHAVS]. The evaluator will then ensure that the correct result is produced when the messages are provided to the TSF.

5.2.2.2.6 Cryptographic Operation for Signing (FCS_COP.1(SIGN))

The evaluator will perform the following activities based on the selections in the ST.

The following tests require the developer to provide access to a test application that provides the evaluator with tools that are typically not found in the production application.

ECDSA Algorithm Tests

- **Test 1**: ECDSA FIPS 186-4 Signature Generation Test. For each supported NIST curve (i.e., P-256, P-384 and P-521) and SHA function pair, the evaluator will 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 will use the signature verification function of a known good implementation.
- **Test 2**: ECDSA FIPS 186-4 Signature Verification Test. For each supported NIST curve (i.e., P-256, P-384 and P-521) and SHA function pair, the evaluator will 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 will verify that 5 responses indicate success and 5 responses indicate failure.

RSA Signature Algorithm Tests

- **Test 1**: Signature Generation Test. The evaluator will verify the implementation of RSA Signature Generation by the OS 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 will have the OS use its private key and modulus value to sign these messages. The evaluator will verify the correctness of the TSF's signature using a known good implementation and the associated public keys to verify the signatures.
- **Test 2**: Signature Verification Test. The evaluator will perform the Signature Verification test to verify the ability of the OS to recognize another party's valid and invalid signatures. The evaluator will 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 evaluator will verify that the OS returns failure when validating each signature.

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

The evaluator will perform the following activities based on the selections in the ST.

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

5.2.2.2.8 Random Bit Generation (FCS_RBG_EXT.1)

FCS_RBG_EXT.1.1

- Test 1: The evaluator will perform 15 trials for the RNG implementation. If the RNG is configurable, the evaluator will perform 15 trials for each configuration. The evaluator will 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 will 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 SP 800-90A).

  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 will 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 values should be set/generated as described.

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 less than or equal to 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 will 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.

FCS_RBG_EXT.1.2

Documentation shall be produced - and the evaluator will perform the activities - in accordance with Appendix E and the Clarification to the Entropy Documentation and Assessment Annex.

In the future, specific statistical testing (in line with NIST SP 800-90B) will be required to verify the entropy estimates.

5.2.2.2.9 Storage of Sensitive Data (FCS_STO_EXT.1)

The evaluator will check the TSS to ensure that it lists all persistent sensitive data for which the OS provides a storage capability. For each of these items, the evaluator will confirm that the TSS lists for what purpose it can be used, and how it is stored. The evaluator will confirm that cryptographic operations used to protect the data occur as specified in FCS_COP.1(1).

The evaluator will also consult the developer documentation to verify that an interface exists for applications to securely store credentials.

5.2.2.2.10 TLS Client Protocol (FCS_TLSC_EXT.1)

FCS_TLSC_EXT.1.1

The evaluator will check the description of the implementation of this protocol in the TSS to ensure that the cipher suites supported are specified. The evaluator will check the TSS to ensure that the cipher suites specified include those listed for this component. The evaluator will also check the operational guidance to ensure that it contains instructions on configuring the OS so that TLS conforms to the description in the TSS. The evaluator will also perform the following tests:

- **Test 1**: The evaluator will establish a TLS connection using each of the cipher suites 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 cipher suite 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 cipher suite being used (for example, that the cryptographic algorithm is 128-bit AES and not 256-bit AES).

- **Test 2**: The evaluator will 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 will send a server certificate in the TLS connection that does not match the serverselected cipher suite (for example, send a ECDSA certificate while using the TLS_RSA_WITH_AES_128_CBC_SHA cipher suite or send a RSA certificate while using one of the ECDSA cipher suites.) The evaluator will verify that the OS disconnects after receiving the server's Certificate handshake message.
- **Test 4**: The evaluator will configure the server to select the TLS_NULL_WITH_NULL_NULL cipher suite and verify that the client denies the connection.
- **Test 5**: The evaluator will perform the following modifications to the traffic:
    - **Test 5.1**: Change the TLS version selected by the server in the Server Hello to a non-supported TLS version (for example 1.3 represented by the two bytes 03 04) and verify that the client rejects the connection.
    - **Test 5.2**: Modify at least one byte in the server's nonce in the Server Hello handshake message, and verify that the client rejects the Server Key Exchange handshake message (if using a DHE or ECDHE cipher suite) or that the server denies the client's Finished handshake message.
    - **Test 5.3**: Modify the server's selected cipher suite in the Server Hello handshake message to be a cipher suite not presented in the Client Hello handshake message. The evaluator will verify that the client rejects the connection after receiving the Server Hello.
    - **Test 5.4**: If an ECDHE or DHE ciphersuite is selected, modify the signature block in the Server's Key Exchange handshake message, and verify that the client rejects the connection after receiving the Server Key Exchange message."[15]
    - **Test 5.5**: 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.
    - **Test 5.6**: Send a garbled message from the Server after the Server has issued the Change Cipher Spec message and verify that the client denies the connection.

FCS_TLSC_EXT.1.2

The evaluator will ensure that the TSS describes the client's method of establishing all reference identifiers from the application-configured reference identifier, including which types of reference identifiers are supported (e.g. Common Name, DNS Name, URI Name, Service Name, or other application-specific Subject Alternative Names) and whether IP addresses and wildcards are supported.

[15] This protection profile assurance activity was modified as part of NIAP Technical Decision 163.

The evaluator will ensure that this description identifies whether and the manner in which certificate pinning is supported or used by the OS.

The evaluator will verify that the AGD guidance includes instructions for setting the reference identifier to be used for the purposes of certificate validation in TLS.

The evaluator will configure the reference identifier according to the AGD guidance and perform the following tests during a TLS connection:

- **Test 1**: The evaluator will present a server certificate that does not contain an identifier in either the Subject Alternative Name (SAN) or Common Name (CN) that matches the reference identifier. The evaluator will verify that the connection fails.
- **Test 2**: The evaluator will present a server certificate that contains a CN that matches the reference identifier, contains the SAN extension, but does not contain an identifier in the SAN that matches the reference identifier. The evaluator shall verify that the connection fails. The evaluator will repeat this test for each supported SAN type.
- **Test 3**: The evaluator will present a server certificate that contains a CN that matches the reference identifier and does not contain the SAN extension. The evaluator will verify that the connection succeeds.
- **Test 4**: The evaluator will present a server certificate that contains a CN that does not match the reference identifier but does contain an identifier in the SAN that matches. The evaluator will verify that the connection succeeds.
- **Test 5**: The evaluator will perform the following wildcard tests with each supported type of reference identifier:
    - **Test 5.1**: The evaluator will present a server certificate containing a wildcard that is not in the left-most label of the presented identifier (e.g. foo.*.example.com) and verify that the connection fails.
    - **Test 5.2**: The evaluator will present a server certificate containing a wildcard in the left-most label but not preceding the public suffix (e.g. *.example.com). The evaluator will configure the reference identifier with a single left-most label (e.g. foo.example.com) and verify that the connection succeeds. The evaluator will configure the reference identifier without a leftmost label as in the certificate (e.g. example.com) and verify that the connection fails. The evaluator will configure the reference identifier with two left-most labels (e.g. bar.foo.example.com) and verify that the connection fails.
    - **Test 5.3**: The evaluator will present a server certificate containing a wildcard in the left-most label immediately preceding the public suffix (e.g. *.com). The evaluator will configure the reference identifier with a single left-most label (e.g. foo.com) and verify that the connection fails. The evaluator will configure the reference identifier with two left-most labels (e.g. bar.foo.com) and verify that the connection fails.
- **Test 6**: [conditional] If URI or Service name reference identifiers are supported, the evaluator will configure the DNS name and the service identifier. The evaluator will present a server certificate containing the correct DNS name and service identifier in the URIName or SRVName

fields of the SAN and verify that the connection succeeds. The evaluator will repeat this test with the wrong service identifier (but correct DNS name) and verify that the connection fails.

- **Test 7**: [conditional] If pinned certificates are supported the evaluator will present a certificate that does not match the pinned certificate and verify that the connection fails.

FCS_TLSC_EXT.1.3

The evaluator will use TLS as a function to verify that the validation rules in FIA_X509_EXT.1.1 are adhered to and shall perform the following additional test:

- **Test 1**: The evaluator will demonstrate that a peer using a certificate without a valid certification path results in an authenticate failure. Using the administrative guidance, the evaluator will then load the trusted CA certificate(s) needed to validate the peer's certificate, and demonstrate that the connection succeeds. The evaluator then shall delete one of the CA certificates, and show that the connection fails.
- **Test 2**: The evaluator will demonstrate that a peer using a certificate which has been revoked results in an authentication failure.
- **Test 3**: The evaluator will demonstrate that a peer using a certificate which has passed its expiration date results in an authentication failure.
- **Test 4**: the evaluator will demonstrate that a peer using a certificate which does not have a valid identifier shall result in an authentication failure.

5.2.2.2.11 TLS Client Protocol (FCS_TLSC_EXT.2)[16]

The evaluator will verify that TSS describes the supported Elliptic Curves Extension and whether the required behavior is performed by default or may be configured. If the TSS indicates that the supported Elliptic Curves Extension must be configured to meet the requirement, the evaluator will verify that AGD guidance includes configuration of the supported Elliptic Curves Extension.

The evaluator will also perform the following test:

The evaluator shall configure a server to perform ECDHE key exchange using each of the TOE’s supported curves and shall verify that the TOE successfully connects to the server.

5.2.2.2.12 TLS Client Protocol (FCS_TLSC_EXT.3)

The evaluator will verify that TSS describes the signature_algorithm extension and whether the required behavior is performed by default or may be configured. If the TSS indicates that the signature_algorithm extension must be configured to meet the requirement, the evaluator will verify that AGD guidance includes configuration of the signature_algorithm extension.

The evaluator will also perform the following test:

The evaluator will configure the server to send a certificate in the TLS connection that is not supported according to the Client’s HashAlgorithm enumeration within the signature_algorithms extension (for example, send a certificate with a SHA-1 signature). The evaluator will verify that the OS disconnects after receiving the server’s Certificate handshake message.

[16] This protection profile assurance activity was replaced as part of NIAP Technical Decision 244.

5.2.2.2.13 TLS Client Protocol (FCS_TLSC_EXT.4)

The evaluator will ensure that the TSS description required per FIA_X509_EXT.2.1 includes the use of client-side certificates for TLS mutual authentication.

The evaluator will verify that the AGD guidance required per FIA_X509_EXT.2.1 includes instructions for configuring the client-side certificates for TLS mutual authentication.

The evaluator will also perform the following test:

Configure the server to require mutual authentication and then 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 will verify the connection is unsuccessful.

5.2.2.2.14 DTLS Implementation (FCS_DTLS_EXT.1)

FCS_DTLS_EXT.1.1

Test 1: The evaluator will attempt to establish a connection with a DTLS server, observe the traffic with a packet analyzer, and verify that the connection succeeds and that the traffic is identified as DTLS.

Other tests are performed in conjunction with the Assurance Activity listed for FCS_TLSC_EXT.1.

FCS_DTLS_EXT.1.2

The evaluator will perform the assurance activities listed for FCS_TLSC_EXT.1.

5.2.2.3 User Data Protection (FDP)

5.2.2.3.1 Access Controls for Protecting User Data (FDP_ACF_EXT.1)

The evaluator will confirm that the TSS comprehensively describes the access control policy enforced by the OS. The description must include the rules by which accesses to particular files and directories are determined for particular users. The evaluator will inspect the TSS to ensure that it describes the access control rules in such detail that given any possible scenario between a user and a file governed by the OS the access control decision is unambiguous.

The evaluator will create two new standard user accounts on the system and conduct the following tests:

- **Test 1**: The evaluator will authenticate to the system as the first user and create a file within that user's home directory. The evaluator will then log off the system and log in as the second user. The evaluator will then attempt to read the file created in the first user's home directory. The evaluator will ensure that the read attempt is denied.
- **Test 2**: The evaluator will authenticate to the system as the first user and create a file within that user's home directory. The evaluator will then log off the system and log in as the second user. The evaluator will then attempt to modify the file created in the first user's home directory. The evaluator will ensure that the modification is denied.
- **Test 3**: The evaluator will authenticate to the system as the first user and create a file within that user's user directory. The evaluator will then log off the system and log in as the second user. The evaluator will then attempt to delete the file created in the first user's home directory. The evaluator will ensure that the deletion is denied.

- **Test 4**: The evaluator will authenticate to the system as the first user. The evaluator will attempt to create a file in the second user's home directory. The evaluator will ensure that the creation of the file is denied.
- **Test 5**: The evaluator will authenticate to the system as the first user and attempt to modify the file created in the first user's home directory. The evaluator will ensure that the modification of the file is accepted.
- **Test 6**: The evaluator will authenticate to the system as the first user and attempt to delete the file created in the first user's directory. The evaluator will ensure that the deletion of the file is accepted.

5.2.2.3.2 Information Flow Control (FDP_IFC_EXT.1)

The evaluator will verify that the TSS section of the ST describes the routing of IP traffic when a VPN client is enabled. The evaluator will ensure that the description indicates which traffic does not go through the VPN and which traffic does, and that a configuration exists for each in which only the traffic identified by the ST author as necessary for establishing the VPN connection (IKE traffic and perhaps HTTPS or DNS traffic) is not encapsulated by the VPN protocol (IPsec).

5.2.2.4 Identification and Authentication (FIA)

5.2.2.4.1 Authentication Failure Handling (FIA_AFL.1)

FIA_AFL.1.1

The evaluator will set an administrator-configurable threshold for failed attempts, or note the ST-specified assignment. The evaluator will then (per selection) repeatedly attempt to authenticate with an incorrect password, PIN, or certificate until the number of attempts reaches the threshold. Note that the authentication attempts and lockouts must also be logged as specified in FAU_GEN.1.

FIA_AFL.1.2

- **Test 1**: The evaluator will attempt to authenticate repeatedly to the system with a known bad password. Once the defined number of failed authentication attempts has been reached the evaluator will ensure that the account that was being used for testing has had the actions detailed in the assignment list above applied to it. The evaluator will ensure that an event has been logged to the security event log detailing that the account has had these actions applied.
- **Test 2**: The evaluator will attempt to authenticate repeatedly to the system with a known bad certificate. Once the defined number of failed authentication attempts has been reached the evaluator will ensure that the account that was being used for testing has had the actions detailed in the assignment list above applied to it. The evaluator will ensure that an event has been logged to the security event log detailing that the account has had these actions applied.
- **Test 3**: The evaluator will attempt to authenticate repeatedly to the system using both a bad password and a bad certificate. Once the defined number of failed authentication attempts has been reached the evaluator will ensure that the account that was being used for testing has had the actions detailed in the assignment list above applied to it. The evaluator will ensure that an event has been logged to the security event log detailing that the account has had these actions applied.

5.2.2.4.2 Multiple Authentication Mechanisms (FIA_UAU.5)

FIA_UAU.5.1

If user name and password authentication is selected, the evaluator will configure the OS with a known user name and password and conduct the following tests:

- **Test 1**: The evaluator will attempt to authenticate to the OS using the known user name and password. The evaluator will ensure that the authentication attempt is successful.
- **Test 2**: The evaluator will attempt to authenticate to the OS using the known user name but an incorrect password. The evaluator will ensure that the authentication attempt is unsuccessful.

If user name and PIN that releases an asymmetric key is selected, the evaluator will examine the TSS for guidance on supported protected storage and will then configure the TOE or OE to establish a PIN which enables release of the asymmetric key from the protected storage (such as a TPM, a hardware token, or isolated execution environment) with which the OS can interface. The evaluator will then conduct the following tests:

- **Test 1**: The evaluator will attempt to authenticate to the OS using the known user name and PIN. The evaluator will ensure that the authentication attempt is successful.
- **Test 2**: The evaluator will attempt to authenticate to the OS using the known user name but an incorrect PIN. The evaluator will ensure that the authentication attempt is unsuccessful.

If X.509 certificate authentication is selected, the evaluator will generate an X.509v3 certificate for a user with the Client Authentication Enhanced Key Usage field set. The evaluator will provision the OS for authentication with the X.509v3 certificate. The evaluator will ensure that the certificates are validated by the OS as per FIA_X509_EXT.1.1 and then conduct the following tests:

- **Test 1**: The evaluator will attempt to authenticate to the OS using the X.509v3 certificate. The evaluator will ensure that the authentication attempt is successful.
- **Test 2**: The evaluator will generate a second certificate identical to the first except for the public key and any values derived from the public key. The evaluator will attempt to authenticate to the OS with this certificate. The evaluator will ensure that the authentication attempt is unsuccessful.

FIA_UAU.5.2[17]

The evaluator shall ensure that for all authentication mechanisms specified in FIA_UAU.5.1, the TSS shall describe each mechanism provided to support user authentication and the rules describing how the authentication mechanism(s) provide authentication. Example rules are how the authentication mechanism authenticates the user (e.g. how does the TSF verify that the correct password was entered), the result of a successful authentication (i.e. is the user input used to derive or unlock a key) and which authentication mechanism can be used at which authentication factor interfaces (i.e. if there are times, for example, after a reboot, that only specific authentication mechanisms can be used).

[17] The protection profile assurance activities in this section were updated as part of NIAP Technical Decision 246.

The evaluator shall verify that configuration guidance for each authentication mechanism is addressed in the AGD guidance.

Test: For each authentication mechanism rule, the evaluator shall ensure that the authentication mechanism(s) behave as documented in the TSS.

5.2.2.4.3 X.509 Certification Validation (FIA_X509_EXT.1)

FIA_X509_EXT.1.1

The evaluator will 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 functions in FIA_X509_EXT.2.1. The tests for the extendedKeyUsage rules are performed in conjunction with the uses that require those rules. The evaluator will create a chain of at least four certificates: the node certificate to be tested, two Intermediate CAs, and the self-signed Root CA.

- **Test 1**: The evaluator will demonstrate that validating a certificate without a valid certification path results in the function failing. The evaluator will then load a certificate or certificates as trusted CAs needed to validate the certificate to be used in the function, and demonstrate that the function succeeds. The evaluator shall then delete one of the certificates, and show thatthe function fails.
- **Test 2**: The evaluator will demonstrate that validating an expired certificate results in the function failing.
- **Test 3**: The evaluator will test that the OS can properly handle revoked certificates-–conditional on whether CRL, OCSP, or OCSP stapling is selected; if multiple methods are selected, then a test shall be performed for each method. The evaluator will test revocation of the node certificate and revocation of the intermediate CA certificate (i.e. the intermediate CA certificate should be revoked by the root CA). The evaluator will ensure that a valid certificate is used, and that the validation function succeeds. The evaluator then attempts the test with a certificate that has been revoked (for each method chosen in the selection) to ensure when the certificate is no longer valid that the validation function fails.
- **Test 4**: If either OCSP option is selected, the evaluator will configure the OCSP server or use a man-in-themiddle tool to present a certificate that does not have the OCSP signing purpose and verify that validation of the OCSP response fails. If CRL is selected, the evaluator will configure the CA to sign a CRL with a certificate that does not have the cRLsign key usage bit set, and verify that validation of the CRL fails.
- **Test 5**: The evaluator will modify any byte in the first eight bytes of the certificate and demonstrate that the certificate fails to validate. (The certificate should fail to parse correctly.)
- **Test 6**: The evaluator will modify any byte in the last byte of the certificate and demonstrate that the certificate fails to validate. (The signature on the certificate should not validate.)
- **Test 7**: The evaluator will modify any byte in the public key of the certificate and demonstrate that the certificate fails to validate. (The signature on the certificate should not validate.)

FIA_X509_EXT.1.2

The tests described must be performed in conjunction with the other certificate services assurance activities, including the functions in FIA_X509_EXT.2.1. The evaluator will create a chain of at least four certificates: the node certificate to be tested, two Intermediate CAs, and the self-signed Root CA.

- **Test 1**: The evaluator will construct a certificate path, such that the certificate of the CA issuing the OS's certificate does not contain the basicConstraints extension. The validation of the certificate path fails.
- **Test 2**: The evaluator will construct a certificate path, such that the certificate of the CA issuing the OS's certificate has the CA flag in the basicConstraints extension not set. The validation of the certificate path fails.
- **Test 3**: The evaluator will construct a certificate path, such that the certificate of the CA issuing the OS's certificate has the CA flag in the basicConstraints extension set to TRUE. The validation of the certificate path succeeds.

5.2.2.4.4 X.509 Certificate Authentication (FIA_X509_EXT.2)

The evaluator will acquire or develop an application that uses the OS TLS mechanism with an X.509v3 certificate. The evaluator will then run the application and ensure that the provided certificate is used to authenticate the connection.

The evaluator will repeat the activity for any other selections listed.

5.2.2.5 Security Management (FMT)

5.2.2.5.1 Management of Security Functions Behavior (FMT_MOF_EXT.1)[18]

The evaluator shall verify that the TSS describes those management functions that are restricted to Administrators, including how the user is prevented from performing those functions, or not able to use any interfaces that allow access to that function.

- **Test 1**: For each function that is indicated as restricted to the administrator, the evaluation shall perform the function as an administrator, as specified in the Operational Guidance, and determine that it has the expected effect as outlined by the Operational Guidance and the SFR. The evaluator shall then perform the function (or otherwise attempt to access the function) as a non-administrator and observe that they are unable to invoke that functionality.

5.2.2.5.2 Specification of Management Functions (FMT_SMF_EXT.1)

The evaluator will verify that every management function captured in the ST is described in the operational guidance and that the description contains the information required to perform the management duties associated with the management function. The evaluator will test the operating system's ability to provide the management functions by configuring the operating system and testing each option selected from above. The evaluator is expected to test these functions in all the ways in which the ST and guidance documentation state the configuration can be managed.

[18] The protection profile assurance activities in this section were updated as part of NIAP Technical Decision 104.

5.2.2.6 Protection of the TSF (FPT)

5.2.2.6.1 Access Controls (FPT_ACF_EXT.1)

FPT_ACF_EXT.1.1

The evaluator will confirm that the TSS specifies the locations of kernel drivers/modules, security audit logs, shared libraries, system executables, and system configuration files. Every file does not need to be individually identified, but the system's conventions for storing and protecting such files must be specified. The evaluator will create an unprivileged user account. Using this account, the evaluator will ensure that the following tests result in a negative outcome (i.e., the action results in the OS denying the evaluator permission to complete the action):

- **Test 1**: The evaluator will attempt to modify all kernel drivers and modules.
- **Test 2**: The evaluator will attempt to modify all security audit logs generated by the logging subsystem.
- **Test 3**: The evaluator will attempt to modify all shared libraries that are used throughout the system.
- **Test 4**: The evaluator will attempt to modify all system executables.
- **Test 5**: The evaluator will attempt to modify all system configuration files.
- **Test 6**: The evaluator will attempt to modify any additional components selected.

FPT_ACF_EXT.1.2

The evaluator will create an unprivileged user account. Using this account, the evaluator will ensure that the following tests result in a negative outcome (i.e., the action results in the OS denying the evaluator permission to complete the action):

- **Test 1**: The evaluator will attempt to read security audit logs generated by the auditing subsystem
- **Test 2**: The evaluator will attempt to read system-wide credential repositories
- **Test 3**: The evaluator will attempt to read any other object specified in the assignment

5.2.2.6.2 Address Space Layout Randomization (FPT_ASLR_EXT.1)

The evaluator will select 3 executables 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 executables on two separate instances of the OS on identical hardware and compare all memory mapping locations. The evaluator will ensure that no memory mappings are placed in the same location. If the rare chance occurs that two mappings are the same for a single executable and not the same for the other two, the evaluator will repeat the test with that executable to verify that in the second test the mappings are different.

5.2.2.6.3 Stack Buffer Overflow Protection (FPT_SBOP_EXT.1)

The evaluator will determine that the TSS contains a description of stack-based buffer overflow protections used by the OS. Example implementations may be activated through compiler options such as "-fstack-protector-all", "fstack-protector", and "/GS" flags. These are referred to by a variety of terms, such as stack cookie, stack guard, and stack canaries. The TSS must include a rationale for any binaries that are not protected in this manner.

- Test 1: The evaluator will inventory the kernel, libraries, and application binaries to determine those that do not implement stack-based buffer overflow protections. This list should match up with the list provided in the TSS.

5.2.2.6.4 Software Restriction Policies (FPT_SRP_EXT.1)

For each selection specified in the ST, the evaluator will ensure that the corresponding tests result in a negative outcome (i.e., the action results in the OS denying the evaluator permission to complete the action):

- Test 1: The evaluator will configure the OS to only allow code execution from the core OS directories. The evaluator will then attempt to execute code from a directory that is in the allowed list. The evaluator will ensure that the code they attempted to execute has been executed.
- Test 2: The evaluator will configure the OS to only allow code execution from the core OS directories. The evaluator will then attempt to execute code from a directory that is not in the allowed list. The evaluator will ensure that the code they attempted to execute has not been executed.
- Test 3: The evaluator will configure the OS to only allow code that has been signed by the OS vendor to execute. The evaluator will then attempt to execute code signed by the OS vendor. The evaluator will ensure that the code they attempted to execute has been executed.
- Test 4: The evaluator will configure the OS to only allow code that has been signed by the OS vendor to execute. The evaluator will then attempt to execute code signed by another digital authority. The evaluator will ensure that the code they attempted to execute has not been executed.
- Test 5: The evaluator will configure the OS to allow execution of a specific application based on version. The evaluator will then attempt to execute the same version of the application. The evaluator will ensure that the code they attempted to execute has been executed.
- Test 6: The evaluator will configure the OS to allow execution of a specific application based on version. The evaluator will then attempt to execute an older version of the application. The evaluator will ensure that the code they attempted to execute has not been executed.
- Test 7: The evaluator will configure the OS to allow execution based on the hash of the application executable. The evaluator will then attempt to execute the application with the matching hash. The evaluator will ensure that the code they attempted to execute has been executed.
- Test 8: The evaluator will configure the OS to allow execution based on the hash of the application executable. The evaluator will modify the application in such a way that the application hash is changed. The evaluator will then attempt to execute the application with the matching hash. The evaluator will ensure that the code they attempted to execute has not been executed.

5.2.2.6.5 Boot Integrity (FPT_TST_EXT.1)

The evaluator will verify that the TSS section of the ST includes a comprehensive description of the boot procedures, including a description of the entire bootchain, for the TSF . The evaluator will ensure that the OS cryptographically verifies each piece of software it loads in the bootchain to include bootloaders and the kernel. Software loaded for execution directly by the platform (e.g. first-stage bootloaders) is out of scope. For each additional category of executable code verified before execution, the evaluator will verify that the description in the TSS describes how that software is cryptographically verified.

The evaluator will verify that the TSS contains a description of the protection afforded to the mechanism performing the cryptographic verification.

The evaluator will perform the following tests:

- **Test 1**: The evaluator will perform actions to cause TSF software to load and observe that the integrity mechanism does not flag any executables as containing integrity errors and that the OS properly boots.
- **Test 2**: The evaluator will modify a TSF executable that is part of the bootchain verified by the TSF (i.e. Not the first-stage bootloader) and attempt to boot. The evaluator will ensure that an integrity violation is triggered and the OS does not boot (Care must be taken so that the integrity violation is determined to be the cause of the failure to load the module, and not the fact that in such a way to invalidate the structure of the module.).
- **Test 3**: If the ST author indicates that the integrity verification is performed using a public key, the evaluator will verify that the update mechanism includes a certificate validation according to FIA_X509_EXT.1. The evaluator will digitally sign the TSF executable with a certificate that does not have the Code Signing purpose in the extendedKeyUsage field and verify that an integrity violation is triggered. The evaluator shall repeat the test using a certificate that contains the Code Signing purpose and verify that the integrity verification succeeds. Ideally, the two certificates should be identical except for the extendedKeyUsage field.

5.2.2.6.6 Trusted Update (FPT_TUD_EXT.1)

FPT_TUD_EXT.1.1

The evaluator will check for an update using procedures described in the documentation and verify that the OS provides a list of available updates. Testing this capability may require installing and temporarily placing the system into a configuration in conflict with secure configuration guidance which specifies automatic update. (The evaluator is also to ensure that this query occurs over a trusted channel as described in FTP_ITC_EXT.1.)

FPT_TUD_EXT.1.2

For the following tests, the evaluator will initiate the download of an update and capture the update prior to installation. The download could originate from the vendor's website, an enterprise-hosted update repository, or another system (e.g. network peer). All supported origins for the update must be indicated in the TSS and evaluated.

- **Test 1**: The evaluator will ensure that the update has a digital signature belonging to the vendor prior to its installation. The evaluator will modify the downloaded update in such a way that the digital signature is no longer valid. The evaluator will then attempt to install the modified update. The evaluator will ensure that the OS does not install the modified update.

- **Test 2**: The evaluator will ensure that the update has a digital signature belonging to the vendor. The evaluator will then attempt to install the update (or permit installation to continue). The evaluator will ensure that the OS successfully installs the update.

5.2.2.6.7 Trusted Update for Application Software (FPT_TUD_EXT.2)

FPT_TUD_EXT.2.1

The evaluator will check for updates to application software using procedures described in the documentation and verify that the OS provides a list of available updates. Testing this capability may require temporarily placing the system into a configuration in conflict with secure configuration guidance which specifies automatic update. (The evaluator is also to ensure that this query occurs over a trusted channel as described in FTP_ITC_EXT.1.)

FPT_TUD_EXT.2.2

The evaluator will initiate an update to an application. This may vary depending on the application, but it could be through the application vendor's website, a commercial app store, or another system. All origins supported by the OS must be indicated in the TSS and evaluated. However, this only includes those mechanisms for which the OS is providing a trusted installation and update functionality. It does not include user or administrator driven download and installation of arbitrary files.

- **Test 1**: The evaluator will ensure that the update has a digital signature which chains to the OS vendor or another trusted root managed through the OS. The evaluator will modify the downloaded update in such a way that the digital signature is no longer valid. The evaluator will then attempt to install the modified update. The evaluator will ensure that the OS does not install the modified update.
- **Test 2**: The evaluator will ensure that the update has a digital signature belonging to the OS vendor or another trusted root managed through the OS. The evaluator will then attempt to install the update. The evaluator will ensure that the OS successfully installs the update.

5.2.2.7 TOE Access (FTA)

5.2.2.7.1 Default TOE Access Banners (FTA_TAB.1)

The evaluator will configure the OS, per instructions in the OS manual, to display the advisory warning message "TEST TEST Warning Message TEST TEST". The evaluator will then log out and confirm that the advisory message is displayed before logging in can occur.

5.2.2.8 Trusted Path / Channels (FTP)

5.2.2.8.1 Trusted Channel Communication (FTP_ITC_EXT.1)

The evaluator will configure the OS to communicate with another trusted IT product as identified in the second selection. The evaluator will monitor network traffic while the OS performs communication with each of the servers identified in the second selection. The evaluator will ensure that for each session a trusted channel was established in conformance with the protocols identified in the first selection.

5.2.2.8.2 Trusted Path (FTP_TRP.1)

The evaluator will examine the TSS to determine that the methods of remote OS administration are indicated, along with how those communications are protected. The evaluator will also confirm that all protocols listed in the TSS in support of OS administration are consistent with those specified in the requirement, and are included in the requirements in the ST. The evaluator will confirm that the operational guidance contains instructions for establishing the remote administrative sessions for each supported method. The evaluator will also perform the following tests:

- **Test 1**: The evaluator will ensure that communications using each remote administration method is tested during the course of the evaluation, setting up the connections as described in the operational guidance and ensuring that communication is successful.
- **Test 2**: For each method of remote administration supported, the evaluator will follow the operational guidance to ensure that there is no available interface that can be used by a remote user to establish a remote administrative sessions without invoking the trusted path.
- **Test 3**: The evaluator will ensure, for each method of remote administration, the channel data is not sent in plaintext.
- **Test 4**: The evaluator will ensure, for each method of remote administration, modification of the channel data is detected by the OS.

6 TOE Summary Specification (TSS)

This chapter describes the Windows 10 security functions that satisfy the security functional requirements of the 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.

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:

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

6.1 Audit

The TOE Audit security function performs:

- Audit Collection
- Selective Audit
- Audit Log Overflow Protection
- Audit Log Restricted Access Protection

6.1.1 Audit Collection

The Windows Event Log service creates the security event log, which contains security relevant audit records collected on a system, along with other event logs which are also registered by other audit entry providers. The Local Security Authority (LSA) server collects audit events from all other parts of the TSF and forwards them to the Windows Event Log service which will place the event into the log for the appropriate provider. While there is no size limit for a single audit record, the authorized administrator can specify a limit for the size of each event log. For each audit event, the Windows Event Log service stores the following data in each audit entry:

Field in Audit Entry	Description
Date	The date the event occurred.
Time	The time the event occurred.
User	The security identifier (SID) of that represents the user on whose behalf the event occurred that represents the user.
Event ID	A unique number within the audit category that identifies the specific audit event.
Source	The Windows component that generated the audit event.

Outcome	Indicates whether the security audit event recorded is the result of a successful or failed attempt to perform the action.
Category	The type of the event defined by the event source.

Table 8 Standard Fields in a Windows Audit Entry

The LSA service defines the following categories for audit events in the security log:

- System,
- Logon / Logoff
- Object Access
- Directory Service Access
- Privilege Use
- Detailed Process Tracking
- Policy Change
- Account Management
- Account Logon

Each audit entry may also contain category-specific data that is contained in the body of the entry as described below:

- For the System Category, the audit entry includes information relating to the system such as the time the audit trail was cleared, start or shutdown of the audit function, and startup and shutdown of Windows. Furthermore, the specific cryptographic operation is identified when such operations are audited.
- For the Logon and Account Logon Category, the audit entry includes the reason the attempted logon failed.
- For the Object Access and the Directory Service Access Category, the audit entry includes the object name and the desired access requested.
- For the Privilege Use Category, the audit entry identifies the privilege.
- For the Detailed Process Tracking Category, the audit event includes the process identifier.
- For the Policy Change and Account Management Category, the audit event includes the new values of the policy or account attributes.
- For the Account Logon Category, the audit event includes the logon type that indicates the source of the logon attempt as one of the following types in the audit record:
    - Interactive (local logon)
    - Network (logon from the network)
    - Service (logon as a service)
    - Batch (logon as a batch job)
    - Unlock (for Unlock screen saver)
    - Network_ClearText (for anonymous authentication to IIS)

There are two places within the TSF where security audit events are collected. Inside the kernel, the Security Reference Monitor (SRM), a part of the NT Executive, is responsible for generation of all audit entries for the object access, privilege use, and detailed process tracking event categories. Windows components can request the SRM to generate an audit record and supply all of the elements in the audit record except for the system time, which the Executive provides. With one exception, audit events for the other event categories are generated by various services that either co-exist in the LSA server or call, with the SeAuditPrivilege privilege, the Authz Report Audit interfaces implemented in the LSA Policy subcomponent. The exception is that the Event Log Service itself records an event record when the security log is cleared and when the security log exceeds the warning level configured by the authorized administrator.

The LSA server maintains an audit policy in its database that determines which categories of events are actually collected. Defining and modifying the audit policy is restricted to the authorized administrator. The authorized administrator can select events to be audited by selecting the category or categories to be audited. An authorized administrator can individually select each category. Those services in the security process determine the current audit policy via direct local function calls. The only other TSF component that uses the audit policy is the SRM in order to record object access, privilege use, and detailed tracking audit. LSA and the SRM share a private local connection port, which is used to pass the audit policy to the SRM. When an authorized administrator changes the audit policy, the LSA updates its database and notifies the SRM. The SRM receives a control flag indicating if auditing is enabled and a data structure indicating that the events in particular categories to audit.

In addition to the system-wide audit policy configuration, it is possible to define a per-user audit policy using auditpol.exe. This allows individual audit categories (of success or failure) to be enabled or disabled on a per user basis.[19] The per-user audit policy refines the system-wide audit policy with a more precise definition of the audit policy for which events will be audited for a specific user.

Within each category, auditing can be performed based on success, failure, or both. For object access events, auditing can be further controlled based on user/group identify and access rights using System Access Control Lists (SACLs). SACLs are associated with objects and indicate whether or not auditing for a specific object, or object attribute, is enabled.

6.1.2 SFR Summary

- **FAU_GEN.1**: The TOE audit collection is capable of generating audit events for items identified in section **5.1.1.1**. For each audit event the TSF records the date, time, user Security Identifier (SID) or name, logon type (for logon audit records), event ID, source, type, and category.

6.2 Cryptographic Support

6.2.1 Cryptographic Algorithms and Operations

The Cryptography API: Next Generation (CNG) API is designed to be extensible at many levels and agnostic to cryptographic algorithm suites. Windows uses CNG exclusively for its own encryption needs and provides public APIs for external developers. An important feature of CNG is its native

[19] Windows will prevent a local administrator from disabling auditing for local administrator accounts. If an administrator can bypass auditing, they can avoid accountability for such actions as exfiltrating files without authorization.

implementation of the Suite B algorithms, including algorithms for AES (128, 192, 256 key sizes)[20], 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. Windows has different entropy sources (deterministic and nondeterministic) which produce entropy data that is used for random numbers generation. In particular, this entropy data together with other data (such as the nonce) seed the DRBG algorithm. 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). [21]
- 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), (320 bits of entropy on boot, 384 bits thereafter on demand based on an OS timer).
- Random values gathered periodically by calling the RDRAND CPU instruction, (256 bits of entropy).

The entropy data is obtained from the entropy sources in a raw format and is health-tested before using it as input for the DRBG. The main source of entropy in the system is the CPU cycle counter which continuously tracks hardware interrupts. This serves as 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 any entropy to seed, or reseed, the random bit generator. In the same manner, a failure of the TPM chip or the RDRAND instruction for the processor would be a critical error that halts the computer, effectively serving as an on-demand self-test.[22] 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) on start-up. Windows always runs the SP 800-90-mandated self-tests for AES-CTR-DRBG during a reseed when the user chooses to operate Windows in the FIPS validated mode.[23]

[20] Note that the 192-bit key size is not used by Windows but is available to developers.

[21] 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.

[22] In other words, the expected result from the CPU cycle counter, the RDRAND instruction, and the TPM RBG is an apparently random value which will be used as an input to seed the RBG.

[23] Running Windows in FIPS validated mode is required according to the administrative guidance.

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.

The CNG provider for random number generation is the AES_CTR_DRBG, 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). 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 random bit generation). When Windows needs to establish an RSA-based shared secret key it can act both as a sender or recipient, any decryption errors which occur during key establishment are presented to the user at a highly abstracted level, such as a failure to connect.

6.2.2 Cryptographic Algorithm Validation

Table 9 Windows 10 Cryptographic Algorithm Standards and Evaluation Methods

Cryptographic Operation	Standard	Requirement	Evaluation Method
Encryption/Decryption	FIPS 197 AES	FCS_COP.1(SYM)	NIST CAVP #4624, #4625, #4627
	NIST SP 800-38A CBC mode		#4624, #4627
	NIST SP 800-38C CCM mode		#4624, #4625
	NIST SP 800-38E XTS mode		#4624
	NIST SP 800-38F KW mode		#4626
	NIST SP 800-38D GCM mode		#4624
Digital signature (key generation)	FIPS 186-4 RSA	FCS_CKM.1	NIST CAVP #2521, #2522
Digital signature (generation)	FIPS 186-4 RSA	FCS_COP.1(SIGN)	NIST CAVP #2521, #2522, #2524
Digital signature (verification)	FIPS 186-4 RSA	FCS_COP.1(SIGN)	NIST CAVP #2521, #2522, #2523, #2524
Digital signature (key generation, signature generation and verification)	FIPS 186-4 DSA	Added as a prerequisite of NIST CAVP KAS #127	NIST CAVP #1223
Digital signature (key generation, signature generation and verification)	FIPS 186-4 ECDSA	FCS_CKM.1	NIST CAVP #1133, #1135

Hashing	FIPS 180-4 SHA-1 and SHA-256, SHA-384, SHA-512	FCS_COP.1 (HASH)	NIST CAVP #3790
Keyed-Hash Message Authentication Code	FIPS 198-2 HMAC	FCS_COP.1(HMAC)	NIST CAVP #3061, #3062
Random number generation	NIST SP 800-90 CTR_DRBG	FCS_RBG_EXT.1	NIST CAVP #1555, #1556
Key agreement	NIST SP 800-56A ECDH	FCS_CKM.2	NIST CAVP #127, #128
Key establishment	NIST SP 800-56B RSA	FCS_CKM.2	NIST CVL #1280, #1281 Tested by the CC evaluation lab[24]
Key-based key derivation	SP800-108		NIST CAVP #140, #141
IKEv1	SP800-135		NIST CVL #1278
IKEv2	SP800-135		NIST CVL #1278
TLS	SP800-135	FCS_TLSC_EXT.1, FCS_TLSC_EXT.2, FCS_TLSC_EXT.3, FCS_TLSC_EXT.4, FCS_DTLS_EXT.1	NIST CVL #1278

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

- For volatile memory, the overwrite is a single direct overwrite consisting of zeros using the RtlSecureZeroMemory function.

The following table describes the keys and secrets used for networking and data protection; when these ephemeral keys or secrets are no longer needed for a network session, due to either normal end of the session or abnormal termination, or after protecting sensitive data using DPAPI, they are deleted as described above and in section 5.1.2.3. Note that the administrative guidance precludes hibernating the computer and so these keys are not persisted into volatile storage

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

[24] The test results are described in the evaluation and Assurance Activity Report.

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 as well as TPM-based health attestations. The key size can be 2048 or 3072 bits.
Asymmetric DSA Private Keys	Keys used for the calculation of DSA digital signatures (FIPS 186-4) for IPsec and TLS. The key size can be 2048 or 3072 bits.
Asymmetric DSA Public Keys	Keys used for the verification of DSA digital signatures (FIPS 186-4) for IPsec and TLS. The key size can be 2048 or 3072 bits.
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.
DPAPI master secret	512-bit random value used by DPAPI
DPAPI master AES key	256-bit encryption key that protects the DPAPI master secret
DPAPI AES key	256-bit encryption key used by DPAPI
DRBG seed	Seed for the main DRBG, zeroized during reseeding

Table 10 Types of Keys Used by Windows

6.2.3 Networking (TLS)

The TOE implements TLS to enable a trusted network path that is used for client and server authentication, as well as HTTPS.

The following table summarizes the TLS RFCs implemented in 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 1.1.
4366	Transport Layer Security (TLS) Extensions	Obsoletes RFC 3546 Requirements for TLS 1.1 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.

Table 11 TLS RFCs Implemented by Windows

These protocols are described at:

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

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_CBC_SHA as defined in RFC 4492
- TLS_ECDHE_ECDSA_WITH_AES_256_CBC_SHA as defined in RFC 4492
- TLS_ECDHE_ECDSA_WITH_AES_128_GCM_SHA256 as defined in RFC 5289
- TLS_ECDHE_ECDSA_WITH_AES_256_GCM_SHA384 as defined in RFC 5289
- TLS_ECDHE_ECDSA_WITH_AES_128_CBC_SHA256 as defined in RFC 5289
- TLS_ECDHE_ECDSA_WITH_AES_256_CBC_SHA384 as defined in RFC 5289.

When negotiating a TLS 1.2 elliptic curve cipher suite, Windows will include automatically as part of the Client Hello message both its supported elliptic curves extension, i.e., secp256r1, secp384r1, and secp521r1 as well as signature algorithm, i.e., SHA256, SHA384, and SHA512 based on the ciphersuites selected by the administrator**.** By default, the curve secp521r1 is disabled. This curve can be enabled adding its name in the ECC Curve Order file. In addition, the curve priority can be edited in this file**.**

On the other hand, by default the signature algorithms in the Client Hello message are: SHA1, SHA256, SHA384 and SHA512. The signature algorithm extension is configurable by editing a registry key to meet with the FCS_TLSC_EXT.3 requirement. 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 any applicable extended key usage identifiers. 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. The reference identifier in Windows 10 for TLS is the DNS name or IP address of the remote server, which is compared against the DNS name as presented identifier in the Subject Alternative Name (SAN) or the Subject Name of the certificate. There is no configuration of the reference identifier.

A certificate that uses a wildcard in the leftmost portion of the resource identifier (i.e., *.contoso.com) can be accepted for authentication, otherwise the certificate will be deemed invalid. Windows does not provide a general-purpose capability to "pin" TLS certificates.

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.

6.2.4 Protecting Data with DPAPI

Windows provides the Data Protection API, DPAPI, which Windows components, first-party and third-party applications can use to protect any persisted data which the developer deems to be sensitive. DPAPI will use AES CBC encryption with a key that is based in part on the user's password to protect the user data. When storing private keys and secrets associated with the user account, the encrypted data is stored on the file system in a directory which is part of the user's profile.

6.2.5 SFR Summary

- **FCS_CKM.1,[25] FCS_CKM.2,[26] FCS_COP.1(SYM), FCS_COP.1(HASH), FCS_COP.1(SIGN), FCS_COP.1(HMAC), FCS_RBG_EXT.1**: See **Table 9 Windows 10 Cryptographic Algorithm Standards and Evaluation Methods**.
- **FCS_CKM.4:** Windows overwrites critical cryptographic parameters immediately after that data is no longer needed.
- **FCS_STO_EXT.1**: Windows provides the Data Protection API (DPAPI) for developers to encrypt and decrypt sensitive data using the CryptProtectData and CryptUnprotectData interfaces.
- **FCS_TLS_EXT.1, FCS_TLS_EXT.2, FCS_TLS_EXT.3, FCS_TLS_EXT.4**: Windows implements TLS 1.2 to provide server and mutual authentication using X.509v3 certificates, confidentiality and integrity to upper-layer protocols such as Extensible Authentication Protocol and HTTP.

[25] In the context of this evaluation, Windows will generate RSA and ECC key pairs as part of establishing a TLS session.

[26] In the context of this evaluation, Windows will generate RSA and ECC key pairs as part of establishing a TLS session.

- **FCS_DTLS_EXT.1**: The Windows implementation of DTLS 1.0 and DTLS 1.2 is based on underlying SChannel component which implements TLS

6.3 User Data Protection

6.3.1 Discretionary Access Control

The executive component within the Windows kernel mediates access between subjects and user data objects, also known as named objects. Subjects consist of processes with one or more threads running on behalf of users. While the Windows Discretionary Access Control policy manages several different kinds of named objects, the protection profile that is the basic for this evaluation focuses on the NTFS File and NTFS Directory objects.

6.3.1.1 Subject DAC Attributes

Windows security access tokens contain the security attributes for a subject. Tokens are associated with processes and threads running on behalf of the user. Information in a security access token that is used by DAC includes:

- The Security Identifier (SID) for the user account
- SIDs representing groups for which the user is a member
- Privileges assigned to the user
- An owner SID that identifies the SID to assign as owner for newly created objects
- A default Discretionary Access Control List (DACL) for newly created objects
- Token type which is either a primary or an impersonation token
- The impersonation level (for impersonation tokens)
- The integrity label SID
- An optional list of restricting SIDs
- The logon SID that identifies the logon session.

An administrator can change all of these except for the user account SID and logon SID.

A thread can be assigned an impersonation token that would be used instead of the process' primary token when making an access check and generating audit data. Hence, that thread is impersonating the client that provided the impersonation token. Impersonation stops when the impersonation token is removed from the thread or when the thread terminates.

An access token may also include a list of restricting SIDs which are used to limit access to objects. Restricting SIDs are contained in restricted tokens, (which is a special form of a thread impersonation token), and when configured serve to limit the corresponding process access to no more than that available to the restricted SID.

Access decisions are made using the impersonation token of a thread if it exists, and otherwise the thread's process primary token (which always exists).

6.3.1.2 Object DAC Attributes

Security Descriptors (SDs) contain all of the security attributes associated with an object. All named objects have an associated SD. The security attributes from a SD used for discretionary access control are the object owner SID which specifies the owner of the security descriptor, the DACL present flag, and the DACL itself, when present.

DACLs contain a list of Access Control Entries (ACEs). Each ACE specifies an ACE type, a SID representing a user or group, and an access mask containing a set of access rights. Each ACE has inheritance attributes associated with it that specify if the ACE applies to the associated object only, to its children objects only, or to both its children objects and the associated object.

There are two types of ACEs that apply to discretionary access control:

- ALLOW ACES
    - ACCESS_ALLOWED_ACE: used to grant access to a user or group of users.
    - ACCESS_ALLOWED_OBJECT_ACE: (for DS objects) used to grant access for a user or group to a property or property set on the directory service object, or to limit the ACE_inheritance to a specified type of child object. This ACE type is only supported for directory service objects.
- DENY ACES
    - ACCESS_DENIED_ACE: used to deny access to a user or group of users.
    - ACCESS_DENIED_OBJECT_ACE: (for DS objects) used to deny access for a user or group to a property or property set on the directory service object or to limit the ACE_inheritance to a specified type of child object. This ACE type is only supported for directory service objects.

In the ACE, an access mask contains object access rights granted (or denied) to the SID, representing a user or group. An access mask is also used to specify the desired access to an object when accessing the object and to identify granted access associated with an opened object. Each bit in an access mask represents a particular access right. There are four categories of access rights: standard, specific, special, and generic. Standard access rights apply to all object types. Specific access rights have different semantic meanings depending on the type of object. Special access rights are used in desired access masks to request special access or to ask for all allowable rights. Generic access rights are convenient groupings of specific and standard access rights. Each object type provides its own mapping between generic access rights and the standard and specific access rights.

For most objects, a subject requests access to the object (e.g., opens it) and receives a pointer to a handle in return. The TSF associates a granted access mask with each opened handle. For kernel-mode objects, handles are maintained in a kernel-mode handle table. There is one handle table per process; each entry in the handle table identifies an opened object and the access rights granted to that object. For user-mode TSF servers, the handle is a server-controlled context pointer associated with the connection between the subject and the server. The server uses this context handle in the same manner as with the kernel mode (i.e., to locate an opened object and it’s associated granted access mask). In both cases (user and kernel-mode objects), the SRM makes all access control decisions.

The following table summarizes every DAC access right for each named object which were tested by the evaluation lab:

Named Object	Access Rights
NTFS Directory	ACCESS_SYSTEM_SECURITY READ_CONTROL WRITE_DAC WRITE_OWNER SYNCHRONIZE FILE_LIST_DIRECTORY FILE_ADD_FILE FILE_ADD_SUBDIRECTORY FILE_DELETE_CHILD FILE_READ_ATTRIBUTES FILE_WRITE_ATTRIBUTES FILE_DELETE_CHILD|FILE_ADD_FILE DELETE
NTFS File	ACCESS_SYSTEM_SECURITY READ_CONTROL WRITE_DAC WRITE_OWNER SYNCHRONIZE FILE_WRITE_DATA FILE_READ_DATA FILE_APPEND_DATA FILE_WRITE_EA FILE_EXECUTE FILE_READ_ATTRIBUTES FILE_WRITE_ATTRIBUTES FILE_WRITE_ATTRIBUTES. FILE_WRITE_DATA and FILE_WRITE_ATTRIBUTES. DELETE FILE_WRITE_DATA | FILE_READ_DATA FILE_READ_DATA | FILE_EXECUTE FILE_READ_DATA | FILE_EXECUTE | FILE_WRITE_DATA FILE_WRITE_DATA | FILE_WRITE_EA | FILE_WRITE_ATTRIBUTES

Table 12 DAC Access Rights and Named Objects

6.3.1.3 DAC Enforcement Algorithm

The TSF enforces the DAC policy to objects based on SIDs and privileges in the requestor's token, the desired access mask requested, and the object's security descriptor.

Below is a summary of the algorithm used to determine whether a request to access a user data object is allowed. In order for access to be granted, all access rights specified in the desired access mask must be granted by one of the following steps. At the end of any step, if all of the requested access rights have been granted then access is allowed. At the end of the algorithm, if any requested access right has not been granted, then access is denied.

1. Privilege Check:

   a. Check for SeSecurity privilege: This is required if ACCESS_SYSTEM_SECURITY is in the desired access mask. If ACCESS_SYSTEM_SECURITY is requested and the requestor does not have this privilege, access is denied. Otherwise ACCESS_SYSTEM_SECURITY is granted.
   b. Check for SeTakeOwner privilege: If the desired mask has WRITE_OWNER access right, and the privilege is found in the requestor's token, then WRITE_OWNER access is granted.
   c. Check for SeBackupPrivilege: The Backup Files and Directories privilege allows a subject process to read files and registry objects for backup operations regardless of their ACE in the DACL. If the subject process has the SeBackupPrivilege privilege and the operation requires the privilege, no further checking is performed and access is allowed. Otherwise this check is irrelevant and the access check proceeds.
   d. Check for SeRestorePrivilege: The Restore Files and Directories privilege allows a subject process to write files and registry objects for restore operations regardless of their ACE in the DACL. If the subject process has the SeRestorePrivilege privilege and the operation requires the privilege no further checking is performed, and access is allowed. Otherwise this check is irrelevant and the access check proceeds.
2. Owner Check:
   a. If the DACL contains one or more ACEs with the OwnerRights SID, those entries, along with all other applicable ACEs for the user, are used to determine the owner's rights.
   b. Otherwise, check all the SIDs in the token to determine if there is a match with the object owner. If so, the READ_CONTROL and WRITE_DAC rights are granted if requested.
3. DACL not present:
   a. All further access rights requested are granted.
4. DACL present but empty:
   a. If any additional access rights are requested, access is denied.
5. Iteratively process each ACE in the order that they appear in the DACL as described below:
   a. If the inheritance attributes of the ACE indicate the ACE is applicable only to children objects of the associated object, the ACE is skipped.
   b. If the SID in the ACE does not match any SID in the requestor's access token, the ACE is skipped.
   c. If a SID match is found, and the access mask in the ACE matches an access in the desired access mask:
      i. Access Allowed ACE Types: If the ACE is of type ACCESS_ALLOWED_OBJECT_ACE and the ACE includes a GUID representing a property set or property associated with the object, then the access is granted to the property set or specific property represented by the GUID (rather than to the entire object). Otherwise the ACE grants access to the entire object.
      ii. Access Denied ACE Type: If the ACE is of type ACCESS_DENIED_OBJECT_ACE and the ACE includes a GUID representing a property set or property associated with the object, then the access is denied to the property set or specific property

represented by the GUID. Otherwise the ACE denies access to the entire object. If a requested access is specifically denied by an ACE, then the entire access request fails.

6. If all accesses are granted but the requestor's token has at least one restricting SID, the complete access check is performed against the restricting SIDs. If this second access check does not grant the desired access, then the entire access request fails.

6.3.1.4 Default DAC Protection

The TSF provides a process ensuring a DACL is applied by default to all new objects. When new objects are created, the appropriate DACL is constructed. The default DAC protections for DS objects and non-DS objects are slightly different.

The TOE uses the following rules to set the DACL in the SDs for new named kernel objects:

- The object's DACL is the DACL from the SD specified by the creating process. The TOE merges any inheritable ACEs into the DACL unless SE_DACL_PROTECTED is set in the SD control flags. The TOE then sets the SE_DACL_PRESENT SD control flag. Note that a creating process can explicitly provide a SD that includes no DACL. The result will be an object with no protections. This is distinct from providing no SD which is described below.
- If the creating process does not specify a SD, the TOE builds the object's DACL from inheritable ACEs in the parent object's DACL. The TOE then sets the SE_DACL_PRESENT SD control flag.
- If the parent object has no inheritable ACEs, the TOE uses its object manager subcomponent to provide a default DACL. The TOE then sets the SE_DACL_PRESENT and SE_DACL_DEFAULTED SD control flags.
- If the object manager does not provide a default DACL, the TOE uses the default DACL in the subject's access token. The TOE then sets the SE_DACL_PRESENT and SE_DACL_DEFAULTED SD control flags.
- The subject's access token always has a default DACL, which is set by the LSA subcomponent when the token is created.

The method used to build a DACL for a new DS object is slightly different. There are two key differences, which are as follows:

- The rules for creating a DACL distinguish between generic inheritable ACEs and object-specific inheritable ACEs in the parent object's SD. Generic inheritable ACEs can be inherited by all types of child objects. Object-specific inheritable ACEs can be inherited only by the type of child object to which they apply.
- The AD schema definition for the object can include a SD. Each object class defined in the schema has a defaultSecurityDescriptor attribute. If neither the creating process nor inheritance from the parent object provides a DACL for a new AD object, the TOE uses the DACL in the default SD specified by the schema.

The TOE uses the following rules to set the DACL in the security descriptor for new DS objects:

- The object's DACL is the DACL from the SD specified by the creating process. The TOE merges any inheritable ACEs into the DACL unless SE_DACL_PROTECTED is set in the SD control flags. The TOE then sets the SE_DACL_PRESENT SD control flag.
- If the creating process does not specify a SD, the TOE checks the parent object's DACL for inheritable object-specific ACEs that apply to the type of object being created. If the parent object has inheritable object-specific ACEs for the object type, the TOE builds the object's DACL from inheritable ACEs, including both generic and object-specific ACEs. It then sets the SE_DACL_PRESENT SD control flag.
- If the parent object has no inheritable object-specific ACEs for the type of object being created, the TOE uses the default DACL from the AD schema for that object type. It then sets the SE_DACL_PRESENT and SE_DACL_DEFAULTED SD control flags.
- If the AD schema does not specify a default DACL for the object type, the TOE uses the default DACL in the subject's access token. It then sets the SE_DACL_PRESENT and SE_DACL_DEFAULTED SD control flags.
- The subject's access token always has a default DACL, which is set by the LSA subcomponent when the token is created.

All tokens are created with an appropriate default DACL, which can be applied to the new objects as appropriate. The default DACL is restrictive in that it only allows the SYSTEM SID and the user SID that created the object to have access. The SYSTEM SID is a special SID representing TSF trusted processes.

6.3.1.5 DAC Management

- The following are the four methods that DACL changes are controlled:
    - Object owner: Has implicit WRITE_DAC access.
    - Explicit DACL change access: A user granted explicit WRITE_DAC access on the DACL can change the DACL.
    - Take owner access: A user granted explicit WRITE_OWNER access on the DACL can take ownership of the object and then use the owner’s implicit WRITE_DAC access.
    - Take owner privilege: A user with SeTakeOwner privilege can take ownership of the object and then user the owner’s implicit WRITE_DAC access.

6.3.1.6 Reference Mediation

Access to objects on the system is generally predicated on obtaining a handle to the object. Handles are usually obtained as the result of opening or creating an object. In these cases, the TSF ensures that access validation occurs before creating a new handle for a subject. Handles may also be inherited from a parent process or directly copied (with appropriate access) from another subject. In all cases, before creating a handle, the TSF ensures that that the security policy allows the subject to have the handle (and thereby access) to the object. A handle always has a granted access mask associated with it. This mask indicates, based on the security policy, which access rights to the object that the subject was granted. On every attempt to use a handle, the TSF ensures that the action requested is allowed according to the handle’s granted access mask. In a few cases, such as with DS, objects are directly accessed by name without the intermediate step of obtaining a handle first. In these cases, the TSF checks the request against the access policy directly (rather than checking for a granted access mask).

6.3.2 VPN Client

The Windows IPsec VPN client can be configured by the device local administrator. The administrator can configure the IPsec VPN client that all IP traffic is routed through the IPsec tunnel except for:

- IKE traffic used to establish the VPN tunnel
- IPv4 ARP traffic for resolution of local network layer addresses and to establish a local address
- IPv6 NDP traffic for resolution of local network layer addresses and to establish a local address

The IPsec VPN is an end-to-end internetworking technology and so VPN sessions can be established over physical network protocols such as 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.

Universal Windows App developers can implement their own VPN client if authorized by Microsoft to use the **networkingVpnProvider** capability, which includes setting the policy to lockdown networking traffic as described above.[27]

6.3.3 SFR Summary

- **FDP_ACF_EXT.1**: Windows provides a Discretionary Access Control policy to limit modification and reading of objects by non-authorized users.
- **FDP_IFC_EXT.1**: Windows provides interfaces for developers to implement their own VPN client.

6.4 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.

Windows 10, Windows 10 S, and Surface Hub can authenticate users based on username and password; Windows 10 and Windows 10 can authenticate users with a Windows Hello PIN which is backed by a TPM, smart card; and Windows 10 and Surface Hub can use physical or virtual smart card thus supporting multiple user authentication.

Password-based authentication to the Windows NT OS succeeds when the credential provided by the user matches the stored protected representation of the password; Windows Hello and smart cards both use PIN-based authentication to unlock a protected resource, a private key, the stored representation of the PIN is protected by the Secure Kernel.

[27] See https://msdn.microsoft.com/en-us/library/windows/apps/windows.networking.vpn.aspx.

Password authentication can be used for interactive, service, and network logons and to initiate the "change password" screen; the Windows Hello PIN, physical and virtual smart cards can be used for interactive logons; and the Windows Hello PIN is used to re-authenticate the user when the user chooses to change their PIN.

When the authentication succeeds, the user will be logged onto their desktop, their screen unlocked, or their authentication factors changed depending whether the user logged onto the computer, the display was locked, or the PIN or password was to be changed.

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 will lockout the user account. Windows persists the number of consecutive failed logons on for the user and so rebooting the computer does not reset the failed logon counter. 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.

6.4.1 X.509 Certificate Validation and Generation

Every Windows component that uses X.509 certificates is responsible for performing certificate validation, however all components use a common system subcomponent,[28] which validates certificates as described in RFC 5280, and particular, the specific validation listed in section 5.1.4.3, including all applicable usage constraints such as Server Authentication for networking sessions and Code Signing when installing product updates. Every 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. The Windows component will use the same kinds of information along with a certification path and certificate trust lists as part of deciding to accept 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, however it will inform the user and seek their consent before establishing a HTTPS web browsing session. Certification validation for updates to Windows, mobile applications, and integrity verification is mandatory, neither the administrator nor the user have the option to bypass the results of a failed certificate validation; software installation and updates is further described in **Windows and Application Updates**.

When Windows needs to generate a certificate enrollment request it will include a distinguished name, information about the cryptographic algorithms used for the request, any certification extensions, and information about the client requesting the certificate.

[28] See https://msdn.microsoft.com/en-us/library/windows/desktop/aa380252(v=vs.85).aspx for the win32 interface description for this component.

6.4.2 SFR Summary

- **FIA_AFL.1**: After the number of consecutive failed authentication attempts for a user account has been surpassed, Windows can be configured to lockout the user account.
- **FIA_UAU.5**: Windows provides authentication using a username and password.
- **FIA_X509_EXT.1**: 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 TLS and HTTPS.

6.5 Security Management

The complete set of management functions are described in Security Management (FMT), the following table maps which activities can be done by a standard Windows user or a local administrator. A checkmark indicates which entity can invoke the management function. Standard users, or programs running on their behalf, are not able to modify policy or configuration that is set by the administrator, the result is that the user cannot override the configuration specified by the administrator.

#	Management Function	Administrator	User
1.	Enable/disable screen lock	√	√
2.	Configure screen lock inactivity timeout	√	√
3.	Configure local audit storage capacity	√	
4.	Configure minimum password Length	√	
5.	~~Configure minimum number of special characters in password~~		
6.	~~Configure minimum number of numeric characters in password~~		
7.	~~Configure minimum number of uppercase characters in password~~		
8.	~~Configure minimum number of lowercase characters in password~~		
9.	Configure remote connection inactivity timeout	√	
10.	~~Enable/disable unauthenticated logon~~		
11.	Configure lockout policy for unsuccessful authentication attempts through [***timeouts between attempts, limiting number of attempts during a time period***]	√	
12.	Configure host-based firewall	√	
13.	Configure name/address of directory server to bind with[29]	√	
14.	Configure name/address of remote management server from which to receive management settings	√	
15.	~~Configure name/address of audit/logging server to which to send audit/logging records~~		
16.	Configure audit rules	√	
17.	Configure name/address of network time server	√	
18.	Enable/disable automatic software update	√	
19.	Configure Wi-Fi interface	√	

[29] For Windows 10 Pro, Windows 10 Enterprise and Surface Hub.

20.	Enable/disable Bluetooth interface	√	
21.	Configure USB interfaces	√	
22.	Enable/disable [**local area network interface**]	√	
23.	[**none**]	√	

6.5.1 SFR Summary

- **FMT_MOF_EXT.1, FMT_SMF_EXT.1**: 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.

6.6 Protection of the TSF

6.6.1 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
- Kernel-mode software
- Trusted user-mode processes
- User-mode Administrative tools process
- Application Containers (Windows 10 only)

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.

Application Containers (“App Containers”) provide an execution environment for Universal Windows Applications which prevents Universal Windows Applications from accessing data created by other

Universal Windows Applications except through brokered operating system services such as the File Picker dialog.

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.

6.6.2 Protection of OS Binaries, Audit and Configuration Data

By default, a Windows operating system is installed into the \Windows\ directory of the first bootable storage partition for the computer. The logical name for this directory is %systemRoot%. The kernel, device drivers (.sys files), system executables (.exe files) and dynamically loadable libraries (.dll files) are stored in the \%systemRoot%\system32 directory and subdirectories below system32. Standard users have permissions to read and execute these files, however modify and write permissions are limited to the local administrator and system service accounts.

The root directory for audit logs is %systemRoot%\system32\winevt. The local administrator, Event Log service, and the system account have full control over the audit files; standard users are not authorized to access the logs.

The primary configuration data store for Windows is the registry, and there are separate registry hives for the computer itself and each user authorized to use the computer. The registry hives for operating system configuration data is located at %systemRoot%\system32\config; the registry hive for the user is located in the user’s profile home directory. Registry files use the same protection scheme as event log files.

6.6.3 Protection From Implementation Weaknesses

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.[30] 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. [31]

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,[32] 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.

6.6.4 Windows Platform Integrity and Code Integrity

A Windows operating system verifies the integrity of Windows program code using the combination of Secure Boot and Code Integrity capabilities in Windows. On computers with a TPM, such as those used in this evaluation, before Windows will boot, the computer will verify the integrity of the early boot components, which includes the Boot Loader, the OS Loader, and the OS Resume binaries.

This capability, known as Secure Boot, checks that the file integrity of early boot components has not been compromised, mitigating the risk of rootkits and viruses, and additionally checks that critical boot-time data have not been modified. Secure Boot collects these file and configuration measurements and seals them to the TPM. When Secure Boot starts in the preboot environment, it will compare the sealed values from the TPM to the measured values from the current boot cycle and if those values do not match the sealed values, Secure Boot will lock the system (which prevents booting) and display a warning on the computer display. While the TPM is part of the external IT environment in this evaluation, the hardware-protected hashes serve as the first step of the chain that provides integrity from the hardware, through the bootchain into the kernel and required device drivers.

When the measurements match, the UEFI firmware will load the OS Boot Manager, which is an Authenticode-signed image file, based on the Portable Executable (PE) image file format. A SHA-256 hash based signature and a public key certificate chain are embedded in the boot manager Authenticode signed image file under the "Certificate" IMAGE_DATA_DIRECTORY of the

[30] The 64-bit version of the Windows microkernel, ntoskrnl.exe, implements Kernel Patch Protection to prevent the modification of kernel data structures which could be exploited by stack-based vulnerabilities.
[31] The PRNG is seeded by the TPM RBG, the RDRAND instruction and other sources.
[32] Winload.exe, winresume.exe, and hvloader.exe are loaded before the stack buffer overrun protection mechanism is operational and therefore are not compiled with this option.

IMAGE_OPTIONAL_HEADER of the file. This public key certificate chain ends in a root public key. The boot manager uses the embedded SHA-256 hash based signature and public key certificate chain to validate its own integrity. A SHA-256 hash of the boot manager image file is calculated for the whole file, with the exception of the following three elements which are excluded from the hash calculation: the CheckSum field in the IMAGE_OPTIONAL_HEADER, the IMAGE_DIRECTORY_ENTRY_SECURITY IMAGE_DATA_DIRECTORY, and the public key certificate table, which always resides at the end of the image file.

If the boot manager is validated, then the root public key of the embedded public key certificate chain must match one of the Microsoft root public keys which indicate that Microsoft is the publisher of the boot manager. These root public keys are necessarily hardcoded in the boot manager. If the boot manager cannot validate its own integrity, then the boot manager does not continue to load other modules and displays an error message.

After the boot manager determines its integrity, it attempts to load one application from the following list of boot applications:

- OS Loader: (Winload.exe or Winload.efi): the boot application started by the boot manager load the Windows kernel to start the boot process
- OS Resume (winresume.exe or winresume.efi): the boot application started by the boot manager to resume the instance of the executing OS which is persisted in the hibernation file "hiberfil.sys"[33]
- A physical memory testing application (memtest.exe) to check the physical memory ICs for the machine are working correctly.[34]

These boot applications are also Authenticode signed image files and so, the Boot Manager uses the embedded trusted SHA-256 hash based signature and public key certificate chain within the boot application's IMAGE_OPTIONAL_HEADER to validate the integrity of the boot application before attempting to load it. Except for three elements which are excluded from the hash calculation (these are the same three elements mentioned above in the Boot Manager description), a hash of a boot application image file is calculated in the same manner as for the Boot Manager.[35]

If the boot application is validated, then the root public key of the embedded public key certificate chain must match one of the hardcoded Microsoft's root public keys. If the boot manager cannot validate the integrity of the boot application, then the boot manager will not load the boot application and instead displays an error message below along with the full name of the boot application that failed the integrity check.

After the boot application's integrity has been determined, the boot manager attempts to load the boot application. If the boot application is successfully loaded, the boot manager then transfers execution to the loaded application.

[33] The evaluated configuration precludes suspending/resuming Windows and so this boot application will not be used when operating Windows per the administrative guidance.

[34] This is considered to be a non-operational mode for the evaluation.

[35] Note that this is an additional integrity check in addition to the TPM measurements check.

After the Winload boot application is loaded, it receives the transfer of execution from the boot manager. During its execution, Winload attempts to load the Windows kernel (ntoskrnl.exe) together with a number of early-launch drivers. Among the modules that Winload must validate in the Portable Executable (PE) image file format, are the cryptography-related modules listed below

- The Windows kernel (ntoskrnl.exe)
- The BitLocker drive encryption filter driver (fvevol.sys)
- The Windows kernel cryptography device driver (cng.sys)
- The Windows code integrity library module (ci.dll)

The four image files above have their trusted SHA hashes stored in catalog files that reside in the local machine catalog directory.

Because they are PKCS #7 SignedData messages, catalog files are signed. The root public key of the certificate chain used to verify the signature of a Microsoft's catalog file 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 Winload boot application.

If the image files are validated, their SHA-256 hashes, as calculated by the Winload boot application, must match their trusted SHA-256 hashes in a Microsoft's catalog file, which has been verified by the Winload boot application. A hash of an image file is calculated for the whole file, with the exception of the following three elements which are excluded from the hash calculation: the CheckSum field in the IMAGE_OPTIONAL_HEADER, the IMAGE_DIRECTORY_ENTRY_SECURITY IMAGE_DATA_DIRECTORY, and the public key certificate table, which always resides at the end of the image file.

Should the Winload boot application be unable to validate the integrity of one of the Windows image files, the Winload boot application does not continue to load other Windows image files. Rather it displays an error message and fails into a non-operational mode. In limited circumstances the pre-boot environment will attempt to repair the boot environment, such as copying files from a repair partition to repair files with integrity errors. When repair is not possible, the boot manager will ask the user to reinstall Windows.

After the initial device drivers have been loaded, the Windows kernel will continue to boot the rest of the operating system using the Code Integrity capability (ci.dll) to measure code integrity for (1) the remaining kernel-mode and user-mode programs which need to be loaded for the OS to complete its boot and (2) after booting, CI also verifies the integrity of applications launched by the user (applications from Microsoft are always signed by Microsoft, and third-party applications which may be signed by the developer) by checking the RSA signature for the binary and SHA-256 hashes of the binary which are compared to the catalog files described above.

Kernel-mode code signing (KMCS), also managed by CI, prevents kernel-mode device drivers, such as the TCIP/IP network driver (tcpip.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.[36]

In addition, Windows File Protection maintains a set of protected files that are stored in a cache along with cryptographic hashes of each of those files. Once the system is initialized, Windows File Protection is loaded and will scan the protected files to ensure they have valid cryptographic hashes. Windows File Protection also registers itself to be notified should any of the protected files be modified so that it can recheck the cryptographic checksum at any point while the system is operational. Should the any of the cryptographic hash checks fail, the applicable file will be restored from the cache.

6.6.5 Windows and Application Updates

Updates to Windows are delivered as Microsoft Update Standalone Package files (.msu files) which are signed by Microsoft with two digital signatures, a RSA SHA1 signature for legacy applications and a RSA SHA-256 signature for modern applications. The 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.

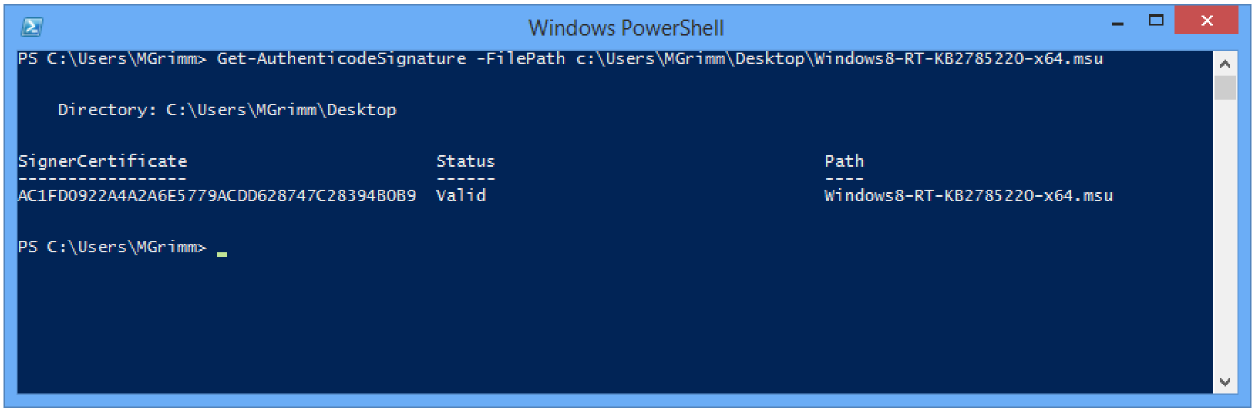
The Windows operating system will check that the certificate is valid and has not been revoked using a standard PKI CRL. Once the Trusted Installer determines that the package is valid, it will update Windows; otherwise the installation will abort and there will be an error message in the event log. 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 the Windows operating system, Windows applications, and Microsoft desktop applications are delivered through the Windows Update capability (for Windows) and Microsoft Update (for Microsoft desktop applications), which is enabled by default, or the user can go to http://catalog.update.microsoft.com to search and obtain security updates on their own volition.

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:

[36] Enforcing the Kernel Mode Code Signing policy is mandatory for the x64 version of Windows. For the x86 version of Windows, Windows will check the signatures for all kernel executable code and will halt OS if it detects an integrity error in ntoskrnl.exe, bootvid.dll, hall.dll, kdcom.dll, ci.dll, clfs.dll, ksecdd.sys, pshed.dll, or tpm.sys.



```
PS C:\Users\MGrimm> Get-AuthenticodeSignature -FilePath c:\Users\MGrimm\Desktop\Windows8-RT-KB2785220-x64.msu

    Directory: C:\Users\MGrimm\Desktop

SignerCertificate                         Status                                 Path
-----------------                         ------                                 ----
AC1FD0922A4A2A6E5779ACDD628747C28394B0B9  Valid                                  Windows8-RT-KB2785220-x64.msu

PS C:\Users\MGrimm> _
```

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.

6.6.5.1 Windows 10 and Windows Store Applications

Universal Windows Platform (UWP) apps can be downloaded from the Windows Store and their installation packages are verified using a digital signature from *Microsoft Corporation* with the Code Signing usage. These applications are contained in either *AppX packages*, or a collection of AppX packages known as an *AppX bundle*.[37] The AppX package uses the Open Packaging Conventions (OPC) standard.[38] Each package contains a directory file which lists the other files in the package, a digital signature for the package, a block map representing the application files which may be installed on the target computer, and the application files themselves. The AppX Deployment Service will verify the RSA SHA-256 digital signature for the block map and the other AppX metadata at the beginning of the AppX package (or bundle) download. This is described in more detail as part at http://blogs.msdn.com/b/windowsappdev/archive/2012/12/04/designing-a-simple-and-secure-app-package-appx.aspx.

6.6.5.2 Distributing updates

There are several distribution channels for updates to Windows and Windows applications:

- Windows Update: Windows Update is the web service for delivering Windows updates to directly to consumers.
- Windows Server Update Services (WSUS): WSUS is a server role in Windows Server which IT administrators can use to distribute application updates to users within their enterprise.
- Windows Store: The Windows Store is a web service for delivering updates to Universal Windows Platform apps which were originally installed from the Windows Store.

[37] Windows Store Applications are typically downloaded from the Windows Store for the Windows 10 operating system.

[38] OPC is also part of ISO/IEC 2900-2 and ECMA 376-2.

6.6.6 SFR Summary

- **FPT_ACF_EXT.1**: Windows provides a Discretionary Access Control policy to limit modification and reading of objects by non-authorized users.
- **FPT_ASLR_EXT.1**: Windows randomizes user-mode process address spaces and kernel-mode address space.
- **FPT_SBOP_EXT.1**: Windows binaries are compiled with stack overflow protection (compiled using the **/Gs** option for native applications).
- **FPT_SRP_EXT.1**: Windows 10 Enterprise can restrict program execution based on the file path for the executable, a digital signature for the executable, a version number for the executable, or a hash of the executable file.
- **FPT_TST_EXT.1**: 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, FPT_TUD_EXT.2**: Windows provides a means to identify the current version of the Windows software, the hardware model, and installed applications. Windows has update mechanisms to deliver updated operating system and application 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, applications, and Windows Store Applications.

6.7 TOE Access

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. 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.

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. Either action will result in an authentication dialog. 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.

6.7.1 SFR Summary

- **FTA_TAB.1**: An authorized administrator can define and modify a banner that will be displayed prior to allowing a user to logon.

6.8 Trusted Channels

Windows provides trusted network channels to communicate with supporting IT infrastructure or applications:

- Using TLS (HTTPS) for certificate enrollment; CRL checking; authentication to network resources such as web (HTTPS) and directory (LDAP-S) servers; and management via configuration sevrice providers in Windows that are local interface for processing Mobile Device Management (MDM) requests.
- Using DTLS for datagram-based services and web browsing using a DTLS version which is specified by the client application.

In order to establish a trusted channel, these communications are protected as described above in section 6.2.3.

The remote access can be performed through the following methods:

- Remote Desktop Services Overview: https://technet.microsoft.com/en-us/library/hh831447.aspx
- Connect to another computer using Remote Desktop Connection: http://windows.microsoft.com/en-us/windows/connect-using-remote-desktop-connection#connect-using-remote-desktop-connection=windows-7

Both methods use TLS (1.2) protocol for establishing the remote connection.

6.8.1 SFR Summary

- **FTP_ITC_EXT.1(TLS), FTP_ITC_EXT.1(DTLS)**: Windows provides several trusted network channels that protect data in transit from disclosure, provide data integrity, and endpoint identification that is used by TLS, HTTPS, and DTLS. TLS and HTTPS is used as part of network-based authentication and certification validation, HTTPS and DTLS are used for web-browsing and by other connection-based and datagram-based application protocols.

- **FTP_TRP.**1: Windows provide a local trusted path service as described in TOE Access and a network-based trusted channel built on the network protocols described in this section.

6.9 Security Response Process

Microsoft utilizes industry standard practices to address reported product vulnerabilities. This includes a central email address (secure@microsoft.com) to report issues (as described at https://technet.microsoft.com/en-us/security/ff852094), timely triage and root cause analysis, and responsible resolution of the report which may result in the release of a binary update. If a binary update is required, it is made available through automated channels to all customers following the process described at https://technet.microsoft.com/en-us/security/dn436305. If the sender wishes to send secure email, there is a public PGP key for S/MIME at https://technet.microsoft.com/en-us/security/dn606155.aspx. Security updates for Microsoft products – operating system, firmware, and applications – are delivered as described in section **6.6.4** and **6.6.5**.

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 compliance with the General Purpose Operating Systems Protection Profile, Version 4.1, March 9, 2016 (GP OS 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. Moreover, as demonstrated in this security target Windows runs on a wide variety of hardware ranging from tablets, convertibles, notebooks, desktop, and server computers and so it is a general purpose operating system.

The requirements in the protection profile are assumed to represent a complete set of requirements that serve to address any interdependencies. All the functional requirements in this security target have been copied from the protection profile so that all dependencies between SFRs are satisfied by the inclusion of the relevant component.

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 13 Rationale for Operations**. SFR operations left incomplete in the protection profile have been completed in this security and are identified within each SFR in section 5.1 TOE Security Functional Requirements.

Table 13 Rationale for Operations

GP OS PP Requirement	ST Requirement	Operation & Rationale
FAU_GEN.1	FAU_GEN.1	A selection and multiple assignments which are allowed by the PP.
FCS_CKM.1(1)	FCS_CKM.1(1)	Multiple selections which are allowed by the PP.
FCS_CKM.2(1)	FCS_CKM.2(1)	A selection which is allowed by the PP.
FCS_CKM.4	FCS_CKM.4	Multiple selections which are allowed by the Technical Decision #239.
FCS_COP.1(1)	FCS_COP.1(SYM)	Multiple selections which are allowed by the PP.
FCS_COP.1(2)	FCS_COP.1(HASH)	Multiple 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)	An assignment and multiple selections which are allowed by the PP.
FCS_RBG_EXT.1	FCS_RBG_EXT.1	Multiple selections which are allowed by the PP.
FCS_STO_EXT.1	FCS_STO_EXT.1	Copied from the PP without changes.
FCS_TLSC_EXT.1	FCS_TLSC_EXT.1	A selection which is allowed by the PP.
FCS_TLSC_EXT.2	FCS_TLSC_EXT.2	Copied from the PP without changes.
FCS_TLSC_EXT.3	FCS_TLSC_EXT.3	A selection which is allowed by the PP.
FCS_TLSC_EXT.4	FCS_TLSC_EXT.4	Copied from the PP without changes.
FCS_DTLS_EXT.1	FCS_DTLS_EXT.1	A selection which is allowed by the PP.
FDP_ACF_EXT.1	FDP_ACF_EXT.1	Copied from the PP without changes.
FDP_IFC_EXT.1	FDP_IFC_EXT.1	A selection which is allowed by the PP.

GP OS PP Requirement	ST Requirement	Operation & Rationale
FIA_AFL.1	FIA_AFLT.1	Multiple assignment and multiple selections which are allowed by the PP.
FIA_UAU.5	FIA_UAU.5	An assignment and a selection which are 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	A selection which is allowed by the PP.
FMT_MOF_EXT.1	FMT_MOF_EXT.1	Copied from the Technical Decision #0104 without changes.
FMT_SMF_EXT.1	FMT_SMF_EXT.1	Refinements, selections and assignments which are allowed by the Technical Decision #104.
FPT_ACF_EXT.1	FPT_ACF_EXT.1	Two assignment which is allowed by the PP.
FPT_ASLR_EXT.1	FPT_ASLR_EXT.1	An assignment which is allowed by the PP.
FPT_SBOP_EXT.1	FPT_SBOP_EXT.1	Copied from the PP without changes.
FPT_SRP_EXT.1	FPT_SRP_EXT.1	A selection which is allowed by the PP.
FPT_TST_EXT.1	FPT_TST_EXT.1	An assignment and multiple selections which are allowed by the PP.
FPT_TUD_EXT.1	FPT_TUD_EXT.1	Added a refinement to align on SFR labels.
FPT_TUD_EXT.2	FPT_TUD_EXT.2	Added a refinement to align on SFR labels.
FTA_TAB.1	FTA_TAB.1	Copied from the PP without changes.
FTP_TRP.1	FTP_TRP.1	Multiple selections which are allowed by the PP and Technical Decision #0208.
FTP_ITC_EXT.1	FTP_ITC_EXT.1(TLS)	An assignment and a selection which are allowed by the PP.
FTP_ITC_EXT.1	FTP_ITC_EXT.1(DTLS)	An assignment and a selection 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 General Purpose Operating Systems Protection Profile.

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 14**. 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 14 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
FAU_GEN.1	X								
FCS_CKM.1(1)		X							
FCS_CKM.2(1)		X							
FCS_CKM.4		X							
FCS_COP.1(SYM)		X							
FCS_COP.1(HASH)		X							
FCS_COP.1(SIGN)		X							
FCS_COP.1(HMAC)		X							
FCS_RBG_EXT.1		X							
FCS_STO_EXT.1		X							
FCS_TLSC_EXT.1		X							
FCS_TLSC_EXT.2		X							
FCS_TLSC_EXT.3		X							
FCS_TLSC_EXT.4		X							
FCS_DTLS_EXT.1		X							
FDP_ACF_EXT.1			X						
FDP_IFC_EXT.1			X						
FIA_AFL.1				X					
FIA_UAU.5				X					
FIA_X509_EXT.1				X					
FIA_X509_EXT.2				X					
FMT_MOF_EXT.1					X				
FMT_SMF_EXT.1					X				
FPT_ACF_EXT.1						X			

Requirement	Audit	Cryptographic Protection	User Data Protection	I & A	Security Management	TSF Protection	Resource Utilization	TOE Access	Trusted Path / Channel
FPT_ASLR_EXT.1						X			
FPT_SBOP_EXT.1						X			
FPT_SRP_EXT.1						X			
FPT_TST_EXT.1						X			
FPT_TUD_EXT.1						X			
FPT_TUD_EXT.2						X			
FTA_TAB.1								X	
FTP_TRP.1									X
FTP_ITC_EXT.1(TLS)									X
FTP_ITC_EXT.1(DTLS)									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
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

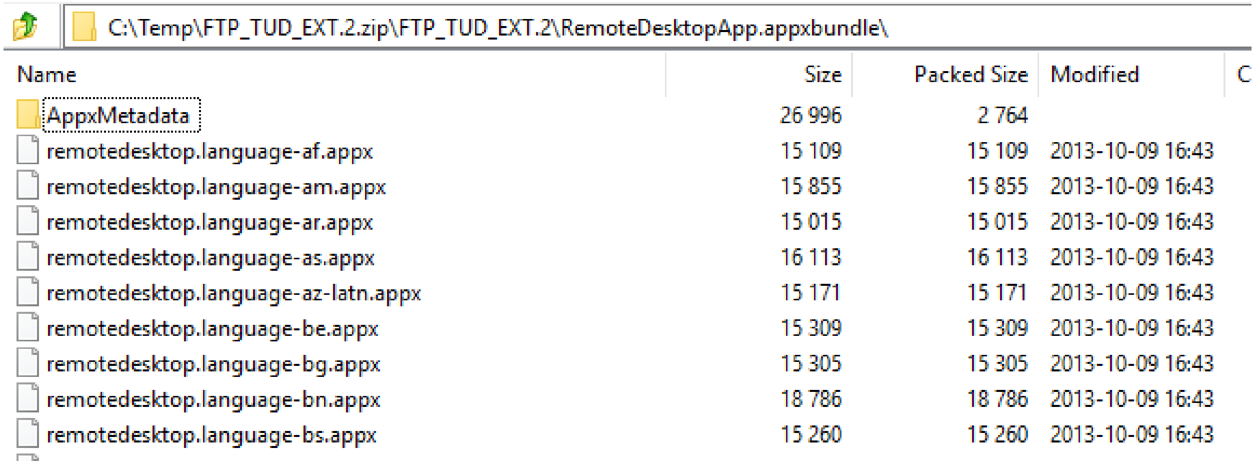
10 Appendix B: AppX Package Implementation

The AppX Deployment Service security checks are enforced at a lower level then a signature check of the entire install **Package Content** – either .appx or .appxbundle. The AppxSignature.p7x file contains three (3) signed hashes:

- AppxBlockMap.xml hash
- Package Content hash
- ZIP Central Directory hash

During install, only the AppBlockMap.xml and ZIP Central Directory hashes are validated against the contents of the install package. The installer itself does **not** check the signature of the hash of the **Package Content**. Note that the .p7x signature file occurs at the .appxbundle and .appx levels, and the scope of the security check is specific to each level.

Since the install package uses a common compressed format, it can be opened in 3rd party tools compressed file tools such as 7-zip. Opening the package supplied by the validator shows the contents of the appxbundle as a collection of .appx packages:

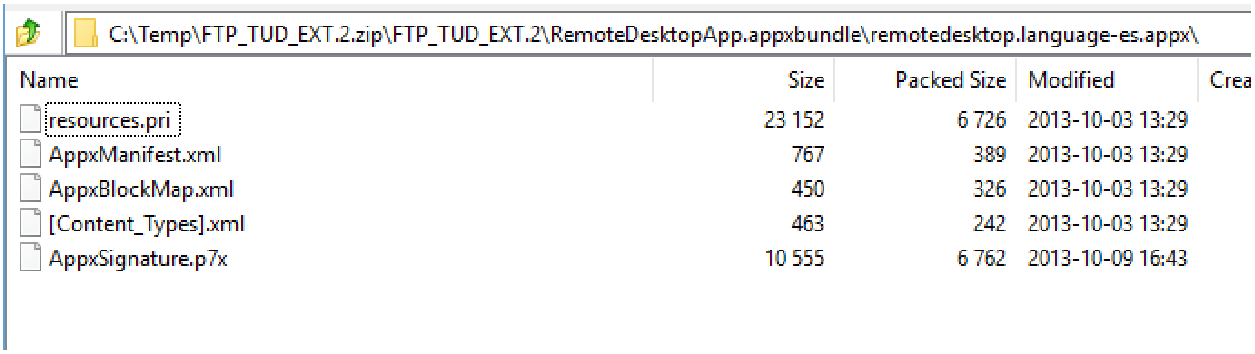


C:\Temp\FTP_TUD_EXT.2.zip\FTP_TUD_EXT.2\RemoteDesktopApp.appxbundle\

Name	Size	Packed Size	Modified	C
AppxMetadata	26 996	2 764		
remotedesktop.language-af.appx	15 109	15 109	2013-10-09 16:43	
remotedesktop.language-am.appx	15 855	15 855	2013-10-09 16:43	
remotedesktop.language-ar.appx	15 015	15 015	2013-10-09 16:43	
remotedesktop.language-as.appx	16 113	16 113	2013-10-09 16:43	
remotedesktop.language-az-latn.appx	15 171	15 171	2013-10-09 16:43	
remotedesktop.language-be.appx	15 309	15 309	2013-10-09 16:43	
remotedesktop.language-bg.appx	15 305	15 305	2013-10-09 16:43	
remotedesktop.language-bn.appx	18 786	18 786	2013-10-09 16:43	
remotedesktop.language-bs.appx	15 260	15 260	2013-10-09 16:43	

There are many .appx bundles in this bundle, most of them being language resource files.

Opening an .appx package shows:



C:\Temp\FTP_TUD_EXT.2.zip\FTP_TUD_EXT.2\RemoteDesktopApp.appxbundle\remotedesktop.language-es.appx\

Name	Size	Packed Size	Modified	Crea
resources.pri	23 152	6 726	2013-10-03 13:29	
AppxManifest.xml	767	389	2013-10-03 13:29	
AppxBlockMap.xml	450	326	2013-10-03 13:29	
[Content_Types].xml	463	242	2013-10-03 13:29	
AppxSignature.p7x	10 555	6 762	2013-10-09 16:43	

This is the Spanish language resource file. Using it as an example, the following contents are present:

Resource.pri – This is the actual content in this package. In other packages there may be more content files.

AppxManifest.xml – A file that describes the magic. Not security relevant.

AppxBlockMap.xml – Contains the SHA256 hashes of each 64k block of data in all of the files in this package. Note that if a single file spans more than 64k than it is represented by multiple block maps.

Content_Types.xml – Metadata that describes the content types of the files. Not security relevant.

AppxSignature.p7x – This file contains the digital signature that encompasses the security relevant information in the package.

Validation Processes

When processing an .appx package the installer first verifies the AppxBlockMap.xml by computing a hash value and comparing the value with the signed hash value in the associated AppxSignature.p7x file. If the hash and signature validate passes, each block-hash is compared with the hash in the AppxBlockMap.xml file.

The first time an application is downloaded and installed, each component of the .appxbundle (or .appx) that is to be installed on the target computer is validated. It is important to note that the installer will decide if certain components are needed, mainly language resource files. For example in the first screenshot there are many resource files that are included in the bundle but only one will probably be installed. Components that are NOT installed are NOT validated. Modifying a part of the .appxbundle that will NOT be installed will NOT cause an application install failure or an integrity failure.

In order to optimize the install process, the Block Map mechanism is utilized to determine which 64k blocks of any particular application need to be updated. This is done by comparing the block maps in the .appx bundles with the block maps for each application that are maintained in the \Program Files\WindowsApps folder. Note that the user does not have permissions to access \Program Files\WindowsApps but access can be granted through the Properties\Security tabs.

Once the list of modified app blocks is created, the installer can download each required block. Since each block has its own hash value maintained in the AppxBlockMap.xml file, each block can be validated as it is downloaded, optimizing the install experience.

Because of this optimized implementation, testing the integrity of an application update first requires determining which blocks have been changed and will be updated. This may be a laborious process for any particular 3rd party application as most of the blocks may not have been changed. In any case, modifying a block that will not be installed will NOT cause an installation failure.