Microsoft Windows

Common Criteria Evaluation

Microsoft Windows 11 (versions 24H2, 23H2)

Microsoft Windows Server 2025

Microsoft Azure Local (versions 24H2, 23H2)

Security Target

Document Information	
Version Number	0.02
Updated On	July 2, 2025

Version History

Version	Date	Summary of changes
0.01	November 20, 2025	Initial draft
0.02	July 2, 2025	Updates from evaluation

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.

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.

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

TABLE OF CONTENTS

LIST OF TABLES

1 Security Target Introduction

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

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

1.1 ST Reference

ST Title: Microsoft Windows 11 (versions 24H2 and 23H2), Microsoft Windows Server 2025, Microsoft Azure Local (versions 24H2, 23H2) Security Target

ST Version: version 0.02, July 2, 2025

1.2 TOE Reference

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

- Microsoft Windows 11 version 24H2 Enterprise edition
- Microsoft Windows 11 version 24H2 Pro edition
- Microsoft Windows 11 version 24H2 Education edition
- Microsoft Windows 11 version 24H2 IoT Enterprise edition
- Microsoft Windows 11 version 23H2 Enterprise edition
- Microsoft Windows Server 2025 Standard edition
- Microsoft Windows Server 2025 Datacenter edition
- Microsoft Windows Server 2025 Datacenter: Azure edition
- Microsoft Azure Local version 24H2
- Microsoft Azure Local version 23H2

TOE Versions:

- Microsoft Windows 11 build 10.0.26100.1 (also known as version 24H2)
- Microsoft Windows 11 build 10.0.22631.2428 (also known as version 23H2)
- Microsoft Windows Server 2025 build 10.0.26100.1
- Microsoft Azure Local version 10.0.26100.1 (also known as version 24H2)

The following security updates must be applied for:

- Windows 11, Windows Server and Azure Local: all critical updates as of July 1, 2025.

1.3 TOE Overview

The TOE includes the Windows 11 operating system; the Windows Server 2025 operating system; Azure Local; and those applications necessary to manage, support and configure the operating system. Windows 11 and Windows Server can be delivered preinstalled on a new computer or downloaded from the Microsoft website.

1.3.1 TOE Types

All Windows 11, Windows Server editions, plus the Windows operating systems in Azure Local products, 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 11 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.

Built for workloads ranging from the department to the enterprise to the cloud, Windows Server delivers intelligent file and printer sharing; secure connectivity based on Internet technologies, and centralized desktop policy management. It provides the necessary scalable and reliable foundation to support mission-critical solutions for databases, enterprise resource planning software, high-volume, real-time transaction processing, server consolidation, public key infrastructure, virtualization, and additional server roles.

The Azure Local product line extends Azure services and capabilities to a local IT environment spanning from the datacenter to edge locations and remote offices. Azure Local is a hyperconverged solution for scalable virtualization and storage, high-performance workloads, in modernized on-premise architecture and remote branch offices using compute and hardware-accelerated machine learning at edge location for Internet of Things (IoT) and artificial intelligence (AI) workloads.

Windows provides an interactive User Interface (UI), as well as a network interface. The TOE includes a set of computer 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 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 certificates generated and used by Windows authenticate users and machines as well as protect both user and system data in transit.
    - **TLS:** Windows implements Transport Layer Security to provide protected, authenticated, confidential, and tamper-proof networking between two peer computers.
    - **IPsec:** Windows implements IPsec to provide protected, authenticated, confidential, and tamper-proof networking between two peer computers.
    - **Wi-Fi:** Windows implements IEEE 802.11 wireless networking to provide protected, authenticated, confidential, and tamper-proof networking between Windows clients and Wi-Fi access points.
    - **Bluetooth:** Windows implements Bluetooth version 5.1 wireless networking protocols to provide protected, authenticated, confidential, and tamper-proof networking between Windows operating systems and Bluetooth peer devices.
- **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

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

attempts, the duration of lockout, and password age. Windows provides the ability to use, store, and protect X.509 certificates that are used for IPsec VPN sessions.

- **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 TLS, HTTPS, DTLS, EAP-TLS, and IPsec 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 6
- Microsoft Surface Pro 10
- Microsoft Surface Pro 11th edition (ARM)
- Microsoft Surface Laptop Go 3
- Microsoft Surface Go 4
- Microsoft Surface Laptop Studio 2
- HP EliteBook 840 14-inch G11 Notebook PCHP Elite x360 830 13-inch G11 2-in-1 Notebook PC
- Dell Precision 3490
- Dell Latitude 5550
- Dell PowerEdge R640
- Dell PowerEdge R760
- Microsoft Windows Server 2025 Hyper-V

1.4 TOE Description

The Windows TOE product series includes the Windows operating system, the Windows Server operating system, Azure Local, supporting hardware, and those applications necessary to manage, support and configure the operating system.

1.4.1 Evaluated Configurations

The Windows TOE is a series of products which includes:

- Four product variants for Windows 11 version 24H2 (build 10.0.26100.1):
  - Microsoft Windows 11 Enterprise edition
  - Microsoft Windows 11 version 24H2 Pro edition
  - Microsoft Windows 11 version 24H2 Education edition
  - Microsoft Windows 11 version 24H2 IoT Enterprise edition
- One product variant for Windows 11 version 23H2 (build 10.0.22631.2428):
  - Microsoft Windows 11 Enterprise edition
- Three variants of Windows Server 2025 (build 10.0. 26100.1)
  - Microsoft Windows Server 2025 Standard edition
  - Microsoft Windows Server 2025 Datacenter edition
  - Microsoft Windows Server 2025 Datacenter: Azure edition
- Two product variants for the Windows Server Azure product line:
  - Microsoft Azure Local version 24H2 (build 10.0. 26100.1)
  - Microsoft Azure Local version 23H2 (build 10.0.25398.469)

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:

- 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.
- **Windows OS Resume** which reloads an image of the executing operating system from a hibernation file as part of resuming from a hibernated state.
- The **Windows Kernel** which contains device drivers for the Windows NT File System, full volume encryption, the crash dump filter, and the kernel-mode cryptographic library.
- The **IPv4 / IPv6 network stack** in the kernel.
- The **IPsec** module in user-mode.
- The **IKE and AuthIP Keying Modules** service which hosts the IKE and Authenticated Internet Protocol (AuthIP) keying modules. These keying modules are used for authentication and key exchange in Internet Protocol security (IPsec).
- The **Remote Access Service** device driver in the kernel, which is used primarily for ad hoc or user-defined VPN connections; known as the "RAS IPsec VPN" or "RAS VPN".
- The **IPsec Policy Agent** service which enforces IPsec policies.
- The **Key Isolation Service** which protects secret and private keys.
- 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.
- **Local and remote administrative interfaces** for security management.
- **Windows Explorer** which can be used to manage the OS and check the integrity of Windows files and updates.
- The **Windows Trusted Installer** which installs updates to the Windows operating system.

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 (x64), AMD (x64), or Qualcomm (ARM64) 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 6
- Microsoft Surface Pro 10
- Microsoft Surface Pro 11th edition (ARM)
- Microsoft Surface Laptop Go 3
- Microsoft Surface Go 4
- Microsoft Surface Laptop Studio 2
- HP EliteBook 840 14-inch G11 Notebook PCHP Elite x360 830 13-inch G11 2-in-1 Notebook PC
- Dell Precision 3490
- Dell Latitude 5550
- Dell PowerEdge R640
- Dell PowerEdge R760
- Microsoft Windows Server 2025 Hyper-V

The Assurance Activity Report describes the relationship between the different hardware platforms and the operating systems examined during the evaluation.

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 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 11 from https://www.microsoft.com/en-us/software-download/windows11, and IT professionals can obtain a copy of Windows Server from https://admin.microsoft.com/adminportal/home#/subscriptions/vlnew. The obtained file is in .iso format. Enterprises typically obtain Windows using volume licensing programs and subscriptions such as these for Windows 11.

Windows is pre-installed on all Microsoft Surface computers.

The operating system is pre-installed on Azure Local products.

- Microsoft Windows 11 version 24H2 Enterprise edition
  - Build: 10.0.26100.1
  - ISO: 26100.1.240331-1435.ge_release_CLIENT_ENTERPRISE_OEM_x64FRE_en-us.iso
  - ISO hash: E8F1431C4E6289B3997C20EADBB2576670300BB6E1CF8948B5D7AF179010A962
- Microsoft Windows 11 version 24H2 Pro editions (x64) and Education editions
  - Build: 10.0.26100.1
  - ISO: 26100.1.240331-1435.ge_release_CLIENT_BUSINESS_VOL_x64FRE_en-us.iso
  - ISO hash: 16B20ED488999032F74B23CC51360E7A7B3C55AB6910F60103193E7D190710B3
- Microsoft Windows 11 version 24H2 IoT Enterprise edition (x64) edition
  - Build: 10.0. 26100.1
  - ISO: 26100.1.240331-1435.ge_release_CLIENT_IOTENTERPRISES_vl_x64FRE_en-us.iso
  - ISO hash: 4E43CE7CD414F0C43F4772CE2F91B0D8DD2F1A3C71833EEEA7C756053125DDA6

- Microsoft Windows 11 version 23H2 version Enterprise edition
  - Build: 10.0. 22631.2428
  - ISO: 22631.2428.231001-0608.23H2_NI_RELEASE_SVC_REFRESH_CLIENTENTERPRISE_OEM_x6FRE_en-us.iso
  - ISO hash: 5D9B86AD467BC89F488D1651A6C5AD3656A7EA923F9F914510657A24C501BB8

- Microsoft Windows Server 2025 Standard, Datacenter editions
  - Build: 10.0. 26100.1
  - ISO: 26100.1.240331-1435.ge_release_SERVER_OEMRET_x64FRE_en-us.iso

  - ISO hash: 2293897341FEBDCEA599F5412300B470B5288C6FD2B89666A7B27D283E8D3CF3

- Microsoft Windows Server 2022 2025 Datacenter: Azure edition
  - Build: 10.0. 26100.1
  - ISO: 26100.1.240331-1435.ge_release_SERVERDCAZURE_VOL_x64FRE_en-us.iso
  - ISO hash: FF42F96D1349C3035E7724090FCFBE8417074D7F9D06E1059C89BB48ED889421

- Microsoft Azure Local version 24H2
  - Build: 10.0. 26100.1
  - ISO: 26100.1.240331-1435.ge_release_SERVERAZURESTACKHCICOR_OEMRET_x64FRE_en-us.iso
  - ISO hash: B110092D8F46EEE763248BE706DCAF5B6A234E03277C388B139460DC2D641B3D
- Microsoft Azure Local version 23H2
  - Build: 10.0. 25398.469
  - ISO: 25398.469.231004-1141.zn_release_svc_refresh_SERVERAZURESTACKHCICOR_OEMRET_x64FRE_en-us.iso
  - ISO hash: 140D2A6BC53DADCCB9FB66B0D6D2EF61C9D23EA937F8CCC62788866D02997BCA

TOE Guidance Identification: The following administrator, user, and configuration guides were evaluated as part of the TOE and published at Common Criteria Certifications - Windows security | Microsoft Docs:

- *Microsoft Windows, Windows Server, and Azure Local GP OS Operational and Administrative Guidance* along with all the documents referenced therein.
  - Document SHA2-256 hash:
  - 886F346A26998E9D11F1D8F0A6B537E466C944358A51A02D9B5956BC77F0D224

The administrator and user must follow the instructions in the *Microsoft Windows, Windows Server, and Azure Local GP OS Operational and Administrative Guidance* 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 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 and mobile device management Configuration Service Providers, which is part of the Windows 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 supports 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.

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.

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

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.
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 11 Pro and Windows 11 Enterprise.
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.

[4] According to a defined metric

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 function. 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 ***Protection Profile for General Purpose Operating Systems***, ***PP-Module for WLAN Clients***, ***PP-Module for Virtual Private Network (VPN) Clients***, ***PP-Module for Bluetooth, Functional Package for Transport Layer Security (TLS)*** and Assurance Package for ***Flaw Remediation***.
- 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 ST and the Windows 11 editions (TOEs) 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)
- Protection Profile for General Purpose Operating Systems, Version 4.3, September 27, 2022 (GP OS PP)
- PP-Module for WLAN Clients, version 1.0, March 31, 2022 ("WLAN Client Module")
- PP-Module for Virtual Private Network (VPN) Clients, version 2.4, March 31, 2022 ("VPN Client Module")
- PP-Module for Bluetooth, version 1.0, April 15, 2021 ("Bluetooth Module")
- Functional Package for Transport Layer Security (TLS), version 2.0, December 19, 2022, ("TLS Module")
- Assurance Package for Flaw Remediation, version 1.0, June 28, 2024, ("ALC_FLR Module")

This ST and the Windows Server editions and Azure Local product (TOEs) 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 conformant)
- Protection Profile for General Purpose Operating Systems, Version 4.3, September 27, 2022 (GP OS PP)
- PP-Module for Virtual Private Network (VPN) Clients, version 2.4, March 31, 2022 (VPN Client Module)
- PP-Module for Bluetooth, version 1.0, April 15, 2021 ("Bluetooth Module")
- Functional Package for Transport Layer Security (TLS), version 2.0, December 19, 2022, ("TLS Module")
- Assurance Package for Flaw Remediation, version 1.0, June 28, 2024, ("ALC_FLR Module")

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

- NIAP Technical Decision 952 for the GP OS PP
- NIAP Technical Decision 930 for the GP OS PP
- NIAP Technical Decision 914 for the GP OS PP
- NIAP Technical Decision 912 for the TLS Module
- NIAP Technical Decision 911 for the TLS Module

- NIAP Technical Decision 906 for the GP OS PP
- NIAP Technical Decision 904 for the GP OS PP (not applicable)
- NIAP Technical Decision 873 for the GP OS PP
- NIAP Technical Decision 844 for the GP OS PP
- NIAP Technical Decision 839 for the GP OS PP
- NIAP Technical Decision 837 for the WLAN Client Module
- NIAP Technical Decision 821 for the GP OS PP
- NIAP Technical Decision 812 for the GP OS PP
- NIAP Technical Decision 809 for the GP OS PP
- NIAP Technical Decision 797 for the WLAN Client Module
- NIAP Technical Decision 793 for the TLS Module
- NIAP Technical Decision 789 for the GP OS PP
- NIAP Technical Decision 788 for VPN Client Module is not applicable to the GP OS PP
- NIAP Technical Decision 773 for the GP OS PP
- NIAP Technical Decision 772 for the TLS Module (archived)
- NIAP Technical Decision 753 for VPN Client Module is not applicable to the GP OS PP
- NIAP Technical Decision 731 for the TLS Module
- NIAP Technical Decision 729 for the TLS Module
- NIAP Technical Decision 725 for the VPN Client Module
- NIAP Technical Decision 713 for the GP OS PP
- NIAP Technical Decision 712 for the GP OS PP
- NIAP Technical Decision 710 for the WLAN Client Module
- NIAP Technical Decision 707 for the Bluetooth Module
- NIAP Technical Decision 703 for the WLAN Client Module
- NIAP Technical Decision 701 for the GP OS PP
- NIAP Technical Decision 696 for the GP OS PP
- NIAP Technical Decision 693 for the GP OS PP
- NIAP Technical Decision 691 for the GP OS PP
- NIAP Technical Decision 690 for VPN Client Module
- NIAP Technical Decision 685 for the Bluetooth module
- NIAP Technical Decision 675 for the GP OS PP
- NIAP Technical Decision 667 for the WLAN Client Module
- NIAP Technical Decision 662 for the VPN Client Module
- NIAP Technical Decision 650 for the Bluetooth module
- NIAP Technical Decision 647 for the VPN Client Module
- NIAP Technical Decision 645 for the Bluetooth module
- NIAP Technical Decision 640 for the Bluetooth module

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, CC:2022, Revision 1, November 2022.

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. The assumptions, threats, and policies are copied from the Protection Profile for General Purpose Operating Systems, Version 4.3, September 27, 2022 ("GP OS PP"), the PP-Module for WLAN Clients ("WLAN Client Module"), the PP-Module for Virtual Private Network (VPN) Clients ("VPN Client Module"), and the PP-Module for Bluetooth ("Bluetooth Module").

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.

Table 2 presents known or presumed threats to protected resources that are addressed by Windows based on conformance to the WLAN Client Module.

Table 2 WLAN Client Module Threats Addressed by Windows

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

T.UNAUTHORIZED ACCESS (Unauthorized Access)	A user may gain unauthorized access to the TOE data and TOE executable code. A malicious user, process, or external IT entity may masquerade as an authorized entity in order to gain unauthorized access to data or TOE resources. A malicious user, process, or external IT entity may misrepresent itself as the TOE to obtain identification and authentication data.
T.UNDETECTED_ACTIONS (Undetected Actions)	Malicious remote users or external IT entities may take actions that adversely affect the security of the TOE. These actions may remain undetected and thus their effects cannot be effectively mitigated.

The following table presents known or presumed threats to protected resources that are addressed by Windows based on conformance to the VPN Client Module.

Table 3 VPN Client Module Threats Addressed by Windows

Threat	Description
T.UNAUTHORIZED_ACCESS	This PP-Module does not include requirements that can protect against an insider threat. Authorized users are not considered hostile or malicious and are trusted to follow appropriate guidance. Only authorized personnel should have access to the system or device that contains the IPsec VPN client. Therefore, the primary threat agents are the unauthorized entities that try to gain access to the protected network (in cases where tunnel mode is used) or to plaintext data that traverses the public network (regardless of whether transport mode or tunnel mode is used). The endpoint of the network communication can be both geographically and logically distant from the TOE, and can pass through a variety of other systems. These intermediate systems may be under the control of the adversary, and offer an opportunity for communications over the network to be compromised. Plaintext communication over the network may allow critical data (such as passwords, configuration settings, and user data) to be read or manipulated directly by a malicious user or process on intermediate systems, leading to a compromise of the TOE or to the secured environmental systems that the TOE is being used to facilitate communications with. IPsec can be used to provide protection for this communication; however, there are numerous options that can be implemented for the protocol to be compliant to the protocol specification listed in the RFC. Some of these options can have negative impacts on the security of the connection. For instance, using a weak encryption algorithm (even

	one that is allowed by the RFC, such as DES) can allow an adversary to read and even manipulate the data on the encrypted channel, thus circumventing countermeasures in place to prevent such attacks. Further, if the protocol is implemented with little-used or non-standard options, it may be compliant with the protocol specification but will not be able to interact with other diverse equipment that is typically found in large enterprises. Even though the communication path is protected, there is a possibility that the IPsec peer could be tricked into thinking that a malicious third-party user or system is the TOE. For instance, a middleman could intercept a connection request to the TOE, and respond to the request as if it were the TOE. In a similar manner, the TOE could also be tricked into thinking that it is establishing communications with a legitimate IPsec peer when in fact it is not. An attacker could also mount a malicious man-in-the-middle type of attack, in which an intermediate system is compromised, and the traffic is proxied, examined, and modified by this system. This attack can even be mounted via encrypted communication channels if appropriate countermeasures are not applied. These attacks are, in part, enabled by a malicious attacker capturing network traffic (for instance, an authentication session) and "playing back" that traffic in order to fool an endpoint into thinking it was communicating with a legitimate remote entity.
T.TSF_CONFIGURATION	Configuring VPN tunnels is a complex and time-consuming process, and prone to errors if the interface for doing so is not well-specified or well-behaved. The inability or failure of an ignorant or careless administrator to configure certain aspects of the interface may also lead to the mis-specification of the desired communications policy or use of cryptography that may be desired or required for a particular site. This may result in unintended weak or plaintext communications while the user thinks that their data are being protected. Other aspects of configuring the TOE or using its security mechanisms (for example, the update process) may also result in a reduction in the trustworthiness of the VPN client.
T.USER_DATA_REUSE	Data traversing the TOE could inadvertently be sent to a different user as a consequence of a poorly designed TOE; since these data may be sensitive, this may cause a compromise that is unacceptable. The specific threat that must be addressed concerns user data that is retained by the TOE in the course of processing network traffic that could be inadvertently re-used in sending network traffic to a user other than that intended by the sender of the original network traffic.

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

Table 4 Bluetooth Module Threats Addressed by Windows

Threat	Description
N.A.	This PP-Module defines no additional threats beyond those defined in GP OS PP.

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

Table 5 Organizational Security Policies

Security Policy	Description
N.A.	There are no Organizational Security Policies for the protection profile or the protection profile modules.

3.3 Secure Usage Assumptions

Table 6 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 6 GP OS PP 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.

Table 7 WLAN Client Module Secure Usage Assumptions

Assumption	Description
A.NO_TOE_BYPASS	Information cannot flow between the wireless client and the internal wired network without passing through the TOE.
A.TRUSTED_ADMIN	TOE Administrators are trusted to follow and apply all administrator guidance in a trusted manner.

Table 8 VPN Client Module Secure Usage Assumptions

Assumption	Description
A.NO_TOE_BYPASS	Information cannot flow onto the network to which the VPN client's host is connected without passing through the TOE.
A.PHYSICAL	Physical security, commensurate with the value of the TOE and the data it contains, is assumed to be provided by the environment.
A.TRUSTED_CONFIG	Personnel configuring the TOE and its operational environment will follow the applicable security configuration guidance.

Table 9 Bluetooth Module Secure Usage Assumptions

Assumption	Description
N/A	The Bluetooth Module does not define any additional assumptions.

4 Security Objectives

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

Table 10 GP OS PP Security Objectives for the TOE

Security Objective	Source
O.ACCOUNTABILITY	Conformant OSes 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 OSes ensure the integrity of their update packages. OSes are seldom if ever shipped without errors, and the ability to deploy patches and updates with integrity is critical to enterprise network security. Conformant OSes 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 OSes 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 OSes provide mechanisms to create trusted channels for CSP and sensitive data. Both CSP and sensitive data should not be exposed outside of the platform.

Table 11 and **Table 12** describe the security objectives for Windows which are needed to comply with the PP-Module for WLAN Clients and the PP-Module for Virtual Private Network (VPN) Clients respectively.

Table 11 WLAN Client Module Security Objectives for the TOE

Security Objective	Source
O.AUTH_COMM (Authorized Communication)	The TOE will provide a means to ensure that it is communicating with an authorized access point and not some other entity pretending to be an authorized access point, and will provide assurance to the access point of its identity.
O.CRYPTOGRAPHIC_FUNCTIONS (Cryptographic Functions)	The TOE will provide or use cryptographic functions (i.e., encryption/decryption and digital signature operations) to maintain the confidentiality and allow for detection of modification of data that are transmitted outside the TOE and its host environment.
O.TSF_SELF_TEST (TSF Self Test)	The TOE will provide the capability to test some subset of its security functionality to ensure it is operating properly.
O.SYSTEM_MONITORING (System Monitoring)	The TOE will provide the capability to generate audit data.
O.TOE_ADMINISTRATION (TOE Administration)	The TOE will provide mechanisms to allow administrators to be able to configure the TOE.
O.WIRELESS_ACCESS_POINT_CONNECTION Wireless Access Point Connection	The TOE will provide the capability to restrict the wireless access points to which it will connect.

Table 12 VPN Client Module Security Objectives for the TOE

Security Objective	Source
O.AUTHENTICATION	To address the issues associated with unauthorized disclosure of information in transit, a compliant TOE's authentication ability (IPsec) will allow the TSF to establish VPN connectivity with a remote VPN gateway or peer and ensure that any such connection attempt is both authenticated and authorized.
O.CRYPTOGRAPHIC_FUNCTIONS	To address the issues associated with unauthorized disclosure of information in transit, a compliant TOE will implement cryptographic capabilities. These capabilities are intended to maintain confidentiality and allow for detection and modification of data that is transmitted outside of the TOE.
O.KNOWN_STATE	The TOE will provide sufficient measures to ensure it is operating in a known state. At minimum this

	includes management functionality to allow the security functionality to be configured and self-test functionality that allows it to assert its own integrity. It may also include auditing functionality that can be used to determine the operational behavior of the TOE.
O.NONDISCLOSURE	To address the issues associated with unauthorized disclosure of information at rest, a compliant TOE will ensure that non-persistent data is purged when no longer needed. The TSF may also implement measures to protect against the disclosure of stored cryptographic keys and data through implementation of protected storage and secure erasure methods. The TOE may optionally also enforce split-tunneling prevention to ensure that data in transit cannot be disclosed inadvertently outside of the IPsec tunnel and prohibit transmission of packets through a connection until certain conditions are met.

The PP-Module for Bluetooth does not define any additional security objectives, instead it builds on the security objectives from the Protection Profile for General Purpose Operating Systems.

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

Table 13 GP OS PP 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.

Table 14 WLAN Client Module Security Objectives for the Operational Environment

Environment Objective	Description
OE.NO_TOE_BYPASS	Information cannot flow between external and internal networks located in different enclaves without passing through the TOE.
OE.TRUSTED_ADMIN	TOE Administrators are trusted to follow and apply all administrator guidance in a trusted manner.

Table 15 VPN Client Module Security Objectives for the Operational Environment

Environment Objective	Description
OE.NO_TOE_BYPASS	Information cannot flow onto the network to which the VPN client's host is connected without passing through the TOE.
OE.PHYSICAL	Physical security, commensurate with the value of the TOE and the data it contains, is assumed to be provided by the environment.
OE.TRUSTED_CONFIG	Personnel configuring the TOE and its operational environment will follow the applicable security configuration guidance.

The PP-Module for Bluetooth does not define any additional security objectives for the operational environment, instead it builds on the security objectives for the operational environment from the Protection Profile for General Purpose Operating Systems.

5 Security Requirements

The section defines the Security Functional Requirements (SFRs) and Security Assurance Requirements (SARs) for the TOE. The requirements in this section have been drawn from the Protection Profile for General Purpose Operating Systems, Version 4.3, September 27, 2022 ("GP OS PP"); the PP-Module for WLAN Clients, version 1.0, March 31, 2022 ("WLAN Client Module"); the PP-Module for Virtual Private Network (VPN) Clients, version 2.4, March 31, 2022, ("VPN Client Module"); the PP-Module for Bluetooth, version 1.0, April 15, 2021, ("Bluetooth Module"); the Functional Package for Transport Layer Security (TLS), version 2.0, December 19, 2022, ("TLS Module"); the Assurance Package for Flaw Remediation, version 1.0, June 28, 2024, ("ALC_FLR Module"); 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 16 TOE Security Functional Requirements for GP OS PP

Requirement Class	Requirement Component
Security Audit (FAU)	Audit Data Generation (FAU_GEN.1)
Cryptographic Support (FCS)	Cryptographic Key Generation for (FCS_CKM.1)
	Cryptographic Key Establishment (FCS_CKM.2)
	Cryptographic Key Destruction (FCS_CKM_EXT.4)
	Cryptographic Operation for Data Encryption/Decryption (FCS_COP.1/ENCRYPT)
	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/KEYHMAC)
	Random Bit Generation (FCS_RBG_EXT.1)
	Storage of Sensitive Data (FCS_STO_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 for OS (FMT_SMF_EXT.1)
Protection of the TSF (FPT)	Access Controls (FPT_ACF_EXT.1)
	Address Space Layout Randomization (FPT_ASLR_EXT.1)
	Limitation of Bluetooth Profile Support (FPT_BLT_EXT.1)
	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)

Table 17 TOE Security Functional Requirements for WLAN Client Module

Requirement Class	Requirement Component
Security Audit (FAU)	Audit Data Generation for Wireless LAN (FAU_GEN.1 (WLAN))
Cryptographic Support (FCS)	Cryptographic Key Generation for Symmetric Keys for WPA2/WPA3Connections (FCS_CKM.1(WPA))
	Cryptographic Key Distribution for Symmetric Keys for WPA2/WPA3Connections (FCS_CKM.2(WLAN))

	Extensible Authentication Protocol-Transport Layer Security (FCS_TLSC_EXT.1(WLAN))
	TLS Client Support for Supported Groups Extension (EAP-TLS for WLAN) (FCS_TLSC_EXT.2(WLAN))
	Supported WPA Versions (FCS_WPA_EXT.1)
Identification & Authentication (FIA)	Port Access Entity Authentication (FIA_PAE_EXT.1)
	X.509 Certificate Validation (FIA_X509_EXT.1(WLAN))
	X.509 Certificate Authentication EAP-TLS for WLAN (FIA_X509_EXT.2(WLAN))
	Certificate Storage and Management (FIA_X509_EXT.6)
Security Management (FMT)	Specification of Management Functions for Wi-Fi (FMT_SMF.1(WLAN))
Protection of the TSF (FPT)	TSF Cryptographic Functionality Testing (FPT_TST_EXT.3 (WLAN))
TOE Access (FTA)	Wireless Network Access (FTA_WSE_EXT.1)
Trusted Path/Channels (FTP)	Trusted Channel Communication (FTP_ITC_EXT.1 (WLAN))

Table 18 TOE Security Functional Requirements for VPN Client Module

Requirement Class	Requirement Component
Security Audit (FAU)	Audit Data Generation (FAU_GEN.1(VPN))
	Selective Audit (FAU_SEL.1)
Cryptographic Support (FCS)	Cryptographic Key Generation (FCS_CKM.1 (VPN))
	Cryptographic Key Storage (FCS_CKM_EXT.2)
	EAP-TLS (FCS_EAP_EXT.1)
	IPsec (FCS_IPSEC_EXT.1)
User Data Protection (FDP)	Split Tunnel Prevention (FDP_VPN_EXT.1)
	Full Residual Information Protection (FDP_RIP.2)
Identification & Authentication (FIA)	Pre-Shared Key Composition (FIA_PSK_EXT.1)
	Generated Pre-Shared Keys (FIA_PSK_EXT.2)
	X.509 Certificate Use and Management (FIA_X509_EXT.3)
Security Management (FMT)	Specification of Management Functions for VPN (FMT_SMF.1(VPN))
Protection of the TSF (FPT)	Self-Test for IPsec (FPT_TST_EXT.1 (VPN))
Trusted Path/Channels (FTP)	Inter-TSF Trusted Channel (FTP_ITC.1(VPN))

Table 19 TOE Security Functional Requirements for PP-Module for Bluetooth

Requirement Class	Requirement Component
Security Audit (FAU)	Audit Data Generation (FAU_GEN.1(BT))
Cryptographic Support (FCS)	Bluetooth Key Generation (FCS_CKM_EXT.8)
Identification & Authentication (FIA)	Bluetooth User Authorization (FIA_BLT_EXT.1)
	Bluetooth Mutual Authentication (FIA_BLT_EXT.2)

	Rejection of Duplicate Bluetooth Connections (FIA_BLT_EXT.3)
	Secure Simple Pairing (FIA_BLT_EXT.4)
	Trusted Bluetooth Device User Authorization (FIA_BLT_EXT.6)
	Untrusted Bluetooth Device User Authorization (FIA_BLT_EXT.7)
Security Management (FMT)	Management of Security Functions Behavior for Bluetooth (FMT_MOF_EXT.1(BT))
	Specification of Management Functions for VPN (FMT_SMF_EXT.1(BT))
Trusted Path/Channels (FTP)	Bluetooth Encryption (FTP_BLT_EXT.1)
	Persistence of Bluetooth Encryption (FTP_BLT_EXT.2)
	Bluetooth Encryption Parameters (BR/EDR) (FTP_BLT_EXT.3(BR))
	Bluetooth Encryption Parameters (LE) (FTP_BLT_EXT.3(LE))

Table 20 TOE Security Functional Requirements for Functional Package for Transport Layer Security (TLS)

Requirement Class	Requirement Component
Cryptographic Support (FCS)	TLS Protocol (FCS_TLS_EXT.1)
	TLS Client Protocol (FCS_TLSC_EXT.1)
	TLS Client Support for Mutual Authentication (FCS_TLSC_EXT.2)
	TLS Client Support Downgrade Protection (FCS_TLSC_EXT.3)
	TLS Client Support for Renegotiation (FCS_TLSC_EXT.4)
	TLS Client Support for Session Resumption (FCS_TLSC_EXT.5)
	TLS Client 1.3 Resumption Refinements (FCS_TLSC_EXT.6)
	TLS Server Protocol (FCS_TLSS_EXT.1)
	TLS Server Support for Mutual Authentication (FCS_TLSS_EXT.2)
	TLS Server Support Downgrade Protection (FCS_TLSS_EXT.3)
	TLS Server Support for Session Resumption (FCS_TLSS_EXT.5)
	TLS Server TLS 1.3 Resumption Refinements (FCS_TLSS_EXT.6)
	DTLS Client Protocol (FCS_DTLSC_EXT.1)
	DTLS Client Support for Mutual Authentication (FCS_DTLSC_EXT.2)
	DTLS Client Downgrade Protection (FCS_DTLSC_EXT.3)
	[D]TLS Client Support for Renegotiation (FCS_DTLSC_EXT.4)
	DTLS Client Support for Session Resumption (FCS_DTLSC_EXT.5)
	DTLS Server Protocol (FCS_DTLSS_EXT.1)
	DTLS Server Support for Mutual Authentication (FCS_DTLSS_EXT.2)
	DTLS Server Downgrade Protection (FCS_DTLSS_EXT.3)
	DTLS Server Support for Session Resumption (FCS_DTLSS_EXT.5)

5.1.1 Security Audit (FAU)

5.1.1.1 Security Audit for GP OS PP

5.1.1.1.1 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, enable, and credential change)***
- ***Audit and log data access events (Success/Failure)***
- ***Cryptographic verification of software (Success/Failure)***
- ***Attempted application invocation with arguments (Success/Failure e.g. due to software restriction policy)***
- ***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, audit events from the WLAN Client module listed in*** *Error! Reference source not found.****].***

]

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**].

5.1.1.2 Security Audit for WLAN Client Module

5.1.1.2.1 Audit Data Generation for Wireless LAN (FAU_GEN.1(WLAN))

Application Note: FAU_GEN.1(WLAN) corresponds to FAU_GEN.1/WLAN in the WLAN Client module.

FAU_GEN.1.1(WLAN) The TSF shall [***implement functionality***] 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) all auditable events for mandatory SFRs specified in Table 2**0** and selected SFRs in Table **20** ~~5~~.

FAU_GEN.1.2(WLAN) The [***TSF***] shall record within each audit record at least the following information:
a) Date and time of the event, type of event, subject identity, (if relevant) the 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-Module/ST Additional Audit Record Contents as specified in Table 2**0** and Table 20 ~~5~~.

Table 21 WLAN Client Module Audit Events

Requirement	Auditable Events	Additional Audit Record Contents
FAU_GEN.1/WLAN	No events specified.	N/A
FCS_CKM.1/WPA	No events specified	N/A
FCS_CKM.2/WLAN	No events specified	N/A
FCS_TLSC_EXT.1/WLAN	Failure to establish an EAP-TLS session. Establishment/termination of an EAP-TLS session.	Reason for failure. Non-TOE endpoint of connection.
FCS_TLSC_EXT.2/WLAN	No events specified	N/A
FCS_WPA_EXT.1	No events specified	N/A
FIA_PAE_EXT.1	No events specified	N/A
FIA_X509_EXT.1/WLAN	Failure to validate X.509v3 certificate	Reason for failure of validation.
FIA_X509_EXT.2/WLAN	None.	
FIA_X509_EXT.6	Attempts to load certificates. Attempts to revoke certificates.	None.
FMT_SMF_EXT.1/WLAN	No events specified	N/A
FPT_TST_EXT.3/WLAN	Execution of this set of TSF self-tests. [***Detected integrity violation***].	[***The TSF binary file that caused the integrity violation***].
FTA_WSE_EXT.1	All attempts to connect to access points.	For each access point record the [***Complete SSID and MAC, Certificate Check Message and the last [: integer greater than or equal to 2] octets***] of the MAC Address

		Success and failures (including reason for failure).
FTP_ITC_EXT.1/WLAN	All attempts to establish a trusted channel.	Identification of the non-TOE endpoint of the channel.

5.1.1.3 Security Audit for VPN Client Module

5.1.1.3.1 Audit Data Generation (FAU_GEN.1(VPN))

Application Note: FAU_GEN.1(VPN) corresponds to FAU_GEN.1 in the VPN Client module.

FAU_GEN.1.1(VPN) The TSF and [***no other component***] 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) All administrative actions;
d) Specifically defined auditable events listed in Table **21** ~~C-1~~.

FAU_GEN.1.2(VPN) The TSF and [***no other component***] shall record within each audit record at least the following information:
c) Date and time of the event, type of event, subject identity, and the outcome (success or failure) of the event; and
d) For each audit event type, based on the auditable event definitions of the functional components included in the PP-Module/ST, information specified in column three of Table **21** ~~C-1~~.

Table 22 VPN Client Module Audit Events

Requirement	Auditable Events	Additional Audit Record Contents
FAU_GEN.1(VPN)	No events specified	N/A
FAU_SEL.1	All modifications to the audit configuration that occur while the audit collection functions are operating.	None.
FCS_CKM.1(VPN)	No events specified.	N/A
FCS_IPSEC_EXT.1	Decisions to DISCARD or BYPASS network packets processed by the TOE.	Presumed identity of source subject. The entry in the SPD that applied to the decision.
FCS_IPSEC_EXT.1	Failure to establish an IPsec SA.	Identity of destination subject. Reason for failure.
FCS_IPSEC_EXT.1	Establishment/Termination of an IPsec SA.	Identity of destination subject. Transport layer protocol, if applicable. Source subject service identifier, if applicable.

		Non-TOE endpoint of connection (IP address) for both successes and failures.
FDP_RIP.2	No events specified.	N/A
FMT_SMF.1(VPN)	Success or failure of management function.	No additional information.
FPT_TST_EXT.1(VPN)	No events specified.	N/A

5.1.1.3.2 Selective Audit (FAU_SEL.1)

FAU_SEL.1.1 The [***TSF***] shall be able to select the set of events to be audited from the set of all auditable events based on the following attributes:
event type, success of auditable security events, failure of auditable security events, [**subject or user identity**].

5.1.1.4 Security Audit for Bluetooth Module

5.1.1.4.1 Audit Data Generation (FAU_GEN.1(BT))[5]

Application Note: FAU_GEN.1(BT) corresponds to FAU_GEN.1/BT in the Bluetooth Module.

FAU_GEN.1.1(BT) The TSF 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 selected level of audit
c. Specifically defined auditable events in the Auditable Events table.

Table **2**2 Auditable Events

Table 23 Bluetooth Module Audit Events

Requirement	Auditable Events	Additional Audit Record Contents
FCS_CKM_EXT.8	None.	
FIA_BLT_EXT.1	Failed user authorization of Bluetooth device.	User authorization decision (e.g., user rejected connection, incorrect pin entry). [***complete***] BD_ADDR and [***name of device***]. Bluetooth profile. Identity of local service with [***service ID***]. Bluetooth address and name of device. Bluetooth profile.
	Failed user authorization for local Bluetooth Service.	

[5] This PP-module requirement was replaced as part of NIAP Technical Decision 645 and 707.

		Identity of local service with [***service ID***].
FIA_BLT_EXT.2	Initiation of Bluetooth connection.	[***complete***] BD_ADDR and [***name of device***].
	Failure of Bluetooth connection.	Reason for failure.
~~**FIA_BLT_EXT.3**~~	~~Duplicate connection attempt.~~	~~[***complete***] BD_ADDR and [***name of device***].~~
FIA_BLT_EXT.4	None.	
~~**FIA_BLT_EXT.5**~~	~~None.~~	
FIA_BLT_EXT.6	None.	
FIA_BLT_EXT.7	None.	
FTP_BLT_EXT.1	None.	
FTP_BLT_EXT.2	None.	
FTP_BLT_EXT.3(BR)	None.	
FTP_BLT_EXT.3(LE)	None.	

FAU_GEN.1.2(BT) The TSF shall record within each audit record at least the following information:

a. Date and time of the event
b. Type of event
c. Subject identity
d. The outcome (success or failure) of the event
e. For each audit event type, based on the auditable event definitions of the functional components included in the PP/ST
f. Additional information in the Auditable Events table.

5.1.1.5 Security Audit for TLS Functional Package

5.1.1.5.1 Audit Data Generation for TLS Functional Package[6]

Table 24 TLS Module Audit Events

Requirement[7]	Auditable Events	Additional Audit Record Contents
FCS_TLS_EXT.1	No events specified	N/A
FCS_DTLSC_EXT.1	[***Establishment/termination of a DTLS session***]	[***Non-TOE endpoint of connection.***]
	[***Failure to establish a DTLS session***]	[***Reason for failure.***]
	[***Failure to verify presented identifier***]	[***Presented identifier and reference identifier.***]
FCS_DTLSC_EXT.2	No events specified	N/A
FCS_DTLSC_EXT.3	No events specified	N/A

[6] This Functional Package requirement was replaced as part of NIAP Technical Decision 912.

FCS_DTLSC_EXT.4	No events specified	N/A
FCS_DTLSC_EXT.5	No events specified	N/A
FCS_DTLSC_EXT.6	No events specified	N/A
FCS_DTLSS_EXT.1	[***Failure to establish a DTLS session***]	[***Reason for failure.***]
FCS_DTLSS_EXT.2	No events specified	N/A
FCS_DTLSS_EXT.3	No events specified	N/A
FCS_DTLSS_EXT.5	No events specified	N/A
FCS_DTLSS_EXT.6	No events specified	N/A
FCS_TLSC_EXT.1	[***Failure to establish a TLS session***]	[***Reason for failure.***]
	[***Failure to verify presented identifier***]	[***Presented identifier and reference identifier.***]
	[***Establishment/termination of a TLS session***]	[***Non-TOE endpoint of connection.***]
FCS_TLSC_EXT.2	No events specified	N/A
FCS_TLSC_EXT.3	No events specified	N/A
FCS_TLSC_EXT.4	No events specified	N/A
FCS_TLSC_EXT.5	No events specified	N/A
FCS_TLSC_EXT.6	No events specified	N/A
FCS_TLSS_EXT.1	[***Failure to establish a TLS session***]	[***Reason for failure.***]
FCS_TLSS_EXT.2	No events specified	N/A
FCS_TLSS_EXT.3	No events specified	N/A
FCS_TLSS_EXT.5	No events specified	N/A
FCS_TLSS_EXT.6	No events specified	N/A

5.1.2 Cryptographic Support (FCS)

5.1.2.1 Cryptographic Support for GP OS PP

5.1.2.1.1 Cryptographic Key Generation (FCS_CKM.1)

FCS_CKM.1.1[8] The OS shall generate asymmetric cryptographic keys in accordance with a specified cryptographic key generation algorithm

[

- ***RSA schemes using cryptographic key sizes of 3072-bit or greater that meet the following: FIPS PUB 186-5, "Digital Signature Standard (DSS)", Appendix A.1***
- ***ECC schemes using "NIST curves" P-384 and [-521] that meet the following: FIPS PUB 186-5, "Digital Signature Standard (DSS), Appendix A.2***

[8] This protection profile requirement was modified as part of NIAP Technical Decision 712, 873 and 952.

- ***FFC schemes using [cryptographic key sizes of 3072-bit or greater that meet the following: FIPS PUB 186-5, "Digital Signature Standard (DSS)", Appendix B.1, safe primes that meet the following: 'NIST Special Publication 800-56A Revision 3, "Recommendation for Pair-Wise Key Establishment Schemes]***
- ***FFC Schemes using Diffie-Hellman group 14 that meet the following: RFC 3526***

].

5.1.2.1.2 Cryptographic Key Establishment (FCS_CKM.2)

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: RSAESPKCS1-v1_5 as specified in Section 7.2 of RFC 8017, "Public-Key Cryptography Standards (PKCS) #1: RSA Cryptography Specifications Version 2.2,***
- ***Elliptic curve-based key establishment schemes that meets the following: NIST Special Publication 800-56A Revision 3, "Recommendation for Pair-Wise Key Establishment Schemes Using Discrete Logarithm Cryptography"***
- ***Finite field-based key establishment schemes that meets the following: NIST Special Publication 800-56A Revision 3, "Recommendation for Pair-Wise Key Establishment Schemes Using Discrete Logarithm Cryptography",***
- ***Key establishment scheme using Diffie-Hellman group 14 that meets the following: RFC 3526***

].

5.1.2.1.3 Cryptographic Key Destruction (FCS_CKM_EXT.4)

FCS_CKM_EXT.4.1 The OS shall destroy cryptographic keys and key material in accordance with a specified cryptographic key destruction method [

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

].

FCS_CKM_EXT.4.2 The OS shall destroy all keys and key material when no longer needed.

5.1.2.1.4 Cryptographic Operation for Encryption / Decryption (FCS_COP.1/ENCRYPT)

FCS_COP.1.1/ENCRYPT[9] 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)***
- ***AES-CTR (as defined in NIST SP 800-38A)***

and [

- ***AES Key Wrap (KW) (as defined in NIST SP 800-38F)***

[9] This protection profile requirement was modified as part of NIAP Technical Decision 712.

- ***AES-CCMP-256 (as defined in NIST SP800-38C and IEEE 802.11ac2013),***
- ***AES-GCMP-256 (as defined in NIST SP800-38D and IEEE 802.11ac2013),***
- ***AES-CCM (as defined in NIST SP 800-38C),***
- ***AES-GCM (as defined in NIST SP 800-38D),***
- ***AES-CCMP (as defined in FIPS PUB 197, NIST SP 800-38C and IEEE 802.11-2012)***

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

5.1.2.1.5 Cryptographic Operation for Hashing (FCS_COP.1/HASH)

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

5.1.2.1.6 Cryptographic Operation for Signing (FCS_COP.1/SIGN)

FCS_COP.1.1/SIGN[10] 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 (for secure boot only) or greater, 3072-bit or greater] that meet the following: FIPS PUB 186-5, "Digital Signature Standard (DSS)", Section 4,***
- ***ECDSA schemes using "NIST curves" P-384 and [P-521] that meet the following: SP 800-186 Section 3***

].

5.1.2.1.7 Cryptographic Operation for Keyed Hash Algorithms (FCS_COP.1/KEYHMAC)

FCS_COP.1.1/KEYHMAC The OS shall perform keyed-hash message authentication services in accordance with a specified cryptographic algorithm [***SHA-256, SHA-384, SHA-512***] with key sizes [**256 bits**] and message digest sizes [***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*.

5.1.2.1.8 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.

[10] This protection profile requirement was modified as part of NIAP Technical Decision 809 and 873.

5.1.2.1.9 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.

5.1.2.2 Cryptographic Support for WLAN Client Module

5.1.2.2.1 Cryptographic Key Generation for Symmetric Keys for WPA2/WPA3Connections (FCS_CKM.1(WPA))

Application Note: FCS_CKM.1(WPA) corresponds to FCS_CKM.1/WPA in the WLAN Client module.

FCS_CKM.1.1(WPA) The TSF shall generate symmetric cryptographic keys in accordance with a specified cryptographic key generation algorithm PRF-384 and [***PRF-704***] (as defined in IEEE 802.11-2012) and specified key sizes 256 bits and **[*128 bits]*** using a Random Bit Generator as specified in FCS_RBG_EXT.1.

5.1.2.2.2 Cryptographic Key Distribution for Group Temporal Key (GTK) (FCS_CKM.2(WLAN))

Application Note: FCS_CKM.2(WLAN) corresponds to FCS_CKM.2/WLAN in the WLAN Client module.

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

5.1.2.2.3 Extensible Authentication Protocol-Transport Layer Security (FCS_TLSC_EXT.1(WLAN))

Application Note: FCS_TLCS_EXT.1(WLAN) corresponds to FCS_TLCS_EXT.1/WLAN in the WLAN Client module.

FCS_TLSC_EXT.1.1(WLAN) The TSF shall implement TLS 1.2 (RFC 5246) and **[*TLS 1.1 (RFC 4346)*]** in support of the EAP-TLS protocol as specified in RFC 5216 supporting the following ciphersuites: [

- ***TLS_RSA_WITH_AES_128_CBC_SHA as defined in RFC 5246,***
- ***TLS_RSA_WITH_AES_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 5430***
- ***TLS_ECDHE_ECDSA_WITH_AES_256_GCM_SHA384 as defined in RFC 5289***

].

FCS_TLSC_EXT.1.2(WLAN) The TSF shall generate random values used in the EAP-TLS exchange using the RBG specified in FCS_RBG_EXT.1

FCS_TLSC_EXT.1.3(WLAN) The TSF shall use X509 v3 certificates as specified in FIA_X509_EXT.1**(WLAN)**.

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

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

5.1.2.2.4 TLS Client Support for Supported Groups Extension (EAP-TLS for WLAN) (FCS_TLSC_EXT.2(WLAN))

Application Note: FCS_TLCS_EXT.2(WLAN) corresponds to FCS_TLCS_EXT.2/WLAN in the WLAN Client module.

FCS_TLSC_EXT.2.1(WLAN) The TSF shall present the Supported Groups extension in the Client Hello with the following NIST curves: [***secp256r1, secp384r1, secp521r1***].

5.1.2.2.5 Supported WPA Versions (FCS_WPA_EXT.1)

FCS_WPA_EXT.1.1 The TSF shall support WPA3 and [***WPA2***] security type.

5.1.2.3 Cryptographic Support for VPN Client Module

5.1.2.3.1 Cryptographic Key Generation (FCS_CKM.1 (VPN))

Application Note: FCS_CKM.1(VPN) corresponds to FCS_CKM.1/VPN in the VPN Client Module.

FCS_CKM.1.1(VPN) The TSF shall [***implement functionality***] to **generate** asymmetric cryptographic keys used for IKE peer authentication in accordance with: [

- ***FIPS PUB 186-4, "Digital Signature Standard (DSS)", Appendix B.3 for RSA schemes;***
- ***FIPS PUB 186-4, "Digital Signature Standard (DSS)", Appendix B.4 for ECDSA schemes and implementing "NIST curves", P-256, P-384, and [no other curves]***]

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

5.1.2.3.2 Cryptographic Key Storage (FCS_CKM_EXT.2)

FCS_CKM_EXT.2.1 The [***OS***] shall store persistent secrets and private keys when not in use in OS-provided key storage.

5.1.2.3.3 EAP-TLS (FCS_EAP_EXT.1)

FCS_EAP_EXT.1.1 The TSF shall implement [***EAP-TLS protocol as specified in RFC 5216***] as updated by RFC 8996 with TLS implemented using mutual authentication in accordance with the TLS functional package.

FCS_ EAP_EXT.1.2 The TSF shall generate random values used in the [***EAP-TLS***] exchange using the RBG specified in FCS_RBG_EXT.1.

FCS_ EAP_EXT.1.3 The TSF shall support peer authentication using certificates and [***no other authentication***] as updated by RFC 8996 with TLS implemented using mutual authentication in accordance with the TLS functional package.

FCS_ EAP _EXT.1.4 The TSF shall use the MSK from the [***EAP-TLS***] response as the IKEv2 shared secret in the authentication payload.

5.1.2.3.4 IPsec (FCS_IPSEC_EXT.1)

FCS_IPSEC_EXT.1.1 The TSF shall implement the IPsec architecture as specified in RFC 4301.

FCS_IPSEC_EXT.1.2 The TSF shall implement [***tunnel mode, transport mode***].

FCS_IPSEC_EXT.1.3 The TSF shall have a nominal, final entry in the SPD that matches anything that is otherwise unmatched, and discards it.

FCS_IPSEC_EXT.1.4 The TSF shall implement the IPsec protocol ESP as defined by RFC 4303 using the cryptographic algorithms AES-GCM-128, AESGCM-256 as specified in RFC 4106, [***AES-CBC-128, AES-CBC-256 (both specified by RFC 3602) together with a Secure Hash Algorithm (SHA)-based HMAC***].

FCS_IPSEC_EXT.1.5 The TSF shall implement the protocol: [

- ***IKEv1, using Main Mode for Phase I exchanges, as defined in RFCs 2407, 2408, 2409, RFC 4109, [RFC 4304 for extended sequence numbers], [RFC 4868 for hash functions], and [no support for XAUTH];***
- ***IKEv2 as defined in RFCs 7296 (with mandatory support for NAT traversal as specified in section 2.23), RFC 8784, RFC 8247, and [RFC 4868 for hash functions]***].

FCS_IPSEC_EXT.1.6 The TSF shall ensure the encrypted payload in the [***IKEv1, IKEv2***] protocol uses the cryptographic algorithms AES-CBC-128, AES-CBC-256 as specified in RFC 6379 and [***no other algorithm***].

FCS_IPSEC_EXT.1.7 The TSF shall ensure that [

- ***IKEv2 SA lifetimes can be configured by [VPN Gateway] based on [number of packets/number of bytes, length of time],***
- ***IKEv1 SA lifetimes can be configured by an [an Administrator, VPN Gateway] based on [number of packets/number of bytes, length of time]***

]. If length of time is used, it must include at least one option that is 24 hours or less for Phase 1 SAs and 8 hours or less for Phase 2 SAs.

FCS_IPSEC_EXT.1.8 The TSF shall ensure that all IKE protocols implement DH groups

- 19 (256-bit Random ECP), 20 (384-bit Random ECP) according to

RFC 5114 and

[

- ***[14 (2048-bit MODP)] according to RFC 3526,***
- ***[24 (2048-bit MODP with 256-bit POS] according to RFC 5114***

]

FCS_IPSEC_EXT.1.9 The TSF shall generate the secret value x used in the IKE Diffie-Hellman key exchange ("x" in gx mod p) using the random bit generator specified in FCS_RBG_EXT.1, and having a length of at least [**224, 256, 384**] bits.

FCS_IPSEC_EXT.1.10 The TSF shall generate nonces used in IKE exchanges in a manner such that the probability that a specific nonce value will be repeated during the life a specific IPsec SA is less than 1 in 2^[**256**].

FCS_IPSEC_EXT.1.11 The TSF shall ensure that all IKE protocols perform peer authentication using a [***RSA, ECDSA***] that use X.509v3 certificates that conform to RFC 4945 and [***Pre-shared keys, Pre-shared Keys transmitted via EAP-TLS***].

FCS_IPSEC_EXT.1.12 The TSF shall not establish an SA if the [***IP address, Fully Qualified Domain Name (FQDN), Distinguished Name (DN)***] and [***no other reference identifier type***] contained in a certificate does not match the expected value(s) for the entity attempting to establish a connection.

FCS_IPSEC_EXT.1.13 The TSF shall not establish an SA if the presented identifier does not match the configured reference identifier of the peer.

FCS_IPSEC_EXT.1.14 The [***TSF, VPN Gateway***] shall be able to ensure by default that the strength of the symmetric algorithm (in terms of the number of bits in the key) negotiated to protect the [***IKEv1 Phase 1, IKEv2 IKE_SA***] connection is greater than or equal to the strength of the symmetric algorithm (in terms of the number of bits in the key) negotiated to protect the [***IKEv1 Phase 2, IKEv2 CHILD_SA***] connection.

5.1.2.4 Cryptographic Support for Bluetooth Module

5.1.2.4.1 Bluetooth Key Generation (FCS_CKM_EXT.8)

FCS_CKM_EXT.8.1 The TSF shall generate public/private ECDH key pairs every [**new pairing**].

5.1.2.5 Cryptographic Support for TLS Module

5.1.2.5.1 TLS Protocol (FCS_TLS_EXT.1)

FCS_TLS_EXT.1.1 The TSF shall implement [

- ***TLS as a client***
- ***TLS as a server***
- ***DTLS as a client***
- ***DTLS as a server***

].

5.1.2.5.2 TLS Client Protocol (FCS_TLSC_EXT.1)

FCS_TLSC_EXT.1.1 The TSF shall implement TLS 1.2 (RFC 5246) and [***TLS 1.3 (RFC 8446)***] as a client that supports additional functionality for session renegotiation protection and [

- ***mutual authentication***
- ***supplemental downgrade protection***
- ***session resumption***

] and shall abort attempts by a server to negotiate all other TLS or SSL versions.

FCS_TLSC_EXT.1.2 The TSF shall be able to support the following TLS 1.2 ciphersuites: [

- ***TLS_ECDHE_ECDSA_WITH_AES_256_GCM_SHA384 as defined in RFC 5289 and RFC 8422***
- ***TLS_ECDHE_RSA_WITH_AES_256_GCM_SHA384 as defined in RFC 5289 and RFC 8422***
- ***TLS_RSA_WITH_AES_256_GCM_SHA384 as defined in RFC 5288***
- ***TLS_DHE_RSA_WITH_AES_256_GCM_SHA384 as defined in RFC 5288***
- ***TLS_ECDHE_ECDSA_WITH_AES_256_CBC_SHA384 as defined in RFC 5289***
- ***TLS_ECDHE_RSA_WITH_AES_256_CBC_SHA384 as defined in RFC 5289***
- ***TLS_RSA_WITH_AES_256_CBC_SHA256 as defined in RFC 5246***
- ***TLS_ECDHE_ECDSA_WITH_AES_128_GCM_SHA256 as defined in RFC 5289***

- ***TLS_ECDHE_RSA_WITH_AES_128_GCM_SHA256 as defined in RFC 5289***
- ***TLS_ECDHE_ECDSA_WITH_AES_128_CBC_SHA256 as defined in RFC 5289***
- ***TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA256 as defined in RFC 5289***
- ***TLS_RSA_WITH_AES_128_CBC_SHA256 as defined in RFC 5246***
- ***TLS_RSA_WITH_AES_128_CBC_SHA as defined in RFC 5246***

], the following PP-specific ciphersuites using pre-shared secrets: [

- ***no ciphersuites using pre-shared secrets***

], and the following TLS 1.3 ciphersuites: [

- ***TLS_AES_256_GCM_SHA384 as defined in RFC 8446***
- ***TLS_AES_128_GCM_SHA256 as defined in RFC 8446***

] offering the supported ciphersuites in a client hello message in preference order: [
TLS_AES_256_GCM_SHA384,
TLS_AES_128_GCM_SHA256,
TLS_ECDHE_ECDSA_WITH_AES_256_GCM_SHA384,
TLS_ECDHE_ECDSA_WITH_AES_128_GCM_SHA256,
TLS_ECDHE_RSA_WITH_AES_256_GCM_SHA384,
TLS_ECDHE_RSA_WITH_AES_128_GCM_SHA256,
TLS_DHE_RSA_WITH_AES_256_GCM_SHA384,
TLS_ECDHE_ECDSA_WITH_AES_256_CBC_SHA384,
TLS_ECDHE_ECDSA_WITH_AES_128_CBC_SHA256,
TLS_ECDHE_RSA_WITH_AES_256_CBC_SHA384,
TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA256,
TLS_RSA_WITH_AES_256_GCM_SHA384,
TLS_RSA_WITH_AES_256_CBC_SHA256,
TLS_RSA_WITH_AES_128_CBC_SHA256,
TLS_RSA_WITH_AES_128_CBC_SHA
].

FCS_TLSC_EXT.1.3 The TSF shall not offer ciphersuites indicating the following:

- the null encryption component
- support for anonymous servers
- use of deprecated or export-grade cryptography including DES, 3DES, RC2,
- RC4, or IDEA for encryption
- use of MD

and shall abort sessions where a server attempts to negotiate ciphersuites not enumerated in the client hello message.

FCS_TLSC_EXT.1.4 The TSF shall be able to support the following TLS client hello message extensions:

- signature_algorithms extension (RFC 8446) indicating support for [
  - ***ecdsa-secp384r1_sha384 (RFC 8446)***
  - ***rsa_pkcs1_sha384 (RFC 8446)***

  ], and [

- ***rsa_pss_rsae_sha384 (RFC 8603)***
- ***[rsae-pss/sha256, rsae-pss/sha384, rsae-pss/sha612, rsa/sha256, rsa/sha384, rsa/sha512, ecdsa/sha256, ecdsa/sha512]***

]

- extended_master_secret extension (RFC 7627) enforcing server support
- the following other extensions: [
  - ***signature_algorithms_cert extension (RFC 8446) indicating support for [***
    - ***ecdsa-secp384r1_sha384 (RFC 8446)***
    - ***rsa_pkcs1_sha384 (RFC 8446)***

  ], and [
    - ***rsa_pss_rsae_sha384 (RFC 8603)***
    - ***rsa_pkcs1_sha256 (RFC 8446)***
    - ***rsa_pss_rsae_sha256 (RFC 8446)***
    - ***[rsae-pss/sha256, rsae-pss/sha384, rsae-pss/sha612, rsa/sha256, rsa/sha384, rsa/sha512, ecdsa/sha256, ecdsa/sha512]***

    ]
  - ***supported_versions extension (RFC 8446) indicating support for TLS 1.3***
  - ***supported_groups extension (RFC 7919, RFC 8446) indicating support for [***
    - ***secp256r1***
    - ***secp384r1***
    - ***secp521r1***
    - ***]***
  - ***key_share extension (RFC 8446)***
  - ***post_handshake_auth (RFC 8446),***
  - ***pre_shared_key (RFC 8446), and***
  - ***psk_key_exchange_mode (RFC 8446) indicating DHE or ECDHE mode***
  - ***no other extensions***

] and shall not send the following extensions:

- early_data
- psk_key_exchange_mode indicating PSK only mode.

FCS_TLSC_EXT.1.5 The TSF shall be able to [

- ***verify that a presented identifier of name type: [***
  - ***DNS name type according to RFC 6125***
  - ***URI name type according to RFC 6125[11]***
  - ***Common Name conversion to DNS name according to RFC 6125***
  - ***IPaddress name type according to RFC 5280***

[11] Windows extracts the hostname from the URI and treats it like a DNS name. Full URI matching (e.g., path, scheme) is not performed.

]

- ***interface with a client application requesting the TLS channel to verify that a presented identifier***

] matches a reference identifier of the requested TLS server and shall abort the session if no match is found.

FCS_TLSC_EXT.1.6 The TSF shall not establish a trusted channel if the server certificate is invalid [***except when override is authorized in the case where valid revocation information is not available***].

5.1.2.5.3 TLS Client Support for Mutual Authentication (FCS_TLSC_EXT.2)

FCS_TLSC_EXT.2.1 The TSF shall support mutual authentication using X.509v3 certificates during the handshake and [***in support of post-handshake authentication requests, at no other time***], in accordance with [***RFC 5246, section 7.4.4, RFC 8446, section 4.3.2***].

5.1.2.5.4 TLS Client Support Downgrade Protection (FCS_TLSC_EXT.3)

FCS_TLSC_EXT.3.1 The TSF shall not establish a TLS channel if the server hello message includes [***TLS 1.2 downgrade indicator, TLS 1.1 or below downgrade indicator***] in the server random field.

5.1.2.5.5 TLS Client Support for Renegotiation (FCS_TLSC_EXT.4)

FCS_TLSC_EXT.4.1 The TSF shall support secure renegotiation through use of [***the "renegotiation_info" TLS extension, the TLS_EMPTY_RENEGOTIATION_INFO_SCSV signaling ciphersuite signaling value***] in accordance with RFC 5746, and shall terminate the session if an unexpected server hello is received or [***hello request message is received, in no other case***].

5.1.2.5.6 TLS Client Support for Session Resumption (FCS_TLSC_EXT.5)

FCS_TLSC_EXT.5.1 The TSF shall support session resumption as a client via the use of [***session ID in accordance with RFC 5246, tickets in accordance with RFC 5077, PSK and tickets in accordance with RFC 8446***].

5.1.2.5.7 TLS Client 1.3 Resumption Refinements (FCS_TLSC_EXT.6)

The inclusion of this selection-based component depends upon selection in FCS_TLSC_EXT.5.1.

FCS_TLSC_EXT.6.1 The TSF shall send a psk_key_exchange_mode extension with the value psk_dhe_ke when TLS 1.3 session resumption is offered.

FCS_TLSC_EXT.6.2 The TSF shall not send early data in TLS 1.3 sessions.

5.1.2.5.8 TLS Server Protocol (FCS_TLSS_EXT.1)

FCS_TLSS_EXT.1.1 The TSF shall implement TLS 1.2 (RFC 5246) and [***TLS 1.3 (RFC 8446)***] as a server that supports additional functionality for session renegotiation protection and [

- ***mutual authentication***
- ***supplemental downgrade protection***
- ***session resumption***

] and shall reject connection attempts from clients supporting only TLS 1.1, TLS 1.0, or SSL versions.

FCS_TLSS_EXT.1.2 The TSF shall be able to support the following TLS 1.2 ciphersuites: [

- ***TLS_ECDHE_ECDSA_WITH_AES_256_GCM_SHA384 as defined in RFC 5289 and RFC 8422***
- ***TLS_ECDHE_RSA_WITH_AES_256_GCM_SHA384 as defined in RFC 5289 and RFC 8422***
- ***TLS_RSA_WITH_AES_256_GCM_SHA384 as defined in RFC 5288***
- ***TLS_DHE_RSA_WITH_AES_256_GCM_SHA384 as defined in RFC 5288***
- ***TLS_ECDHE_ECDSA_WITH_AES_256_CBC_SHA384 as defined in RFC 5289***
- ***TLS_ECDHE_RSA_WITH_AES_256_CBC_SHA384 as defined in RFC 5289***
- ***TLS_RSA_WITH_AES_256_CBC_SHA256 as defined in RFC 5246***
- ***TLS_ECDHE_ECDSA_WITH_AES_128_GCM_SHA256 as defined in RFC***
- ***5289***
- ***TLS_ECDHE_RSA_WITH_AES_128_GCM_SHA256 as defined in RFC 5289***
- ***TLS_ECDHE_ECDSA_WITH_AES_128_CBC_SHA256 as defined in RFC 5289***
- ***TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA256 as defined in RFC 5289***
- ***TLS_RSA_WITH_AES_128_CBC_SHA256 as defined in RFC 5246***
- ***TLS_RSA_WITH_AES_128_CBC_SHA as defined in RFC 5246***

], the following PP-specific ciphersuites using pre-shared secrets: [

- ***no ciphersuites using pre-shared secrets***

], and the following TLS 1.3 ciphersuites: [

- ***TLS_AES_256_GCM_SHA384 as defined in RFC 8446***
- ***TLS_AES_128_GCM_SHA256 as defined in RFC 8446***

] using a preference order based on [***RFC 9151 priority, client hello ordering, [***
TLS_AES_256_GCM_SHA384,
TLS_AES_128_GCM_SHA256,
TLS_ECDHE_ECDSA_WITH_AES_256_GCM_SHA384,
TLS_ECDHE_ECDSA_WITH_AES_128_GCM_SHA256,
TLS_ECDHE_RSA_WITH_AES_256_GCM_SHA384,
TLS_ECDHE_RSA_WITH_AES_128_GCM_SHA256,
TLS_DHE_RSA_WITH_AES_256_GCM_SHA384,
TLS_ECDHE_ECDSA_WITH_AES_256_CBC_SHA384,
TLS_ECDHE_ECDSA_WITH_AES_128_CBC_SHA256,
TLS_ECDHE_RSA_WITH_AES_256_CBC_SHA384,
TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA256,
TLS_RSA_WITH_AES_256_GCM_SHA384,
TLS_RSA_WITH_AES_256_CBC_SHA256,
TLS_RSA_WITH_AES_128_CBC_SHA256,
TLS_RSA_WITH_AES_128_CBC_SHA
]].

FCS_TLSS_EXT.1.3 The TSF shall not establish a connection with a client that does not indicate support for at least one of the supported ciphersuites.

FCS_TLSS_EXT.1.4 The TSF shall be able to process the following TLS client hello message extensions:

- signature_algorithms extension (RFC 8446) indicating support for [
  - ***ecdsa-secp384r1_sha384 (RFC 8446)***
  - ***rsa_pkcs1_sha384 (RFC 8446)***

  ], and [
  - ***rsa_pss_rsae_sha384 (RFC 8603)***
  - ***[rsae-pss/sha256, rsae-pss/sha384, rsae-pss/sha612, rsa/sha256, rsa/sha384, rsa/sha512, ecdsa/sha256, ecdsa/sha512]***

  ]
- extended_master_secret extension (RFC 7627) enforcing client support
- the following other extensions: [
  - ***signature_algorithms_cert extension (RFC 8446) indicating support for [***
    - ***ecdsa-secp384r1_sha384 (RFC 8446)***
    - ***rsa_pkcs1_sha384 (RFC 8446)***

    ], and [
    - ***rsa_pss_rsae_sha384 (RFC 8603)***
    - ***rsa_pkcs1_sha256 (RFC 8446)***
    - ***rsa_pss_rsae_sha256 (RFC 8446)***
    - ***[rsae-pss/sha256, rsae-pss/sha384, rsae-pss/sha612, rsa/sha256, rsa/sha384, rsa/sha512, ecdsa/sha256, ecdsa/sha512]***

    ]
  - ***supported_versions extension (RFC 8446) indicating support for TLS 1.3***
  - ***supported_groups extension (RFC 7919, RFC 8446) indicating support for [***
    - ***secp256r1***
    - ***secp384r1***
    - ***secp521r1***

    ]
  - ***key_share extension (RFC 8446)***

].

FCS_TLSS_EXT.1.5 The TSF shall perform key establishment for TLS using [

- ***RSA with size [2048, 3072, 4096] bits and no other sizes***
- ***Diffie-Hellman parameters with size [2048, 3072, 4096, 6144, 8192] bits and no other sizes***
- ***ECDHE parameters using elliptic curves [secp256r1, secp384r1, secp521r1] and no other curves, consistent with the client's supported groups extension and [key share] extension and using non-compressed formatting for points***

].

5.1.2.5.9 TLS Server Support for Mutual Authentication (FCS_TLSS_EXT.2)

FCS_TLSS_EXT.2.1 The TSF shall support authentication of TLS clients using X.509v3 certificates during the TLS handshake and [***during post-handshake requests, at no other time***] using the certificate types indicated in the client's signature_algorithms and [***signature_algorithms_cert, no other***] extension.

FCS_TLSS_EXT.2.2 The TSF shall support authentication of TLS clients using X.509v3 certificates in accordance with FIA_X509_EXT.1.

FCS_TLSS_EXT.2.3 The TSF shall be able to reject the establishment of a trusted channel if the requested client certificate is invalid and [

- ***continue establishment of a server-only authenticated TLS channel in accordance with FCS_TLSS_EXT.1 in support of [[any TLS server applications that choose to accept both authenticated and unauthenticated client sessions]] when an empty certificate message is provided by the client***
- ***continue establishment of a mutually authenticated TLS channel when revocation status information for the [client's leaf certificate, [intermediate CA certificates], any non-trust store certificate in the certificate chain] is not available in support of [[any TLS server application that chooses not to act on the revocation information for the TLS client]] as [a default for [TLS server applications that choose not to act on the revocation information for the TLS client]]***

].

FCS_TLSS_EXT.2.4 The TSF shall be able to [

- ***not establish a TLS session if an entry of the Distinguished Name or a [dns_name, [Common Name, IP address]] in the Subject Alternate Name extension contained in the client certificate does not match one of the expected identifiers for the client in accordance with [RFC 6125, RFC 5280] matching rules***
- ***pass the [validated certificate, DNS name normalized according to RFC 6125, [IP address normalized as in RFC 5280]] to [TLS server applications capable of making access decisions]***

].

5.1.2.5.10 TLS Server Support Downgrade Protection (FCS_TLSS_EXT.3)

FCS_TLSS_EXT.3.1 The TSF shall set the server hello extension to a random value concatenated with the TLS 1.2 downgrade indicator when negotiating TLS 1.2 as indicated in RFC 8446 section 4.1.3.

5.1.2.5.11 TLS Server Support for Session Resumption (FCS_TLSS_EXT.5)

The inclusion of this selection-based component depends upon selection in FCS_TLSS_EXT.1.1.

FCS_TLSS_EXT.5.1 The TSF shall support session resumption as a server via the use of [***session ID in accordance with RFC 5246, tickets in accordance with RFC 5077, PSK and tickets in accordance with RFC 8446***].

5.1.2.5.12 TLS Server TLS 1.3 Resumption Requirements (FCS_TLSS_EXT.6)

The inclusion of this selection-based component depends upon selection in FCS_TLSS_EXT.5.1.

FCS_TLSS_EXT.6.1 The TSF shall support TLS 1.3 resumption using PSK with psk_key_exchange_mode extension with the value psk_dhe_ke.

FCS_TLSS_EXT.6.2 The TSF shall ignore early data received in TLS 1.3 sessions.

5.1.2.5.13 DTLS Client Protocol (FCS_DTLSC_EXT.1)

FCS_DTLSC_EXT.1.1 The TSF shall implement DTLS 1.2 (RFC 6347) and [***no other TLS versions***] as a client that supports additional functionality for session renegotiation protection and [

- ***mutual authentication***
- ***supplemental downgrade protection***
- ***session resumption***

] and shall abort attempts by a server to negotiate all other DTLS versions.

FCS_DTLSC_EXT.1.2 The TSF shall be able to support the following TLS 1.2 ciphersuites: [

- ***TLS_ECDHE_ECDSA_WITH_AES_256_GCM_SHA384 as defined in RFC 5289 and RFC 8422***
- ***TLS_ECDHE_RSA_WITH_AES_256_GCM_SHA384 as defined in RFC 5289 and RFC 8422***
- ***TLS_RSA_WITH_AES_256_GCM_SHA384 as defined in RFC 5288***
- ***TLS_DHE_RSA_WITH_AES_256_GCM_SHA384 as defined in RFC 5288***
- ***TLS_ECDHE_ECDSA_WITH_AES_256_CBC_SHA384 as defined in RFC 5289***
- ***TLS_ECDHE_RSA_WITH_AES_256_CBC_SHA384 as defined in RFC 5289***
- ***TLS_RSA_WITH_AES_256_CBC_SHA256 as defined in RFC 5246***
- ***TLS_ECDHE_ECDSA_WITH_AES_128_GCM_SHA256 as defined in RFC 5289***
- ***TLS_ECDHE_RSA_WITH_AES_128_GCM_SHA256 as defined in RFC 5289***
- ***TLS_ECDHE_ECDSA_WITH_AES_128_CBC_SHA256 as defined in RFC 5289***
- ***TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA256 as defined in RFC 5289***
- ***TLS_RSA_WITH_AES_128_CBC_SHA256 as defined in RFC 5246***
-
- ***TLS_RSA_WITH_AES_128_CBC_SHA as defined in RFC 5246***

], the following PP-specific ciphersuites using pre-shared secrets: [

- ***no ciphersuites using pre-shared secrets***

], and the following TLS 1.3 ciphersuites: [

- ***no TLS 1.3 ciphersuites***

] offering the supported ciphersuites in a client hello message in preference order: [

TLS_ECDHE_ECDSA_WITH_AES_256_GCM_SHA384,
TLS_ECDHE_ECDSA_WITH_AES_128_GCM_SHA256,
TLS_ECDHE_RSA_WITH_AES_256_GCM_SHA384,
TLS_ECDHE_RSA_WITH_AES_128_GCM_SHA256,
TLS_DHE_RSA_WITH_AES_256_GCM_SHA384,
TLS_ECDHE_ECDSA_WITH_AES_256_CBC_SHA384,

TLS_ECDHE_ECDSA_WITH_AES_128_CBC_SHA256,
TLS_ECDHE_RSA_WITH_AES_256_CBC_SHA384,
TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA256,
TLS_RSA_WITH_AES_256_GCM_SHA384,
TLS_RSA_WITH_AES_256_CBC_SHA256,
TLS_RSA_WITH_AES_128_CBC_SHA256,
TLS_RSA_WITH_AES_128_CBC_SHA
].

FCS_DTLSC_EXT.1.3 The TSF shall not offer ciphersuites indicating the following:

- the null encryption component
- support for anonymous servers
- use of deprecated or export-grade cryptography including DES, 3DES, RC2, RC4, or IDEA for encryption
- use of MD

and shall abort sessions where a server attempts to negotiate ciphersuites not enumerated in the client hello message.

FCS_DTLSC_EXT.1.4 The TSF shall be able to support the following TLS client hello message extensions:

- signature_algorithms extension (RFC 8446) indicating support for [
  - ***ecdsa-secp384r1_sha384 (RFC 8446)***
  - ***rsa_pkcs1_sha384 (RFC 8446)***

  ], and [
  - ***rsa_pss_rsae_sha384 (RFC 8603)***

  ]
- extended_master_secret extension (RFC 7627) enforcing server support
- the following other extensions: [
  - signature_algorithms_cert extension (RFC 8446) indicating support for [
    - ***ecdsa-secp384r1_sha384 (RFC 8446)***
    - ***rsa_pkcs1_sha384 (RFC 8446)***

    ], and [
    - ***rsa_pss_rsae_sha384 (RFC 8603)***
    - ***rsa_pkcs1_sha256 (RFC 8446)***
    - ***rsa_pss_rsae_sha256 (RFC 8446)***
    - ***[rsae-pss/sha256, rsae-pss/sha384, rsae-pss/sha612, rsa/sha256, rsa/sha384, rsa/sha512, ecdsa/sha256, ecdsa/sha512]***
    -

    ]
  - supported_groups extension (RFC 7919, RFC 8446) indicating support for [
    - ***secp256r1***
    - ***secp384r1***
    - ***secp521r1***

    ]
- ***key_share extension (RFC 8446)***
- ***post_handshake_auth (RFC 8446), pre_shared_key (RFC 8446), and***

- ***psk_key_exchange_mode (RFC 8446) indicating DHE or ECDHE mode***
- ***no other extensions***

] and shall not send the following extensions:

- early_data
- psk_key_exchange_mode indicating PSK only mode.

FCS_DTLSC_EXT.1.5 The TSF shall be able to [

- ***verify that a presented identifier of name type: [***
  - ***DNS name type according to RFC 6125***
  - ***URI name type according to RFC 6125***[12]
  - ***Common Name conversion to DNS name according to RFC 6125***
  - ***IPaddress name type according to RFC 5280***

  ]
- ***interface with a client application requesting the DTLS channel to verify that a presented identifier***

] matches a reference identifier of the requested DTLS server and shall abort the session if no match is found.

FCS_DTLSC_EXT.1.6 The TSF shall not establish a trusted channel if the server certificate is invalid [***except when override is authorized in the case where valid revocation information is not available***].

FCS_DTLSC_EXT.1.7 The TSF shall [***terminate the DTLS session, silently discard the record***] if a message received contains an invalid MAC or if decryption fails in the case of GCM and other AEAD ciphersuites.

5.1.2.5.14 DTLS Client Support for Mutual Authentication (FCS_DTLSC_EXT.2)

FCS_DTLSC_EXT.2.1 The TSF shall support mutual authentication using X.509v3 certificates during the handshake and [***in support of post-handshake authentication requests, at no other time***], in accordance with [***RFC 5246 section 7.4.4, RFC 8446 section 4.3.2***].

5.1.2.5.15 DTLS Client Downgrade Protection (FCS_DTLSC_EXT.3)

FCS_DTLSC_EXT.3.1 The TSF shall not establish a DTLS channel if the server hello message includes a [***TLS 1.2 downgrade indicator, TLS 1.1 or below downgrade indicator***] in the server random field.

5.1.2.5.16 [D]TLS Client Support for Renegotiation (FCS_DTLSC_EXT.4)

FCS_DTLSC_EXT.4.1 The TSF shall support secure renegotiation through use of [***the "renegotiation_info" TLS extension, the TLS_EMPTY_RENEGOTIATION_INFO_SCSV signaling ciphersuite signaling value***] in accordance with RFC 5746, and shall terminate the session if an unexpected server hello is received or [***hello request message is received, in no other case***]

[12] Windows extracts the hostname from the URI and treats it like a DNS name. Full URI matching (e.g., path, scheme) is not performed.

5.1.2.5.17 DTLS Client Support for Session Resumption (FCS_DTLSC_EXT.5)

FCS_DTLSC_EXT.5.1 The TSF shall support session resumption as a client via the use of [***session ID in accordance with RFC 5246, tickets in accordance with RFC 5077***].

5.1.2.5.18 DTLS Server Protocol (FCS_DTLSS_EXT.1)

FCS_DTLSS_EXT.1.1 The TSF shall implement DTLS 1.2 (RFC 6347) and [***no earlier DTLS versions***] as a server that supports additional functionality for session renegotiation protection and [

- ***mutual authentication***
- ***supplemental downgrade protection***
- ***session resumption***
- ***no optional functionality***

] and shall reject connection attempts from clients supporting only DTLS 1.0.

FCS_DTLSS_EXT.1.2 The TSF shall be able to support the following TLS 1.2 ciphersuites: [

- ***TLS_ECDHE_ECDSA_WITH_AES_256_GCM_SHA384 as defined in RFC 5289 and RFC 8422***
- ***TLS_ECDHE_RSA_WITH_AES_256_GCM_SHA384 as defined in RFC 5289 and RFC 8422***
- ***TLS_RSA_WITH_AES_256_GCM_SHA384 as defined in RFC 5288***
- ***TLS_DHE_RSA_WITH_AES_256_GCM_SHA384 as defined in RFC 5288***
- ***TLS_ECDHE_ECDSA_WITH_AES_256_CBC_SHA384 as defined in RFC 5289***
- ***TLS_ECDHE_RSA_WITH_AES_256_CBC_SHA384 as defined in RFC 5289***
- ***TLS_RSA_WITH_AES_256_CBC_SHA256 as defined in RFC 5246***
- ***TLS_ECDHE_ECDSA_WITH_AES_128_GCM_SHA256 as defined in RFC 5289***
- ***TLS_ECDHE_RSA_WITH_AES_128_GCM_SHA256 as defined in RFC 5289***
- ***TLS_ECDHE_ECDSA_WITH_AES_128_CBC_SHA256 as defined in RFC 5289***
- ***TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA256 as defined in RFC 5289 TLS_RSA_WITH_AES_128_CBC_SHA256 as defined in RFC 5246***
-
- ***TLS_RSA_WITH_AES_128_CBC_SHA as defined in RFC 5246***

], the following PP-specific ciphersuites using pre-shared secrets: [

- ***no ciphersuites using pre-shared secrets***

], and the following TLS 1.3 ciphersuites: [

- ***no TLS 1.3 ciphersuites***

] using a preference order based on [***RFC 9151 priority, client hello ordering, [***
TLS_ECDHE_ECDSA_WITH_AES_256_GCM_SHA384,
TLS_ECDHE_ECDSA_WITH_AES_128_GCM_SHA256,
TLS_ECDHE_RSA_WITH_AES_256_GCM_SHA384,
TLS_ECDHE_RSA_WITH_AES_128_GCM_SHA256,
TLS_DHE_RSA_WITH_AES_256_GCM_SHA384,
TLS_ECDHE_ECDSA_WITH_AES_256_CBC_SHA384,
TLS_ECDHE_ECDSA_WITH_AES_128_CBC_SHA256,

TLS_ECDHE_RSA_WITH_AES_256_CBC_SHA384,
TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA256,
TLS_RSA_WITH_AES_256_GCM_SHA384,
TLS_RSA_WITH_AES_256_CBC_SHA256,
TLS_RSA_WITH_AES_128_CBC_SHA256,
TLS_RSA_WITH_AES_128_CBC_SHA
]]

FCS_DTLSS_EXT.1.3 The TSF shall not establish a connection with a client that does not indicate support for at least one of the supported ciphersuites.

FCS_DTLSS_EXT.1.4 The TSF shall be able to process the following TLS client hello message extensions:

- signature_algorithms extension (RFC 8446) indicating support for [
  - ***ecdsa-secp384r1_sha384 (RFC 8446)***
  - ***rsa_pkcs1_sha384 (RFC 8446)***

  ], and [
  - ***rsa_pss_rsae_sha384 (RFC 8603)***
  - ***[rsae-pss/sha256, rsae-pss/sha384, rsae-pss/sha612, rsa/sha256, rsa/sha384, rsa/sha512, ecdsa/sha256, ecdsa/sha512]***
  -

  ]
- extended_master_secret extension (RFC 7627) enforcing client support
- the following other extensions: [
  - ***signature_algorithms_cert extension (RFC 8446) indicating support for [***
    - ***ecdsa-secp384r1_sha384 (RFC 8446)***
    - ***rsa_pkcs1_sha384 (RFC 8446)***

    ], and [
    - ***rsa_pss_rsae_sha384 (RFC 8603)***
    - ***rsa_pkcs1_sha256 (RFC 8446)***
    - ***rsa_pss_rsae_sha256 (RFC 8446)***
    - ***[rsae-pss/sha256, rsae-pss/sha384, rsae-pss/sha612, rsa/sha256, rsa/sha384, rsa/sha512, ecdsa/sha256, ecdsa/sha512]***

    ]
  - ***supported_versions extension (RFC 8446) indicating support for TLS 1.3***
  - ***supported_groups extension (RFC 7919, RFC 8446) indicating support for [***
    - ***secp256r1***
    - ***secp384r1***
    - ***secp521r1***

    ]
  - ***key_share extension (RFC 8446)***

].

FCS_DTLSS_EXT.1.5 The TSF shall perform key establishment for DTLS using [

- ***RSA with size [2048, 3072, 4096] bits and no other sizes***
- ***Diffie-Hellman parameters with size [2048, 3072, 4096, 6144, 8192] bits and no other sizes***
- ***ECDHE parameters using elliptic curves [secp256r1, secp384r1, secp521r1] and no other curves, consistent with the client's supported groups extension and [key share] extension and using non-compressed formatting for points***

].

FCS_DTLSS_EXT.1.6 The TSF shall not proceed with a connection handshake attempt if the DTLS client fails validation.

5.1.2.5.19 DTLS Server Support for Mutual Authentication (FCS_DTLSS_EXT.2)

FCS_DTLSS_EXT.2.1 The TSF shall support authentication of DTLS clients using X.509v3 certificates during the DTLS handshake and [***during post-handshake requests, at no other time***] using the certificate types indicated in the client's signature_algorithms and [***signature_algorithms_cert, no other***] extension.

FCS_DTLSS_EXT.2.2 The TSF shall support authentication of DTLS clients using X.509v3 certificates in accordance with FIA_X509_EXT.1.

FCS_DTLSS_EXT.2.3 The TSF shall be able to reject the establishment of a trusted channel if the requested client certificate is invalid and [

- ***continue establishment of a server-only authenticated DTLS channel in accordance with FCS_DTLSS_EXT.1 in support of [[any DTLS server applications that choose to accept both authenticated and unauthenticated client sessions]] when an empty certificate message is provided by the client***
- ***continue establishment of a mutually authenticated DTLS channel when revocation status information for the [client's leaf certificate, [intermediate CA certificates], any non-trust store certificate in the certificate chain] is not available in support of [[DTLS server application that chooses not to act on the revocation information for the DTLS client]] as [a default for [DTLS server applications that choose not to act on the revocation information for the DTLS client]]***

]

FCS_DTLSS_EXT.2.4 The TSF shall be able to [

- ***not establish a DTLS session if an entry of the Distinguished Name or a [dns_name, [Common Name, IP address]] in the Subject Alternate Name extension contained in the client certificate does not match one of the expected identifiers for the client in accordance with [RFC 6125, RFC 5280[ ] matching rules***
- ***pass the [validated certificate, DNS name normalized according to RFC 6125, [IP address normalized as in RFC 5280]]] to [DTLS server applications capable of making access decisions]***

].

5.1.2.5.20 DTLS Server Downgrade Protection (FCS_DTLSS_EXT.3)

FCS_DTLSS_EXT.3.1 The TSF shall set the server hello extension to a random value concatenated with the TLS 1.2 downgrade indicator when negotiating DTLS 1.2 as indicated in RFC 8446 section 4.1.3.

5.1.2.5.21 DTLS Server Support for Session Resumption (FCS_DTLSS_EXT.5)

FCS_DTLSS_EXT.5.1 The TSF shall support session resumption as a server via the use of [***session ID in accordance with RFC 5246, tickets in accordance with RFC 5077***].

5.1.3 User Data Protection (FDP)

5.1.3.1 User Data Protection for GP OS PP

5.1.3.1.1 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.

5.1.3.1.2 Information Flow Control (FDP_IFC_EXT.1)

FDP_IFC_EXT.1.1 The OS shall [

- ***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 and [***no other traffic***].

5.1.3.2 User Data Protection for VPN Client Module

5.1.3.2.1 Spit Tunnel Prevention (FDP_VPN_EXT.1)

FDP_VPN_EXT.1.1 The TSF shall ensure that all IP traffic (other than IP traffic required to establish the VPN connection) flow through the IPsec VPN client.

5.1.3.2.2 Full Residual Information Protection (FDP_RIP.2)

FDP_RIP.2.1 The [***TOE***] shall enforce that any previous information content of a resource is made unavailable upon the [***allocation of the resource to***] all objects.

5.1.4 Identification and Authentication (FIA)

5.1.4.1 Identification and Authentication for GP OS PP

5.1.4.1.1 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 occur related to events with [

- ***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***
- ***authentication based on X.509 certificates***

].

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

5.1.4.1.2 Multiple Authentication Mechanisms (FIA_UAU.5)

FIA_UAU.5.1 The OS shall provide the following authentication mechanisms:
[
- ***Authentication based on username and password,***
- ***authentication based on user name and a PIN that releases an asymmetric key stored in OE-protected storage[13]***
- ***authentication based on X.509 certificates***

] 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, virtual smart card, and X.509 certificate**].

5.1.4.1.3 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, that the CA flag is set to TRUE for all CA certificates, and that any path constraints are met.
- The TSF shall validate that any CA certificate includes "Certificate Signing" as a purpose in the key usage field
- The OS shall validate the revocation status of the certificate using [***OCSP as specified in RFC 6960, CRL as specified in RFC 8603 ~~5759~~, an OCSP TLS Status Request Extension (OCSP stapling) as specified in RFC 6066, OCSP TLS Multi-Certificate Status Request Extension (i.e., OCSP Multi-Stapling) as specified in RFC 6961***] with [***no exceptions***]
- The OS shall validate the extendedKeyUsage field according to the following rules:
  - Certificates used for trusted updates and executable code integrity verification shall have the Code Signing Purpose (id-kp 3 with OID 1.3.6.1.5.5.7.3.3) in the extendedKeyUsage field.
  - Server certificates presented for TLS shall have the Server Authentication purpose (id-kp 1 with OID 1.3.6.1.5.5.7.3.1) in the extendedKeyUsage field.
  - Client certificates presented for TLS shall have the Client Authentication purpose (id-kp 2 with OID 1.3.6.1.5.5.7.3.2) in the EKU field.
  - 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 EKU field.

~~[13] PIN-based authentication is for Windows 11, smart card authentication is for Windows 11 and Windows Server 2025 only.~~

- OCSP certificates presented for OCSP responses shall have the OCSP Signing Purpose (id-kp 9 with OID 1.3.6.1.5.5.7.3.9) in the EKU field.
- Server certificates presented for EST shall have the CMC Registration Authority (RA) purpose (id-kp-cmcRA with OID 1.3.6.1.5.5.7.3.28) in the EKU field. (conditional)

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.

5.1.4.1.4 X.509 Certificate Authentication (FIA_X509_EXT.2)

FIA_X509_EXT.2.1[14] The OS shall use X.509v3 certificates as defined by RFC 5280 to support authentication for [TLS, ***DTLS, HTTPS, [IPsec]***] connections.

5.1.4.2 Identification and Authentication for WLAN Client Module

5.1.4.2.1 Port Access Entity Authentication (FIA_PAE_EXT.1)

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

5.1.4.2.2 X.509 Certificate Validation (FIA_X509_EXT.1(WLAN))

Application Note: FIA_X509_EXT.1(WLAN) corresponds to FIA_X509_EXT.1/WLAN in the WLAN Client Module.

FIA_X509_EXT.1.1(WLAN) The TSF shall validate certificates for EAP-TLS in accordance with the following rules:

- RFC 5280 certificate validation and certificate path validation
- The certificate path must terminate with a certificate in the Trust Anchor Database
- The TSF shall validate a certificate path by ensuring the presence of the basicConstraints extension and that the CA flag is set to TRUE for all CA certificates
- The TSF shall validate the extendedKeyUsage field according to the following rules:
  - Server certificates presented for TLS shall have the Server Authentication purpose (id-kp 1 with OID 1.3.6.1.5.5.7.3.1) in the extendedKeyUsage field
  - Client certificates presented for TLS shall have the Client Authentication purpose (id-kp 2 with OID 1.3.6.1.5.5.7.3.2) in the extendedKeyUsage field.

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

5.1.4.2.3 X.509 Certificate Authentication EAP-TLS for WLAN (FIA_X509_EXT.2(WLAN))

Application Note: FIA_X509_EXT.2(WLAN) corresponds to FIA_X509_EXT.2/WLAN in the WLAN Client module.

[14] This protection profile requirement was replaced as part of NIAP Technical Decision 789.

FIA_X509_EXT.2.1(WLAN) The TSF shall use X.509v3 certificates as defined by RFC 5280 to support authentication for EAP-TLS exchanges.

5.1.4.2.4 Certificate Storage and Management (FIA_X509_EXT.6)

FIA_X509_EXT.6.1 The TSF shall [***store and protect***] certificate(s) from unauthorized deletion and modification.

FIA_X509_EXT.6.2 The TSF shall [***provide the capability for authorized administrators to load X.509v3 certificates into the TOE***] for use by the TSF.

5.1.4.3 Identification and Authentication for VPN Client Module

5.1.4.3.1 Pre-Shared Key Composition (FIA_PSK_EXT.1)

FIA_PSK_EXT.1.1 The TSF shall be able to use pre-shared keys for IPsec and [***no other protocols***].

FIA_PSK_EXT.1.2 The TSF shall be able to accept the following as pre-shared keys: [***generated bit-based***] keys.

5.1.4.3.2 Generated Pre-Shared Keys (FIA_PSK_EXT.2)

FIA_PSK_EXT.2.1 The TSF shall be able to [***accept externally generated pre-shared keys***].

5.1.4.3.3 X.509 Certificate Use and Management (FIA_X509_EXT.3)

FIA_X509_EXT.3.1 The TSF shall use X.509v3 certificates as defined by RFC 5280 to support authentication for IPsec exchanges, and [***digital signatures for FPT_TUD_EXT.1, integrity checks for FPT_TST_EXT.1***].

FIA_X509_EXT.3.2 When a connection to determine the validity of a certificate cannot be established, the [***OS***] shall [***not accept the certificate***].

FIA_X509_EXT.3.3 The [***VPN client***] shall not establish an SA if a certificate or certificate path is deemed invalid.

5.1.4.4 Identification and Authentication for Bluetooth Module

5.1.4.4.1 Bluetooth User Authorization (FIA_BLT_EXT.1)

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

5.1.4.4.2 Bluetooth Mutual Authentication (FIA_BLT_EXT.2)

FIA_BLT_EXT.2.1 The TSF shall require Bluetooth mutual authentication between devices prior tony data transfer over the Bluetooth link.

5.1.4.4.3 Rejection of Duplicate Bluetooth Connections (FIA_BLT_EXT.3)

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

5.1.4.4.4 Secure Simple Pairing (FIA_BLT_EXT.4)

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

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

5.1.4.4.5 Trusted Bluetooth Device User Authorization (FIA_BLT_EXT.6)

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

5.1.4.4.6 Untrusted Bluetooth Device User Authorization (FIA_BLT_EXT.7)

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

5.1.5 Security Management (FMT)

5.1.5.1 Security Management for GP OS PP

5.1.5.1.1 Management of Security Functions Behavior (FMT_MOF_EXT.1)

FMT_MOF_EXT.1.1 The OS shall restrict the ability to perform the function indicated in the "Administrator" column in FMT_SMF_EXT.1.1 to the administrator.

5.1.5.1.2 Specification of Security Functions Behavior for OS (FMT_SMF_EXT.1)[15]

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

Table 25 TOE Security Management Functions

#	Management Function	Administrator	User
1.	Enable/disable [***screen lock, session timeout***]	M	O
2.	Configure [***screen lock, session]*** inactivity timeout	M	O
3.	Import keys/secrets into the secure key storage	O	O
4.	Configure local audit storage capacity	O	O
5.	Configure minimum password Length	O	O
6.	~~Configure minimum number of special characters in password~~	~~O~~	~~O~~
7.	~~Configure minimum number of numeric characters in password~~	~~O~~	~~O~~
8.	~~Configure minimum number of uppercase characters in password~~	~~O~~	~~O~~
9.	~~Configure minimum number of lowercase characters in password~~	~~O~~	~~O~~
10.	Configure lockout policy for unsuccessful authentication attempts through [***timeouts between attempts, limiting number of attempts during a time period***]	O	O
11.	Configure host-based firewall	O	O

[15] This security functional requirement was updated as part of NIAP Technical Decision 693.

12.	Configure name/address of directory server to bind with	O	O
13.	Configure name/address of remote management server from which to receive management settings	O	O
14.	~~Configure name/address of audit/logging server to which to send audit/logging records~~	~~O~~	~~O~~
15.	Configure audit rules	O	O
16.	Configure name/address of network time server	O	O
17.	Enable/disable automatic software update	O	O
18.	Configure Wi-Fi interface	O	O
19.	Enable/disable Bluetooth interface	M	~~O~~
20.	Enable/disable [**local area network interface, configure USB interfaces**]	O	O
21.	[**manage Windows Diagnostics settings, Configure remote connection inactivity timeout**]	O	O

5.1.5.2 Security Management for WLAN Client Module

5.1.5.2.1 Specification of Management Functions for (WLAN Client) (FMT_SMF.1(WLAN)) [16]

Application Note: FMT_SMF.1(WLAN) corresponds to FMT_SMF.1/WLAN in the WLAN Client module.

FMT_SMF.1.1(WLAN) The TSF shall be capable of performing the following management functions:
Table **24** ~~3~~: Management Functions

Status Markers:
M - Mandatory
O - Optional/Objective

Table 26 WLAN Client Module Management Functions

#	Management Function	Impl.	Admin	User
WL-1	configure security policy for each wireless network: • [***specify the CA(s) from which the TSF will accept WLAN authentication server***	M	M	O

[16] This protection profile module requirement was modified as part of NIAP Technical Decision 667.

	certificate(s), specify the Fully Qualified Domain Names (FQDNs) of acceptable WLAN authentication server certificate(s)], • security type, • authentication protocol, • client credentials to be used for authentication			
WL-2	specify wireless networks (SSIDs) to which the TSF may connect	M	M	O
WL-3	enable/disable ~~disable~~ wireless network bridging capability (for example, bridging a connection between the WLAN and cellular radios to function as a hotspot) authenticated by [***pre-shared key, passcode, no authentication***]	M	M	O
WL-4	enable/disable certificate revocation list checking	O	O	O
WL-5	disable ad hoc wireless client-to-client connection capability	O	O	O
WL-6	disable roaming capability	O	O	O
WL-7	enable/disable IEEE 802.1X pre-authentication	O	O	O
WL-8	loading X.509 certificates into the TOE	O	O	O
WL-9	revoke X.509 certificates loaded into the TOE	O	O	O

WL-10	enable/disable and configure PMK caching: • set the amount of time (in minutes) for which PMK entries are cached, • set the maximum number of PMK entries that can be cached	O	O	O
WL-11	configure security policy for each wireless network: set wireless frequency band to [***2.4 GHz, 5 GHz, 6 GHz***]	O	O	O

5.1.5.3 Security Management for VPN Client Module

5.1.5.3.1 Specification of Management Functions for VPN (FMT_SMF.1(VPN))

Application Note: FMT_SMF.1(VPN) corresponds to FMT_SMF.1/VPN in the VPN Client Module.

FMT_SMF.1.1(VPN) The TSF shall be capable of performing the following management functions: [

- ***Specify VPN gateways to use for connections,***
- ***Specify IPsec VPN Clients to use for connections,***
- ***Specify IPsec-capable network devices to use for connections],***
- ***Specify client credentials to be used for connections,***
- ***Configure the reference identifier of the peer***
- ***[no other actions]***].

5.1.5.4 Security Management for Bluetooth Module

5.1.5.4.1 Management of Security Functions Behavior for Bluetooth (FMT_MOF_EXT.1(BT))

Application Note: FMT_MOF_EXT.1(BT) corresponds to FMT_MOF_EXT.1/BT in the Bluetooth Module.

FMT_MOF_EXT.1.1(BT) The OS shall restrict the ability to perform the function indicated in the "Administrator" column in FMT_SMF_EXT.1.1**(BT)**~~/BT~~ to the administrator.

5.1.5.4.2 Specification of Management Functions for Bluetooth (FMT_SMF_EXT.1(BT))

Application Note: FMT_SMF_EXT.1(BT) corresponds to FMT_SMF_EXT.1/BT in the Bluetooth Module.

FMT_SMF_EXT.1.1(BT) The TSF shall be capable of performing the following Bluetooth management functions:

Table 27 Bluetooth Security Management Functions

Function	Administrator	User

BT-1. Configure the Bluetooth trusted channel. • **Disable/enable the Discoverable (for BR/EDR) and Advertising (for LE) modes;**	X	O
~~BT-2. Change the Bluetooth device name (separately for BR/EDR and LE);~~	~~O~~	~~O~~
~~BT-3. Provide separate controls for turning the BR/EDR and LE radios on and off;~~	~~O~~	~~O~~
~~BT-4. Allow/disallow the following additional wireless technologies to be used with Bluetooth: [selection: Wi-Fi, NFC, [assignment: other wireless technologies]];~~	~~O~~	~~O~~
~~BT-5. Configure allowable methods of Out of Band pairing (for BR/EDR and LE);~~	~~O~~	~~O~~
BT-6. Disable/enable the Discoverable (for BR/EDR) and Advertising (for LE) modes separately;	O	O
~~BT-7. Disable/enable the Connectable mode (for BR/EDR and LE);~~	~~O~~	~~O~~
BT-8. Disable/enable the Bluetooth [**all Bluetooth services**];	O	O
~~BT-9. Specify minimum level of security for each pairing (for BR/EDR and LE);~~	~~O~~	~~O~~

5.1.6 Protection of the TSF (FPT)

5.1.6.1 Protection of the TSF for GP OS PP

5.1.6.1.1 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
- 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**]

5.1.6.1.2 Address Space Layout Randomization (FPT_ASLR_EXT.1)

FPT_ASLR_EXT.1.1 The OS shall always randomize process address space memory locations with [***8 bits of entropy for 32-bit applications and at least 17 bits of entropy for 64-bit applications***] ~~bits of entropy~~ except for [**none**].

5.1.6.1.3 Limitation of Bluetooth Profile Support (FPT_BLT_EXT.1)

FPT_BLT_EXT.1.1 The TSF shall disable support for [**all Bluetooth profiles**] Bluetooth profiles when they are not currently being used by an application on the TOE and shall require explicit user action to enable them.

5.1.6.1.4 Buffer Overflow Protection (FPT_SBOP_EXT.1)

FPT_SBOP_EXT.1.1 The OS shall [***employ stack-based buffer overflow protections, not store parameters/variables in the same data structures as control flow values***].

5.1.6.1.5 Software Restriction Policies (FPT_SRP_EXT.1)

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

- ***File path,***
- ***File digital signature,***
- ***Version,***[17]
- ***Hash***

].

[17] Windows 11 Enterprise and Windows Server 2025 can restrict program execution based on a version using AppLocker and Device Guard.

5.1.6.1.6 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 digital signature using an X509 certificate with hardware-based protection, a hardware-protected hash***].[18,]

5.1.6.1.7 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 and shall use a digital signature scheme specified in FCS_COP.1/SIGN to validate the authenticity of the response.

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.

5.1.6.1.8 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 and shall use a digital signature scheme specified in FCS_COP.1/SIGN to validate the authenticity of the response.

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.

5.1.6.2 Protection of the TSF for WLAN Client Module

5.1.6.2.1 TSF Cryptographic Functionality Testing (FPT_TST_EXT.3 (WLAN))

Application Note: FPT_TST_EXT.3(WLAN) corresponds to FPT_TST_EXT.3/WLAN in the WLAN Client module.

FPT_TST_EXT.3.1(WLAN) The [***TOE***] shall run a suite of self-tests during initial start-up (on power on) to demonstrate the correct operation of the TSF.

FPT_TST_EXT.3.2(WLAN) The [***TOE***] shall provide the capability to verify the integrity of stored TSF executable code when it is loaded for execution through the use of the TSF-provided cryptographic services.

5.1.6.3 Protection of the TSF for VPN Client Module

5.1.6.3.1 Self-Test for IPsec (FPT_TST_EXT.1 (VPN))

Application Note: FPT_TST_EXT.1(VPN) corresponds to FPT_TST_EXT.1/VPN in the VPN Client Module.

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

FPT_TST_EXT.1.2(VPN) The [***TOE***] shall provide the capability to verify the integrity of stored TSF executable code when it is loaded for execution through the use of the [**FCS_COP.1(SIGN) cryptographic services provided by the operating system**].

[18] 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.

5.1.7 TOE Access (FTA)

5.1.7.1 TOE Access for GP OS PP

5.1.7.1.1 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.

5.1.7.2 TOE Access for WLAN Client Module

5.1.7.2.1 Wireless Network Access (FTA_WSE_EXT.1)

FTA_WSE_EXT.1.1 The TSF shall be able to attempt connections only to wireless networks specified as acceptable networks as configured by the administrator in FMT_SMF.1**(WLAN)**.1~~/WLAN~~.

5.1.8 Trusted Path / Channels (FTP)

5.1.8.1 Trusted Path / Channels for GP OS PP

5.1.8.1.1 Trusted Path (FTP_TRP.1)

FTP_TRP.1.1 The OS shall provide a communications path between itself and [***remote, local***] 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.

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[19] The OS shall require use of the trusted path for [***initial user authentication, all remote administrative actions***].

5.1.8.1.2 Trusted Channel Communication (FTP_ITC_EXT.1)

FTP_ITC_EXT.1.1 The OS shall use [

- ***TLS as conforming to Functional Package for Transport Security (TLS), version 2.0 as a [client, server]***
- ***DTLS as conforming to Functional Package for Transport Security (TLS), version 2.0 as a [client, server]***
- ***IPsec as conforming to the PP-Module for Virtual Private Network (VPN) Clients, version 2.4***

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

[19] This protection profile requirement was modified as part of NIAP Technical Decision 839.

5.1.8.2 Trusted Path / Channels for WLAN Client Module

5.1.8.2.1 Trusted Channel Communication (FTP_ITC.1(WLAN))

Application Note: FTP_ITC.1(WLAN) corresponds to FTP_ITC.1/WLAN in the WLAN Client Module.

FTP_ITC.1.1(WLAN)	The TSF shall use 802.11-2012, 802.1X, and EAP-TLS to provide a trusted communication channel between itself and a wireless access point that is logically distinct from other communication channels, provides assured identification of its end points, protects channel data from disclosure, and detects modification of the channel data.
FTP_ITC.1.2(WLAN)	The TSF shall permit [the TSF] to initiate communication via the trusted channel.
FTP_ITC.1.3(WLAN)	The TSF shall initiate communication via the trusted channel for wireless access point connections.

5.1.8.3 Trusted Path / Channels for VPN Client Module

5.1.8.3.1 Inter-TSF Trusted Channel (FTP_ITC.1(VPN))

Application Note: FTP_ITC.1(VPN) corresponds to FTP_ITC.1 in the VPN Client module.

FTP_ITC.1.1(VPN)	The [***VPN client, OS***] shall use IPsec to provide a trusted communication channel between itself and [• ***a remote VPN gateway,*** • ***a remote VPN client,*** • ***a remote IPsec-capable network device***] that is logically distinct from other communication channels and provides assured identification of its end points and protection of the channel data from disclosure and detection of modification of the channel data.
FTP_ITC.1.2(VPN)	The [***OS***] shall permit the TSF to initiate communication via the trusted channel.
FTP_ITC.1.3(VPN)	The [***OS***] shall initiate communication via the trusted channel for all traffic traversing that connection.

5.1.8.4 Trusted Path / Channels for Bluetooth Module

5.1.8.4.1 Bluetooth Encryption (FTP_BLT_EXT.1)

FTP_BLT_EXT.1.1	The TSF shall enforce the use of encryption when transmitting data over the Bluetooth trusted channel for BR/EDR and [***LE***].
FTP_BLT_EXT.1.2	The TSF shall use key pairs per FCS_CKM_EXT.8 for Bluetooth encryption.

5.1.8.4.2 Persistence of Bluetooth Encryption (FTP_BLT_EXT.2)

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

5.1.8.4.3 Bluetooth Encryption Parameters (BR/EDR) (FTP_BLT_EXT.3(BR))[20]

Application Note: FTP_BLT_EXT.3(BR) corresponds to FTP_BLT_EXT.3/BR in the Bluetooth Module.

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

5.1.8.4.4 Bluetooth Encryption Parameters (LE) (FTP_BLT_EXT.3(LE))

Application Note: FTP_BLT_EXT.3(LE) corresponds to FTP_BLT_EXT.3/LE in the Bluetooth Module.

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

5.2 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 Protection Profile for General Purpose Operating Systems.

Table 28 TOE Security Assurance Requirements

Requirement Class	Requirement Component
Security Target (ASE)	ST Introduction (ASE_INT.1)
	Conformance Claims (ASE_CCL.1)
	Security Objectives (ASE_OBJ.2)
	Extended Components Definition (ASE_ECD.1)
	Stated Security Requirements (ASE_REQ.2)
	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)
	Systematic Flaw Remediation (ALC_FLR.3)
	Timely Security Updates (ALC_TSU_EXT.1)
Testing (ATE)	Independent Testing – Conformance (ATE_IND.1)
Vulnerability Assessment (AVA)	Vulnerability Survey (AVA_VAN.1)

[20] This PP-module requirement was replaced as part of NIAP Technical Decision 707.

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 OS.
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 OS software.
ALC-TSU _EXT.1.2C	The description shall include the mechanisms publicly available for reporting security issues pertaining to the OS.

Evaluator action elements:

ALC-TSU_EXT.1.1E	The evaluator will confirm that the information provided meets all requirements for content and presentation of evidence.

Evaluation activities:

ALC_TSU_EXT.1

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 will 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)

5.2.2.1.1 Audit Data Generation (FAU_GEN.1)

FAU_GEN.1.1

Guidance

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

The evaluator will 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 will ensure that the fields contains the information required.

Tests

The evaluator will 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 will 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.

The evaluator will 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 will 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.

The evaluator will check the administrative guide and ensure that it lists all of the auditable events. The evaluator will check to make sure that every audit event type selected in the ST is included. The evaluator will 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 will 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 will 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 will 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 will 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) [21]

Tests

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.

[21] This protection profile assurance activity was replaced as part of NIAP Technical Decision 501 and 873.

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.

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,
- $2^{16} \leq e \leq 2^{256}$ and e is an odd integer,
- $|p-q| > 2^{nlen/2 - 100}$,
- $p \geq 2^{nlen/2 -1/2}$,
- $q \geq 2^{nlen/2 -1/2}$,
- $2^{(nlen/2)} < d < LCM(p-1,q-1)$,
- $e \cdot d = 1 \bmod LCM(p-1,q-1)$.

Key Generation for Elliptic Curve Cryptography (ECC)

FIPS 186-5 ECC Key Generation Test

For each supported NIST curve, i.e., 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 186-5 Public Key Verification (PKV) Test

For each supported NIST curve, i.e., 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.

Key Generation for Finite-Field Cryptography (FFC)

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

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

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

and two ways to generate the cryptographic group generator g:

- Cryptographic Group Generator:
  - Generator g constructed through a verifiable process
  - Generator g constructed through an unverifiable process

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

- Private Key:
  - len(q) bit output of RBG where $1 \le x \le q-1$
  - len(q) + 64 bit output of RBG, followed by a mod q-1 operation where $1 \le x \le q-1$

The security strength of the RBG must be at least that of the security offered by the FFC parameter set. To test the cryptographic and field prime generation method for the provable primes method and/or the group generator g for a verifiable process, the evaluator must seed the TSF parameter generation routine with sufficient data to deterministically generate the parameter set. For each key length supported, the evaluator will have the TSF generate 25 parameter sets and 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. Verification must also confirm:

- g != 0,1
- q divides p-1
- $g^q \bmod p = 1$
- $g^x \bmod p = y$

for each FFC parameter set and key pair.

Diffie-Hellman Group 14 and FFC Schemes using "safe-prime" groups

Testing for FFC Schemes using Diffie-Hellman group 14 and/or "safe-prime" groups is done as part of testing in FCS_CKM.2.1

5.2.2.2.2 Cryptographic Key Establishment (FCS_CKM.2)[22]

Tests

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).

Evaluation 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 will 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.

[22] This protection profile assurance activity was modified as part of NIAP Technical Decision 501.

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.

RSAES-PKCS1-v1_5 Key Establishment Schemes

The evaluator shall verify the correctness of the TSF's implementation of RSAES-PKCS1-v1_5 by using a known good implementation for each protocol selected in FTP_ITC_EXT.1 that uses RSAES-PKCS1-v1_5.

Diffie-Hellman Group 14

The evaluator shall verify the correctness of the TSF's implementation of Diffie-Hellman group 14 by using a known good implementation for each protocol selected in FTP_ITC_EXT.1 that uses Diffie-Hellman Group 14.

FFC Schemes using "safe-prime" groups (identified in Appendix D of SP 800-56A Revision 3)

The evaluator shall verify the correctness of the TSF's implementation of "safe-prime" groups by using a known good implementation for each protocol selected in FTP_ITC_EXT.1 that uses "safe-prime" groups. This test must be performed for each "safe-prime" group that each protocol uses.

5.2.2.2.3 Cryptographic Key Destruction (FCS_CKM_EXT.4)[23]

TSS

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 will 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).

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 will verify that the pattern does not contain any CSPs.

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

If the selection “destruction of all key encrypting keys protecting target key according to FCS_CKM_EXT.4.1, where none of the KEKs protecting the target key are derived” is included the evaluator shall examine the TOE’s keychain in the TSS and identify each instance when a key is destroyed by this method. In each instance the evaluator shall verify all keys capable of decrypting the target key are destroyed in accordance with a specified key destruction method in FCS_CKM_EXT.4.1 The evaluator shall verify that all of the keys capable of decrypting the target key are not able to be derived to reestabish the keychain after their destruction.

Operational Guidance

There are a variety of concerns that may prevent or delay key destruction in some cases. The evaluator will 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 will 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.

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

[23] This protection profile assurance activity was replaced as part of NIAP Technical Decision 365.

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.

Tests

- **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 will:
    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 will 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.

Tests 3 and 4 do not apply for the selection instructing the underlying platform to destroy the representation of the key, as the TOE has no visibility into the inner workings and completely relies on the underlying platform.

- **Test 3**: The following tests are used to determine the TOE is able to request the platform to overwrite the key with a TOE supplied pattern.

  Applied to each key held in non-volatile memory and subject to destruction by overwrite by the TOE. The evaluator will 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 will 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.

  The test succeeds if correct pattern is used to overwrite the key in the memory location. If the pattern is not found the test fails.

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

Guidance

The evaluator will verify that the AGD documents contains instructions required to configure the OS to use the required modes and key sizes.

Tests

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.

- **Test 5**. To test the encrypt functionality of AES-CBC, the evaluator will supply a set of 5 plaintext values and obtain the ciphertext value that results from AES-CBC encryption of the given plaintext using a key value of all zeros and an IV of all zeros. Five plaintext values shall be encrypted with a 128-bit all-zeros key, and the other five shall be encrypted with a 256-bit all-zeros key. To test the decrypt functionality of AES-CBC, the evaluator will perform the same test as for encrypt, using 10 ciphertext values as input and AES-CBC decryption.
- **Test 6**. To test the encrypt functionality of AES-CBC, the evaluator will supply a set of 5 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 128bit 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 AES-CBC decryption.
- **Test 7**. To test the encrypt functionality of AES-CBC, the evaluator will supply the a 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. Key i will 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 set 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 set of key/ciphertext pairs will have 256 256-bit key/ciphertext pairs. Key i in each set will 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 will be the value that results in an all-zeros plaintext when decrypted with its corresponding key..
- **Test 8**. To test the encrypt functionality of AES-CBC, the evaluator will supply the set of 256 plaintext values described below and obtain the ciphertext values that result from AES-CBC encryption of the given plaintext using a 256-bit key value of all zeros with an IV of all zeros. Plaintext value i in each set will have the leftmost i bits be ones and the rightmost 256-i bits be zeros, for i in [1,256].

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 $1 < i \leq 10$. 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 $1 < i \leq 10$. 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 AESCBC-Decrypt.

AES-CTR Test

Known Answer Tests (KATs) There are four Known Answer Tests (KATs) described below. For all KATs, the plaintext, initialization vector (IV), and ciphertext values shall be 256-bit blocks. The results from each test may either be obtained by the validator 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.

- Test 9: To test the encrypt functionality, the evaluator will supply 5 plaintext values and obtain the ciphertext value that results from encryption of the given plaintext using a 256-bit key value of all zeros and an IV of all zeros. To test the decrypt functionality, the evaluator will perform the same test as for encrypt, using the 5 ciphertext values as input.
- Test 10: To test the encrypt functionality, the evaluator will supply 5 256-bit key values and obtain the ciphertext value that results from encryption of an all zeros plaintext using the given key value and an IV of all zeros. To test the decrypt functionality, the evaluator will perform the same test as for encrypt, using an all zero ciphertext value as input.
- Test 11: To test the encrypt functionality, the evaluator will supply a set of key values described below and obtain the ciphertext values that result from AES encryption of an all zeros plaintext using the given key values and an IV of all zeros. The set of keys shall have shall have 256 256-bit keys. Keyi shall have the leftmost i bits be ones and the rightmost 256-i bits be zeros, for i in [1, N]. To test the decrypt functionality, the evaluator will supply the set of key and ciphertext value pairs described below and obtain the plaintext value that results from decryption of the given

ciphertext using the given key values and an IV of all zeros. The set of key/ciphertext pairs shall have 256 256-bit pairs. Keyi shall have the leftmost i bits be ones and the rightmost 256-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.

- Test 12: To test the encrypt functionality, the evaluator will supply the set of 256 plaintext values described below and obtain the two ciphertext values that result from encryption of the given plaintext using a 256 bit key value of all zeros, respectively, and an IV of all zeros. Plaintext value i in each set shall have the leftmost bits be ones and the rightmost 256-i bits be zeros, for i in [1, 256]. To test the decrypt functionality, 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

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:

- 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:

- 256 bit key

- 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 13: 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 14: 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 15: 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 16: 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) Test

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

- 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).

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

Tests

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 test MACs. 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 17**: 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 pseudo-randomly 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 18**: 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 pseudo-randomly 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 19**: 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 i^{th} message is $512 + 99 \cdot i$, where $1 \leq i \leq m$. The message text shall be pseudo-randomly 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 20**: 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 i^{th} message is $512 + 8 \cdot 99 \cdot i$, where $1 \leq i \leq m/8$. The message text shall be pseudo-randomly 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 21**: Pseudo-randomly 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))[24]

Tests

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 22**: ECDSA FIPS 186-5 Signature Generation Test. For each supported NIST curve (i.e., 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 186-5 Signature Verification Test. For each supported NIST curve (i.e., 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)

[24] These assurance activities were modified as part of NIAP Technical Decision 873.

in five of the 10 tuples. The evaluator will verify that 5 responses indicate success and 5 responses indicate failure.

TSS

[Conditional: if “2048-bit (for secure boot only) or greater” is selected] The evaluator shall check that the TSS documents that 2048-bit RSA is used only for secure boot and a greater key size is used for any other functions.

Guidance

[Conditional: if “2048-bit (for secure boot only) or greater” is selected] The evaluator shall check that the AGD documents any configuration needed to ensure 2048-bit RSA is used only for secure boot and a greater key size is used for any other functions.

RSA Signature Algorithm Tests

- **Test 24**: 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 25**: 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))

Tests

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

Tests

The evaluator will perform the following tests:

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) un-instantiate. 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) un-instantiate. 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 list contains more information on some of the input values to be generated/selected by the evaluator.

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

Documentation shall be produced - and the evaluator will perform the activities - in accordance with Appendix E – Entropy Documentation and Assessment 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)

TSS

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.

Guidance

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.

5.2.2.3 User Data Protection (FDP)

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

TSS

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.

Tests

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

- **Test 26**: 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 27**: 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 28**: 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 29**: 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 30**: 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 31**: 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)

TSS

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).

Tests

The evaluator will perform the following test:

- **Test 94:**
  - **Step 1**: The evaluator will enable a network connection. The evaluator will sniff packets while performing running applications that use the network such as web browsers and email clients. The evaluator will verify that the sniffer captures the traffic generated by these actions, turn off the sniffing tool, and save the session data.
  - **Step 2**: The evaluator will configure an IPsec VPN client that supports the routing specified in this requirement. The evaluator will turn on the sniffing tool, establish the VPN connection, and perform the same actions with the device as performed in the first step. The evaluator will verify that the sniffing tool captures traffic generated by these actions, turn off the sniffing tool, and save the session data.
  - **Step 3**: The evaluator will examine the traffic from both step one and step two to verify that all non-excepted Data Plane traffic in Step 2 is encapsulated by IPsec. The evaluator will examine the Security Parameter Index (SPI) value present in the encapsulated packets captured in Step 2 from the TOE to the Gateway and shall verify this value is the same for all actions used to generate traffic through the VPN. Note that it is expected that the SPI value for packets from the Gateway to the TOE is different than the SPI value for packets from the TOE to the Gateway.
  - **Step 4**: The evaluator will perform a ping on the TOE host on the local network and verify that no packets sent are captured with the sniffer. The evaluator will attempt to send packets to the TOE outside the VPN tunnel (i.e. not through the VPN gateway), including from the local network, and verify that the TOE discards them.

5.2.2.4 Identification and Authentication (FIA)

5.2.2.4.1 Authentication Failure Handling (FIA_AFL.1)

FIA_AFL.1.1

Tests

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.

- **Test 53**: 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 54**: 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 55**: 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)

TSS

The evaluator will ensure that the TSS describes the rules as to how each authentication mechanism specified in FIA_UAU.5.1 is implemented and used. Example rules are how the authentication mechanism authenticates the user (i.e. how does the TSF verify that the correct password or authentication factor is used), 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). Rules regarding how the authentication factors interact in terms of unsuccessful authentication are covered in FIA_AFL.1.

Guidance

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

Tests

The following content should be included if:

- authentication based on username and password is selected from FIA_UAU.5.1
    - Test 56: 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 57: 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.

The following content should be included if:

- username and a PIN that releases an asymmetric key is selected from FIA_UAU.5.1

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 58: 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 59: 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.

The following content should be included if:

- combination of authentication based on user name, password, and time-based one-time password is selected from FIA_UAU.5.1

The evaluator will configure the OS to authentication to authenticate to the OS using a username, password, and one-time password mechanism. The evaluator will then perform the following tests.

Test 60: The evaluator will attempt to authenticate using a valid username, valid password, and valid one-time password. The evaluator will ensure that the authentication attempt is successful.

Test 61: The evaluator will attempt to authenticate using a valid username, invalid password, and valid one-time password. The evaluator will ensure that the authentication attempt fails.

Test 62: The evaluator will attempt to authenticate using a valid username, valid password, and invalid one-time password. The evaluator will ensure that the authentication attempt fails.

Test 63: The evaluator will attempt to authenticate using a valid username, invalid password, and invalid one-time password. The evaluator will ensure that the authentication attempt fails.

Authentication mechanisms related to authentication based on X.509 certificates are tested under FIA_X509_EXT.1 and SSH public key-based authentication are tested in the Functional Package for Secure Shell (SSH), version 1.0.

For each authentication mechanism rule, the evaluator will 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)

TSS

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.

If the OS cannot perform revocation in accordance with one of the revocation methods, the evaluator will ensure the TSS describes each revocation checking exception use case, and for each exception, the alternate functionality the TOE implements to determine the status of the certificate and disable functionality dependent on the validity of the certificate.

Tests

The tests described must be performed in conjunction with the other certificate services evaluation 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 64: The evaluator shall demonstrate that validating a certificate without a valid certification path results in the function failing, for each of the following reasons, in turn:

- by establishing a certificate path in which one of the issuing certificates is not a CA certificate,
- by omitting the basicConstraints field in one of the issuing certificates,
- by setting the basicConstraints field in an issuing certificate to have CA=False,
- by omitting the CA signing bit of the key usage field in an issuing certificate, and
- by setting the path length field of a valid CA field to a value strictly less than the certificate path.

The evaluator shall then establish a valid certificate path consisting of valid CA certificates, and demonstrate that the function succeeds. The evaluator shall then remove trust in one of the CA certificates, and show that the function fails.

- Test 65: The evaluator will demonstrate that validating an expired certificate results in the function failing.
- Test 66: The evaluator will test that the OS can properly handle revoked certificates - conditional on whether CRL, OCSP, OCSP stapling, or OCSP multi-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). If OCSP stapling per RFC 6066 is the only supported revocation method, testing revocation of the intermediate CA certificate is omitted. 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 67: If any OCSP option is selected, the evaluator shall configure the OCSP server or use a man-in-the-middle 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 shall 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 68: The evaluator shall modify any byte in the first eight bytes of the certificate and demonstrate that the certificate fails to validate. (The certificate will fail to parse correctly.)
- Test 69: The evaluator shall modify any byte in the last byte of the certificate and demonstrate that the certificate fails to validate. (The signature on the certificate will not validate.)
- Test 70: The evaluator shall modify any byte in the public key of the certificate and demonstrate that the certificate fails to validate. (The signature of the certificate will not validate.)
- Test 71: [conditional, to be performed if
    - ECDSA schemes is selected from FCS_COP.1.1/SIGN
    - 6187 is selected from FCS_SSH_EXT.1.1 from Functional Package for Secure Shell (SSH), version 1.0

    ]:

    - Test 71.1 The evaluator shall establish a valid, trusted certificate chain consisting of an EC leaf certificate, an EC Intermediate CA certificate not designated as a trust anchor, and an EC certificate designated as a trusted anchor, where the elliptic curve parameters are specified as a named curve. The evaluator shall confirm that the TOE validates the certificate chain.
    - Test 71.2: The evaluator shall replace the intermediate certificate in the certificate chain for Test 71.1 with a modified certificate, where the modified intermediate CA has a public key information field where the EC parameters uses an explicit format version of the Elliptic Curve parameters in the public key information field of the intermediate CA certificate from Test 71.1, and the modified Intermediate CA certificate is signed by the trusted EC root CA, but having no other changes. The evaluator shall confirm the TOE treats the certificate as invalid.
- Test 72 [conditional, to be performed if
    - exceptions to performing revocation are selected

]: For each exceptional use case for revocation checking described in the ST, the evaluator shall attempt to establish the conditions of the use case, designate the certificate as invalid and perform the function relying on the certificate. The evaluator shall observe that the alternate revocation checking mechanism successfully prevents performance of the function.

The evaluator will generate an X.509v3 certificate for a user with the Client Authentication Extended 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 73: 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 74: 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.

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 75**: 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 76**: 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 77**: 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)

TSS

The evaluator will 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.

Tests

- **Test 32**: 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 will 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)

Guidance

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.

Tests

The evaluator will test the OS'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.

5.2.2.6 Protection of the TSF (FPT)

5.2.2.6.1 Access Controls (FPT_ACF_EXT.1)

TSS

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.

Tests

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 33**: The evaluator will attempt to modify all kernel drivers and modules.
- **Test 34**: The evaluator will attempt to modify all security audit logs generated by the logging subsystem.
- **Test 35**: The evaluator will attempt to modify all shared libraries that are used throughout the system.
- **Test 36**: The evaluator will attempt to modify all system executables.
- **Test 37**: The evaluator will attempt to modify all system configuration files.
- **Test 38**: The evaluator will attempt to modify any additional components selected.

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 39**: The evaluator will attempt to read security audit logs generated by the auditing subsystem
- **Test 40**: The evaluator will attempt to read system-wide credential repositories
- **Test 41**: 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)

Tests

The evaluator will select 3 executables included with the TSF. If the TSF includes a web browser it must be selected. If the TSF includes a mail client it must be selected. 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. This test can also be completed on the same hardware and rebooting between application launches.

5.2.2.6.3 Limitation of Bluetooth Profile Support (FPT_BLT_EXT.1)

TSS

The evaluator will ensure that the TSS lists all Bluetooth profiles that are disabled while not in use by an application and which need explicit user action in order to become enabled.

Guidance

There are no guidance evaluation activities for this component.

The evaluator will perform the following tests:

- Test 82: The evaluator will perform this test with a test device that does not have a trust relationship with the TOE. While the service is not in active use by an application on the TOE, the evaluator will attempt to discover a service associated with a "protected" Bluetooth profile (as specified by the requirement) on the TOE via a Service Discovery Protocol search. The evaluator will verify that the service does not appear in the Service Discovery Protocol search results. Next, the evaluator shall attempt to gain remote access to the service from a device that does not currently have a trusted device relationship with the TOE. The evaluator will verify that this attempt fails due to the unavailability of the service and profile.
- Test 83: The evaluator will repeat Test 1 with a device that currently has a trusted device relationship with the TOE and verify that the same behavior is exhibited.

5.2.2.6.4 Stack Buffer Overflow Protection (FPT_SBOP_EXT.1)[25]

Tests

For stack-based OSes, the evaluator will determine that the TSS contains a description of stack-based buffer overflow protections used by the OS. hese are referred to by a variety of terms. ~~such as~~ **These include, but are not limited to, ASLR, tagging,** stack cookie, stack guard, and stack canaries. The TSS must include a rationale for any binaries that are not protected in this manner. The evaluator will also preform the following test:

- **Test 42 (Conditional: stack-based overflow protection can be determined by inventorying)**: 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.

For OSes that store parameters/variables separately from control flow values, the evaluator will verify that the TSS describes what data structures control values, parameters, and variables are stored. The evaluator will also ensure that the TSS includes a description of the safeguards that ensure parameters and variables do not intermix with control flow values.

[25] This assurance activity was modified as part of NIAP Technical Decision 906.

5.2.2.6.5 Software Restriction Policies (FPT_SRP_EXT.1)

TSS

The evaluator will ensure that the description of the supported characteristics in the TSS is consistent with the SFR. The evaluator will also ensure that any characteristics specified by the ST-author are described in sufficient detail to understand how to test those characteristics.

Guidance

The evaluator will ensure that that the characteristics are described in sufficient detail for administrators to configure policies using them, and that the list of characteristics in the guidance is consistent with the information in the TSS.

Tests

There are two tests for each selection above.

There are two tests for each selection above.

- Test 84[conditional, to be performed if
    - file path is selected from FPT_SRP_EXT.1.1

]: 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 85[conditional, to be performed if
    - file path is selected from FPT_SRP_EXT.1.1

]: 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 not in the allowed list. The evaluator will ensure that the code they attempted to execute has not been executed.

- Test 86[conditional, to be performed if
    - file digital signature is selected from FPT_SRP_EXT.1.1

]: 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 87[conditional, to be performed if
    - file digital signature is selected from FPT_SRP_EXT.1.1

]: 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 88[conditional, to be performed if
    - version is selected from FPT_SRP_EXT.1.1

]: 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 89[conditional, to be performed if
    - version is selected from FPT_SRP_EXT.1.1

]: 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 90[conditional, to be performed if
    - hash is selected from FPT_SRP_EXT.1.1

]: 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 91[conditional, to be performed if
    - hash is selected from FPT_SRP_EXT.1.1

]: 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.

- Test 92[conditional, to be performed if
    - other is selected from FPT_SRP_EXT.1.1

]: The evaluator will attempt to run an application that should be allowed based on the defined software restriction policy and ensure that it runs.

- Test 93[conditional, to be performed if
    - other is selected from FPT_SRP_EXT.1.1

    ]: The evaluator will then attempt to run an application that should not be allowed the defined software restriction policy and ensure that it does not run.

- **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.

5.2.2.6.6 Boot Integrity (FPT_TST_EXT.1)

TSS

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 43**: 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 44**: 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 45** [conditional, to be performed
    - if a digital signature using an X509 certificate with hardware-based protection is selected from FPT_TST_EXT.1.1]**:**

If the ST author indicates that the integrity verification is performed using a public key in an X509 certificate, the evaluator will verify that the update boot integrity mechanism includes a certificate validation according to FIA_X509_EXT.1 for all certificates in the chain from the certificate used for boot integrity to a certificate in the trust store that are not themselves in the trust store. This means that, for each X509 certificate in this chain that is not a trust store element, the evaluator must ensure that revocation information is available to the TOE during the bootstrap mechanism (before the TOE becomes fully operational)[26].

5.2.2.6.7 Trusted Update (FPT_TUD_EXT.1)

Tests

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 the response to this query is authentic by using a digital signature scheme specified in FCS_COP.1/SIGN). The digital signature verification may be performed as part of a network protocol as described in FTP_ITC_EXT.1. If the signature verification is not performed as part of a trusted channel, the evaluator shall send a query response with a bad signature and verify that the signature verification fails. The evaluator shall then send a query response with a good signature and verify that the signature verification is successful.

[26] This protection profile evaluation activity was replaced as part of NIAP Technical Decision 493.

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 46**: 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 47**: 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.

5.2.2.6.8 Trusted Update for Application Software (FPT_TUD_EXT.2)

Tests

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 the response to this query is authentic by using a digital signature scheme specified in FCS_COP.1/SIGN). The digital signature verification may be performed as part of a network protocol as described in FTP_ITC_EXT.1. If the signature verification is not performed as part of a trusted channel, the evaluator shall send a query response with a bad signature and verify that the signature verification fails. The evaluator shall then send a query response with a good signature and verify that the signature verification is successful.

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 48**: 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 49**: 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)

Tests

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)

Tests

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)

TSS

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.

Guidance

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 78**: 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 79**: 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 80**: The evaluator will ensure, for each method of remote administration, the channel data is not sent in plaintext.
- **Test 81**: The evaluator will ensure, for each method of remote administration, modification of the channel data is detected by the OS.

5.2.3 WLAN Client Module Assurance Activities

This section copies the assurance activities from the WLAN Client PP-Module in order to ease reading and comparisons between the extended package and the security target.

5.2.3.1 Security Audit (FAU)

5.2.3.1.1 Audit Data Generation for Wireless LAN (FAU_GEN.1 (WLAN))

TSS

The evaluator shall check the TSS and ensure it provides a format for audit records. Each audit record format type must be covered, along with a brief description of each field.

If "invoke platform-provided functionality" is selected, the evaluator shall examine the TSS to verify it describes (for each supported platform) how this functionality is invoked (it should be noted that this may be through a mechanism that is not implemented by the WLAN Client; however, that mechanism will be identified in the TSS as part of this evaluation activity).

Guidance

The evaluator shall check the operational guidance and ensure it lists all of the auditable events and 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 check to make sure that every audit event type mandated by the PP-Module is described and that the description of the fields contains the information required in FAU_GEN.1.2/WLAN, and the additional information specified in Table 2 in the main document and Table 5 in the main document.

The evaluator shall in particular ensure that the operational guidance is clear in relation to the contents for failed cryptographic events. In the Auditable Events tables, information detailing the cryptographic mode of operation and a name or identifier for the object being encrypted is required. The evaluator shall ensure that name or identifier is sufficient to allow an administrator reviewing the audit log to determine the context of the cryptographic operation (for example, performed during a key negotiation exchange, performed when encrypting data for transit) as well as the non-TOE endpoint of the connection for cryptographic failures relating to communications with other IT systems.

The evaluator shall also make a determination of the administrative actions that are relevant in the context of this PP-Module. The TOE may contain functionality that is not evaluated in the context of this PP-Module because the functionality is not specified in an SFR. This functionality may have administrative aspects that are described in the operational guidance. Since such administrative actions will not be performed in an evaluated configuration of the TOE, the evaluator shall examine the operational guidance and make a determination of which administrative commands, including subcommands, scripts, and configuration files, are related to the configuration (including enabling or disabling) of the mechanisms implemented in the TOE that are necessary to enforce the requirements specified in the PP-Module, which thus form the set of “all administrative actions”. The evaluator may perform this activity as part of the activities associated with ensuring the AGD_OPE guidance satisfies the requirements.

Tests

The evaluator will test the TOE’s ability to correctly generate audit records by having the TOE generate audit records in accordance with the evaluation activities associated with the functional requirements in this PP-Module. When verifying the test results, the evaluator will 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.

Note that the testing here can be accomplished in conjunction with the testing of the security mechanisms directly. For example, testing performed to ensure that the administrative guidance provided is correct verifies that AGD_OPE.1 is satisfied and should address the invocation of the administrative actions that are needed to verify the audit records are generated as expected.

5.2.3.2 Cryptographic Support (FCS)

5.2.3.2.1 Cryptographic Key Generation (Symmetric Keys for WPA2/WPA3 Connections) (FCS_CKM.1(WPA))

TSS

The evaluator shall verify that the TSS describes how the primitives defined and implemented by this PP-Module are used by the TOE in establishing and maintaining secure connectivity to the wireless clients. The TSS shall also provide a description of the developer's method(s) of assuring that their implementation conforms to the cryptographic standards; this includes not only testing done by the developing organization, but also any third-party testing that is performed.

Guidance

There are no guidance evaluation activities for this component.

Tests

The evaluator shall perform the following tests:

- **Test 1**: The evaluator shall configure the access point so the cryptoperiod of the session key is 1 hour. The evaluator shall successfully connect the TOE to the access point and maintain the connection for a length of time that is greater than the configured cryptoperiod. The evaluator shall use a packet capture tool to determine that after the configured cryptoperiod, a re-negotiation is initiated to establish a new session key. Finally, the evaluator shall determine that the renegotiation has been successful and the client continues communication with the access point.
- **Test 2**: The evaluator shall perform the following test using a packet sniffing tool to collect frames between the TOE and a wireless LAN access point:

Step 1: The evaluator shall configure the access point to an unused channel and configure the WLAN sniffer to sniff only on that channel (i.e., lock the sniffer on the selected channel). The sniffer should also be configured to filter on the MAC address of the TOE and/or access point.

Step 2: The evaluator shall configure the TOE to communicate with a WLAN access point using IEEE 802.11-2012 and a 256-bit (64 hex values 0-f) pre-shared key. The pre-shared key is only used for testing.

Step 3: The evaluator shall start the sniffing tool, initiate a connection between the TOE and the access point, and allow the TOE to authenticate, associate, and successfully complete the 4-way handshake with the client.

Step 4: The evaluator shall set a timer for 1 minute, at the end of which the evaluator shall disconnect the TOE from the wireless network and stop the sniffer.

Step 5: The evaluator shall identify the 4-way handshake frames (denoted EAPOL-key in Wireshark captures) and derive the PTK from the 4-way handshake frames and pre-shared key as specified in IEEE 802.11-2012.

Step 6: The evaluator shall select the first data frame from the captured packets that was sent between the TOE and access point after the 4-way handshake successfully completed, and without the frame control value 0x4208 (the first 2 bytes are 08 42). The evaluator shall use the PTK to decrypt the data portion of the packet as specified in IEEE 802.11-2012, and shall verify that the decrypted data contains ASCII-readable text.

Step 7: The evaluator shall repeat Step 6 for the next 2 data frames between the TOE and access point and without frame control value 0x4208.

5.2.3.2.2 Cryptographic Key Distribution for Group Temporal Key (GTK) (FCS_CKM.2(WLAN))

Application Note: FCS_CKM.2(WLAN) corresponds to FCS_CKM.2/WLAN in the WLAN Client module.

TSS

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

Guidance

There are no guidance evaluation activities for this component.

Tests

The evaluator shall perform the following test using a packet sniffing tool to collect frames between the TOE and a wireless access point (which may be performed in conjunction with the assurance activity for FCS_CKM.1.1/WLAN).

Step 1: The evaluator shall configure the access point to an unused channel and configure the WLAN sniffer to sniff only on that channel (i.e., lock the sniffer on the selected channel). The sniffer should also be configured to filter on the MAC address of the TOE and/or access point.

Step 2: The evaluator shall configure the TOE to communicate with the access point using IEEE 802.11-2012 and a 256-bit (64 hex values 0-f) pre-shared key, setting up the connections as described in the operational guidance. The pre-shared key is only used for testing.

Step 3: The evaluator shall start the sniffing tool, initiate a connection between the TOE and access point, and allow the TOE to authenticate, associate, and successfully complete the 4-way handshake with the TOE.

Step 4: The evaluator shall set a timer for 1 minute, at the end of which the evaluator shall disconnect the TOE from the access point and stop the sniffer.

Step 5: The evaluator shall identify the 4-way handshake frames (denoted EAPOL-key in Wireshark captures) and derive the PTK and GTK from the 4-way handshake frames and pre-shared key as specified in IEEE 802.11-2012.

Step 6: The evaluator shall select the first data frame from the captured packets that was sent between the TOE and access point after the 4-way handshake successfully completed, and with the frame control value 0x4208 (the first 2 bytes are 08 42). The evaluator shall use the GTK to decrypt the data portion of the selected packet as specified in IEEE 802.11-2012, and shall verify that the decrypted data contains ASCIIreadable text.

Step 7: The evaluator shall repeat Step 6 for the next 2 data frames with frame control value 0x4208.

To fully test the broadcast and multicast functionality, these steps will be performed as the evaluator connects multiple clients to the TOE. The evaluator will ensure that GTKs established are sent to the appropriate participating clients.

5.2.3.2.3 Extended: Extensible Authentication Protocol-Transport Layer Security (FCS_TLSC_EXT.1(WLAN))

Application Note: FCS_TLSC_EXT.1(WLAN) corresponds to FCS_TLSC _EXT.1/WLAN in the WLAN CLIENT EP.

TSS

The evaluator shall check the description of the implementation of this protocol in the TSS to ensure that the ciphersuites supported are specified. The evaluator shall check the TSS to ensure that the ciphersuites specified include those listed for this component.

Guidance

The evaluator shall also check the operational guidance to ensure that it contains instructions on configuring the TOE so that TLS conforms to the description in the TSS (for instance, the set of ciphersuites advertised by the TOE may have to be restricted to meet the requirements).

The evaluator shall check that the guidance contains instructions for the administrator to configure the list of Certificate Authorities that are allowed to sign certificates used by the authentication server that will be accepted by the TOE in the EAP-TLS exchange, and instructions on how to specify the algorithm suites that will be proposed and accepted by the TOE during the EAP-TLS exchange.

Tests

The evaluator shall write, or the TOE developer shall provide, an application for the purposes of testing TLS.

The evaluator shall also perform the following tests:

- **Test 1**: The evaluator shall 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 shall attempt to establish the connection using a server with a server certificate that contains the Server Authentication purpose in the extendedKeyUsage field and

verify that a connection is established. The evaluator will then verify that the client rejects an otherwise valid server certificate that lacks the Server Authentication purpose in the extendedKeyUsage field and a connection is not established. Ideally, the two certificates should be identical except for the extendedKeyUsage field.

- **Test 3**: The evaluator shall send a server certificate in the TLS connection that does not match the server-selected 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 shall verify that the TOE disconnects after receiving the server's Certificate handshake message.
- **Test 4**: The evaluator shall 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 shall perform the following modifications to the traffic:
    - Change the TLS version selected by the server in the Server Hello to a unsupported TLS version (for example 1.5 represented by the two bytes 03 06) and verify that the client rejects the connection.
    - 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.
    - 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 shall verify that the client rejects the connection after receiving the Server Hello.
    - [conditional: if the TOE supports at least one cipher suite that uses DHE or ECDHE for key exchange] 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. This test does not apply to cipher suites using RSA key exchange.
    - Modify a byte in the Server Finished handshake message, and verify that the client sends an Encrypted Message followed by a FIN and ACK message. This is sufficient to deduce that the TOE responded with a Fatal Alert and no further data would be sent.
    - Send a garbled message from the server after the server has issued the ChangeCipherSpec message and verify that the client denies the connection..

5.2.3.2.4 TLS Client Support for Supported Groups Extension (EAP-TLS for WLAN) (FCS_TLSC_EXT.1(WLAN))

TSS

The evaluator shall verify that the TSS describes the Supported Groups extension and whether the required behavior is performed by default or may be configured.

Guidance

If the TSS indicates that the Supported Groups extension must be configured to meet the requirement, the evaluator shall verify that the operational guidance includes instructions for configuration of this extension.

Tests

The evaluator shall perform the following test:

- **Test 1**: 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.

5.2.3.2.5 Supported WPA Versions (FCS_WPA_EXT.1) [27]

TSS

There are no TSS evaluation activities for this component.

Guidance

The evaluator shall ensure that the AGD contains guidance on how to configure the WLAN client to connect to networks supporting WPA3 and, if selected, WPA2.

Tests

The evaluator shall configure a Wi-Fi network that utilizes WPA3 and verify that the client can connect. The same test shall be repeated for WPA2 if it is selected.

5.2.3.3 Identification and Authentication (FIA)

5.2.3.3.1 Extended: Port Access Entity Authentication (FIA_PAE_EXT.1)

TSS

There are no TSS evaluation activities for this component.

Guidance

There are no guidance evaluation activities for this component.

Tests

The evaluator shall perform the following tests:

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

[27] This protection profile assurance activity was modified as part of NIAP Technical Decision 710.

- **Test 3**: The evaluator shall demonstrate that the TOE has no access to the test network. The evaluator shall attempt to authenticate using an invalid authentication server certificate, such that the EAP-TLS negotiation fails. This should result in the TOE still being unable to access the test network.

5.2.3.3.2 X.509 Certificate Validation (FIA_X509_EXT.1(WLAN))

Application Note: FIA_X509_EXT.1(WLAN) corresponds to FIA_X509_EXT.1/WLAN in the WLAN Client module.

TSS

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

Guidance

There are no guidance evaluation activities for this component.

Tests

The tests described must be performed in conjunction with the other Certificate Services assurance activities. The tests for the extendedKeyUsage rules are performed in conjunction with the uses that require those rules. The evaluator shall 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 shall then load a certificate or certificates to the Trust Anchor Database needed to validate the certificate to be used in the function (e.g. application validation), and demonstrate that the function succeeds. The evaluator then shall delete one of the certificates, and show that the function fails.
- **Test 2**: The evaluator shall demonstrate that validating an expired certificate results in the function failing.
- **Test 3**: The evaluator shall construct a certificate path, such that the certificate of the CA issuing the TOE's certificate does not contain the basicConstraints extension. The validation of the certificate path fails.
- **Test 4**: The evaluator shall construct a certificate path, such that the certificate of the CA issuing the TOE's certificate has the cA flag in the basicConstraints extension not set. The validation of the certificate path fails.
- **Test 5**: The evaluator shall modify any byte in the first eight bytes of the certificate and demonstrate that the certificate fails to validate (the certificate will fail to parse correctly).
- **Test 6**: The evaluator shall modify any bit in the last byte of the signature algorithm of the certificate and demonstrate that the certificate fails to validate (the signature on the certificate will not validate).
- **Test 7**: The evaluator shall modify any byte in the public key of the certificate and demonstrate that the certificate fails to validate (the signature on the certificate will not validate).

5.2.3.3.3 X.509 Certificate Authentication EAP-TLS for WLAN (FIA_X509_EXT.2(WLAN))[28]

Application Note: FIA_X509_EXT.2(WLAN) corresponds to FIA_X509_EXT.2/WLAN in the WLAN Client module.

TSS

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

Guidance

If not already present in the TSS, the evaluator shall check the administrative guidance to ensure that it describes how the TOE chooses which certificates to use, and any necessary instructions for configuring the operating environment so that the TOE can use the certificates.

Tests

The evaluator shall perform the following test:

- **Test 1**: The evaluator shall demonstrate using a valid certificate that requires certificate validation checking to be performed in at least some part by communicating with a non-TOE IT entity. The evaluator shall then manipulate the environment so that the TOE is unable to verify the validity of the certificate, and observe that the action selected in FIA_X509_EXT.2.2 is performed. If the selected action is administrator-configurable, then the evaluator shall follow the operational guidance to determine that all supported administrator-configurable options behave in their documented manner.

5.2.3.3.4 Certificate Storage and Management (FIA_X509_EXT.4)

TSS

The evaluator shall examine the TSS to determine that it describes all certificate stores implemented that contain certificates used to meet the requirements of this PP-Module. This description shall contain information pertaining to how certificates are loaded into the store, and how the store is protected from unauthorized access.

Guidance

The evaluator shall check the administrative guidance to ensure that it describes how to load X.509 certificates into the TOE's certificate store, regardless of whether the TSF provides this mechanism itself or the TOE relies on a platform-provided mechanism for this.

Tests

The evaluator shall perform the following test for each TOE function that requires the use of certificates:

- **Test 1**: The evaluator shall demonstrate that using a certificate without a valid certification path results in the function failing. The evaluator shall then load any certificates needed to validate

[28] This protection profile assurance activity was modified as part of NIAP Technical Decision 703.

the certificate to be used in the function and demonstrate that the function succeeds. The evaluator shall then delete one of these dependent certificates and show that the function fails.

- **Test 2**: The evaluator shall demonstrate that the mechanism used to load or configure X.509 certificates cannot be accessed without appropriate authorization.

5.2.3.4 Security Management (FMT)

5.2.3.4.1 Specification of Management Functions for Wi-Fi (FMT_SMF.1(WLAN))

Application Note: FMT_SMF.1(WLAN) corresponds to FMT_SMF.1/WLAN in the WLAN Client module.

TSS

There are no TSS assurance activities for this SFR.

Guidance

The evaluator shall check to verifythat every management function claimed by the TOE is described there. The evaluator shall also verify that these descriptions include the information required to perform the management duties associated with the function.

Tests

The evaluator shall test the TOE's ability to provide the management functions by configuring the TOE and performing the management activities associated with each function claimed in the SFR.

Note that this may be accomplished in conjunction with the testing of other requirements, such as FCS_TLSC_EXT.1/WLAN and FTA_WSE_EXT.1.

5.2.3.5 Protection of the TSF (FPT)

5.2.3.5.1 TSF Cryptographic Functionality Testing (FPT_TST_EXT.3 (WLAN))

Application Note: FPT_TST_EXT.3(WLAN) corresponds to FPT_TST_EXT.3/WLAN in the WLAN Client module.

TSS

The evaluator shall examine the TSS to ensure that it details the self tests that are run by the TSF on start-up; this description should include an outline of what the tests are actually doing (e.g., rather than saying "memory is tested", a description similar to "memory is tested by writing a value to each memory location and reading it back to ensure it is identical to what was written" shall be used). The evaluator shall ensure that the TSS makes an argument that the tests are sufficient to demonstrate that the TSF is operating correctly.

The evaluator shall examine the TSS to ensure that it describes how to verify the integrity of stored TSF executable code when it is loaded for execution. The evaluator shall ensure that the TSS makes an argument that the tests are sufficient to demonstrate that the integrity of stored TSF executable code has not been compromised. The evaluator also ensures that the TSS (or the operational guidance) describes the actions that take place for successful (e.g. hash verified) and unsuccessful (e.g., hash not verified) cases.

Guidance

The evaluator shall ensure that the operational guidance describes the actions that take place for successful (e.g. hash verified) and unsuccessful (e.g., hash not verified) cases.

Tests

The evaluator shall perform the following tests:

- **Test 1**: The evaluator performs the integrity check on a known good TSF executable and verifies that the check is successful.
- **Test 2**: The evaluator modifies the TSF executable, performs the integrity check on the modified TSF executable and verifies that the check fails.

5.2.3.6 TOE Access (FTA)

5.2.3.6.1 Wireless Network Access (FTA_WSE_EXT.1)

TSS

The evaluator shall examine the TSS to determine that it defines SSIDs as the attribute to specify acceptable networks.

Guidance

The evaluator shall examine the operational guidance to determine that it contains guidance for configuring the list of SSID that the WLAN Client is able to connect to.

Tests

The evaluator shall also perform the following tests for each attribute:

- **Test 1**: The evaluator configures the TOE to allow a connection to a wireless network with a specific SSID. The evaluator also configures the test environment such that the allowed SSID and an SSID that is not allowed are both “visible” to the TOE. The evaluator shall demonstrate that they can successfully establish a session with the allowed SSID. The evaluator will then attempt to establish a session with the disallowed SSID and observe that the attempt fails.

5.2.3.7 Trusted Path / Channels (FTP)

5.2.3.7.1 Trusted Channel Communication (FTP_ITC.1 (WLAN))

Application Note: FTP_ITC_EXT.1(WLAN) corresponds to FTP_ITC _EXT.1/WLAN in the WLAN Client module.

TSS

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

Guidance

The evaluator shall confirm that the operational guidance contains instructions for establishing the connection to the access point and that it contains recovery instructions should a connection be unintentionally broken.

Tests

The evaluator shall perform the following tests:

- Test 1: The evaluators shall ensure that the TOE is able to initiate communications with an access point using the protocols specified in the requirement by setting up the connections as described in the operational guidance and ensuring that communication is successful.
- Test 2: The evaluator shall ensure, for each communication channel with an authorized IT entity, the channel data is not sent in plaintext.
- Test 3: The evaluator shall ensure, for each communication channel with an authorized IT entity, modification of the channel data is detected by the TOE.
- Test 4: The evaluators shall physically interrupt the connection from the TOE to the access point (e.g., moving the TOE host out of range of the access point, turning the access point off). The evaluators shall ensure that subsequent communications are appropriately protected, at a minimum in the case of any attempts to automatically resume the connection or connect to a new access point.

Further assurance activities are associated with the specific protocols.

5.2.4 VPN Client Module Assurance Activities

This section copies the assurance activities from the VPN Client PP-Module in order to ease reading and comparisons between the extended package and the security target.

5.2.4.1 Security Audit (FAU)

5.2.4.1.1 Audit Data Generation (FAU_GEN.1(VPN))

Application Note: FAU_GEN.1(VPN) corresponds to FAU_GEN.1 in the IPsec extended package.

TSS

The evaluator shall examine the TSS to determine that it describes the auditable events and the component that is responsible for each type of auditable event.

Guidance

The evaluator shall check the operational guidance and ensure that it lists all of the auditable events and 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 check to make sure that every audit event type mandated by the PP-Module is described and that the description of the fields contains the information required in FAU_GEN.1.2, and the additional information specified in Table C-1 of the PP-Module.

In particular, the evaluator shall ensure that the operational guidance is clear in relation to the contents for failed cryptographic events. In the Auditable Events table of the VPN Client PP-Module, information detailing the cryptographic mode of operation and a name or identifier for the object being encrypted is

required. The evaluator shall ensure that name or identifier is sufficient to allow an administrator reviewing the audit log to determine the context of the cryptographic operation (for example, performed during a key negotiation exchange, performed when encrypting data for transit) as well as the non-TOE endpoint of the connection for cryptographic failures relating to communications with other IT systems.

The evaluator shall also make a determination of the administrative actions that are relevant in the context of the VPN Client PP-Module. The TOE may contain functionality that is not evaluated in the context of the VPN Client PP-Module because the functionality is not specified in an SFR. This functionality may have administrative aspects that are described in the operational guidance. Since such administrative actions will not be performed in an evaluated configuration of the TOE, the evaluator shall examine the operational guidance and make a determination of which administrative commands, including subcommands, scripts, and configuration files, are related to the configuration (including enabling or disabling) of the mechanisms implemented in the TOE that are necessary to enforce the requirements specified in the VPN Client PP- Module, which thus form the set of “all administrative actions”. The evaluator may perform this activity as part of the activities associated with ensuring the AGD_OPE guidance satisfies the requirements.

For each required auditable event, the evaluator shall examine the operational guidance to determine that it is clear to the reader where each event is generated (e.g. the TSF may generate its own audit logs in one location while the platform-provided auditable events are generated elsewhere).

Tests

The evaluator shall test the TOE’s ability to correctly generate audit records by having the TOE generate audit records in accordance with the Assurance Activities associated with the functional requirements in this PP-Module. Additionally, the evaluator shall test that each administrative action applicable in the context of this PP-Module is auditable. 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.

Note that the testing here can be accomplished in conjunction with the testing of the security mechanisms directly. For example, testing performed to ensure that the administrative guidance provided is correct verifies that AGD_OPE.1 is satisfied and should address the invocation of the administrative actions that are needed to verify the audit records are generated as expected.

5.2.4.1.2 Selective Audit (FAU_SEL.1)

TSS

There are no TSS Assurance Activities for this SFR.

Guidance

The evaluator shall review the administrative guidance to ensure that the guidance itemizes all event types, as well as describes all attributes that are to be selectable in accordance with the requirement, to include those attributes listed in the assignment. The administrative guidance shall also contain instructions on how to set the pre-selection, or how the VPN gateway will configure the client, as well as explain the syntax (if present) for multi-value pre-selection. The administrative guidance shall also

identify those audit records that are always recorded, regardless of the selection criteria currently being enforced.

Tests

The evaluator shall perform the following tests:

- **Test 1**: For each attribute listed in the requirement, the evaluator shall devise a test to show that selecting the attribute causes only audit events with that attribute (or those that are always recorded, as identified in the administrative guidance) to be recorded.
- **Test 2**: [conditional] If the TSF supports specification of more complex audit pre-selection criteria (e.g., multiple attributes, logical expressions using attributes) then the evaluator shall devise tests showing that this capability is correctly implemented. The evaluator shall also, in the test plan, provide a short narrative justifying the set of tests as representative and sufficient to exercise the capability.

5.2.4.2 Cryptographic Support (FCS)

5.2.4.2.1 Cryptographic Key Generation (FCS_CKM.1 (VPN))

Application Note: FCS_CKM.1(VPN) corresponds to FCS_CKM.1/VPN in the IPsec extended package.

TSS

The evaluator shall examine the TSS to verify that it describes how the key generation functionality is invoked.

Guidance

There are no AGD Assurance Activities for this requirement.

Tests

If this functionality is implemented by the TSF, refer to the following EAs, depending on the TOE's claimed Base-PP:

- GPOS PP: FCS_CKM.1

5.2.4.2.2 Cryptographic Key Storage (FCS_CKM_EXT.2)

TSS

Regardless of whether this requirement is met by the VPN client or the OS, the evaluator will check the TSS to ensure that it lists each persistent secret (credential, secret key) and private key needed to meet the requirements in the ST. For each of these items, the evaluator will confirm that the TSS lists for what purpose it is used, and how it is stored.

The evaluator shall review the TSS ~~for~~ to determine that it makes a case that, for each item listed as being manipulated by the VPN client, it is not written unencrypted to persistent memory, and that the item is stored by the OS.

Guidance

There are no AGD Assurance Activities for this requirement.

Tests

There are no test Assurance Activities for this requirement.

5.2.4.2.3 EAP-TLS (FCS_EAP_EXT.1)

TSS

The evaluator shall verify that the TS describes the use of EAP options for each of the selected peer authentication mechanisms, that TLS with mutual authentication is used, that the random values are from an appropriate source, and that the EAP MSK is derived from the TLS master key and is used as the IKEv2 shared key.

Guidance

The evaluator shall verify that the guidance documents describe any configurable features of the EAP or TLS functionality, including instructions for configuration of the authenticators and registration processes for clients.

Tests

Testing for TLS functionality is in accordance with the TLS package. For each supported EAP method claimed in FCS_EAP_TLS_EXT.1.1 and for each authentication method claimed in FCS_EAP_TLS_EXT.1.3, the evaluator shall perform the following tests:

- **Test 1**: The evaluator shall follow AGD guidance to configure the TSF to use the EAP method claimed. The evaluator shall follow AGD guidance to configure the TSF to use the authentication method claimed and, for EAP-TTLS, register a client with the appropriate key material required for the authentication method. The evaluator shall establish an VPN session using a test client with a valid certificate and, for EAP-TTLS, configured to provide a correct value for the configured authenticator. The evaluator shall observe the the VPN session is successful.

- **Test 2**: (conditional for EAP-TTLS support): The evaluator shall cause the test client with a valid certificate to send an invalid authenticator for the claimed authentication method: For HOTP, replay the HOTP value sent previously, For TOTP or PSK, modify a byte of the properly constructed value,and observe that the TSF aborts the session.

- **Test 3**: The evaluator shall establish a new, valid certificate for a test client using an identifier not corresponding to a registered user. For EAP-TTLS, the evaluator shall cause the test client using this certificate to send a correct authenticator value for the registered user. The evaluator shall initiate a VPN session from the test client to the TSF and observe that the TSF aborts the session.
- **Test 4**: The evaluator shall follow AGD guidance to configure the TSF to use a supported EAP method and register the user with the key material required for a supported authentication method. The evaluator shall configure a test client to respond to an IKEv2 exchange with EAP-request, providing valid phase 1 handshake and valid TLS handshake, but computing the phase 2

shared key using standard (nonEAP) methods. The evaluator shall initiate a VPN session between the test client and the TSF, and observe that the TSF aborts the session.

5.2.4.2.4 IPsec (FCS_IPSEC_EXT.1)

FCS_IPSEC_EXT.1.1

TSS

The evaluator shall examine the TSS and determine that it describes how the IPsec capabilities are implemented.

If the TOE is a standalone software application, the evaluator shall ensure that the TSS asserts that all IPsec functionality is implemented by the TSF. The evaluator shall also ensure that the TSS identifies what platform functionality the TSF relies upon to support its IPsec implementation, if any (e.g. does it invoke cryptographic primitive functions from the platform's cryptographic library, enforcement of packet routing decisions by low-level network drivers).

If the TOE is part of a general-purpose desktop or mobile OS, the evaluator shall ensure that the TSS describes at a high level the architectural relationship between the VPN client portion of the TOE and the rest of the TOE (e.g. is the VPN client an integrated part of the OS or is it a standalone executable that is bundled into the OS package). If the SPD is implemented by the underlying platform in this case, then the TSS describes how the client interacts with the platform to establish and populate the SPD, including the identification of the platform's interfaces that are used by the client.

In all cases, the evaluator shall also ensure that the TSS describes how the client interacts with the network stack of the platforms on which it can run (e.g., does the client insert itself within the stack via kernel mods, does the client simply invoke APIs to gain access to network services).

The evaluator shall ensure that the TSS describes how the SPD is implemented and the rules for processing both inbound and outbound packets in terms of the IPsec policy. The TSS describes the rules that are available and the resulting actions available after matching a rule.

The TSS describes how the available rules and actions form the SPD using terms defined in RFC 4301 such as BYPASS (e.g., no encryption), DISCARD (e.g., drop the packet), and PROTECT (e.g., encrypt the packet) actions defined in RFC 4301. As noted in section 4.4.1 of RFC 4301, the processing of entries in the SPD is non-trivial and the evaluator shall determine that the description in the TSS is sufficient to determine which rules will be applied given the rule structure implemented by the TOE. For example, if the TOE allows specification of ranges, conditional rules, etc., the evaluator shall determine that the description of rule processing (for both inbound and outbound packets) is sufficient to determine the action that will be applied, especially in the case where two different rules may apply. This description shall cover both the initial packets (that is, no SA is established on the interface or for that particular packet) as well as packets that are part of an established SA.

Guidance

The evaluator shall examine the operational guidance to verify it describes how the SPD is created and configured. If there is an administrative interface to the client, then the guidance describes how the administrator specifies rules for processing a packet. The description includes all three cases - a rule that ensures packets are encrypted/decrypted, dropped, and allowing a packet to flow in plaintext. The

evaluator shall determine that the description in the operational guidance is consistent with the description in the TSS, and that the level of detail in the operational guidance is sufficient to allow the administrator to set up the SPD in an unambiguous fashion. This includes a discussion of how ordering of rules impacts the processing of an IP packet.

If the client is configured by an external application, such as the VPN gateway, then the operational guidance should indicate this and provide a description of how the client is configured by the external applicationThe description should contain information as to how the SPD is established and set up in an unambiguous fashion. The description should also include what is configurable via the external application, how ordering of entries may be expressed, as well as the impacts that ordering of entries may have on the packet processing.

In either case, the evaluator ensures the description provided In the TSS is consistent with the capabilities and description provided in the operational guidance.

Tests

Depending on the implementation, the evaluator may be required to use a VPN gateway or some form of application to configure the client and platform. For Test 2, the evaluator is required to choose an application that allows for the configuration of the full set of capabilities of the VPN client (in conjunction with the platform). For example, if the client provides a robust interface that allows for specification of wildcards, subnets, etc., it is unacceptable for the evaluator to choose a VPN Gateway that only allows for specifying a single fully qualified IP addresses in the rule.

The evaluator shall perform the following tests:

- **Test 1**: The evaluator shall configure an SPD on the client that is capable of the following: dropping a packet, encrypting a packet, and allowing a packet to flow in plaintext. The selectors used in the construction of the rule shall be different such that the evaluator can generate a packet and send packets to the client with the appropriate fields (fields that are used by the rule - e.g., the IP addresses, TCP/UDP ports) in the packet header. The evaluator performs both positive and negative test cases for each type of rule. The evaluator observes via the audit trail, and packet captures that the TOE exhibited the expected behavior: appropriate packets were dropped, allowed through without modification, was encrypted by the IPsec implementation.
- **Test 2**: The evaluator shall devise several tests that cover a variety of scenarios for packet processing. These scenarios must exercise the range of possibilities for SPD entries and processing modes as outlined in the TSS and operational guidance. Potential areas to cover include rules with overlapping ranges and conflicting entries, inbound and outbound packets, and packets that establish SAs as well as packets that belong to established SAs. The evaluator shall verify, via the audit trail and packet captures, for each scenario that the expected behavior is exhibited, and is consistent with both the TSS and the operational guidance..

FCS_IPSEC_EXT.1.2

TSS

The evaluator shall check the TSS to ensure it states that the VPN can be established to operate in tunnel mode and/or transport mode (as selected).

Guidance

The evaluator shall confirm that the operational guidance contains instructions on how to configure the connection in each mode selected.

If both transport mode and tunnel mode are implemented, the evaluator shall review the operational guidance to determine how the use of a given mode is specified.

Tests

The evaluator shall perform the following test(s) based on the selections chosen:

- **Test 1**: [conditional] If tunnel mode is selected, the evaluator uses the operational guidance to configure the TOE to operate in tunnel mode and also configures a VPN gateway to operate in tunnel mode. The evaluator configures the TOE and the VPN gateway to use any of the allowable cryptographic algorithms, authentication methods, etc. to ensure an allowable SA can be negotiated. The evaluator shall then initiate a connection from the client to connect to the VPN GW peer. The evaluator observes (for example, in the audit trail and the captured packets) that a successful connection was established using the tunnel mode.
- **Test 2**: [conditional] : If transport mode is selected, the evaluator uses the operational guidance to configure the TOE to operate in transport mode and also configures an IPsec peer to accept IPsec connections using transport mode. The evaluator configures the TOE and the endpoint device to use any of the allowed cryptographic algorithms, authentication methods, etc. to ensure an allowable SA can be negotiated. The evaluator then initiates a connection from the TOE to connect to the remote endpoint. The evaluator observes (for example, in the audit trail and the captured packets) that a successful connection was established using the transport mode.
- **Test 3**: [conditional] If both tunnel mode and transport mode are selected, the evaluator shall perform both Test 1 and Test 2 above, demonstrating that the TOE can be configured to support both modes.
- **Test 4**: [conditional] If both tunnel mode and transport mode are selected, the evaluator shall modify the testing for FCS_IPSEC_EXT.1 to include the supported mode for SPD PROTECT entries to show that they only apply to traffic that is transmitted or received using the indicated mode.

FCS_IPSEC_EXT.1.3

TSS

The evaluator shall examine the TSS to verify that the TSS provides a description of how a packet is processed against the SPD and that if no “rules” are found to match, that a final rule exists, either implicitly or explicitly, that causes the network packet to be discarded.

Guidance

The evaluator checks that the operational guidance provides instructions on how to construct or acquire the SPD and uses the guidance to configure the TOE/platform for the following test.

Tests

The evaluator shall perform the following test:

- **Test 1**: The evaluator shall configure the SPD such that it has entries that contain operations that DISCARD, PROTECT, and (if applicable) BYPASS network packets. The evaluator may use the SPD that was created for verification of FCS_IPSEC_EXT.1.1. The evaluator shall construct a network packet that matches a BYPASS entry and send that packet. The evaluator should observe that the network packet is passed to the proper destination interface with no modification. The evaluator shall then modify a field in the packet header; such that it no longer matches the evaluator-created entries (there may be a "TOE/platform created" final entry that discards packets that do not match any previous entries). The evaluator sends the packet, and observes that the packet was not permitted to flow to any of the TOE's interfaces.

FCS_IPSEC_EXT.1.4

TSS

The evaluator shall examine the TSS to verify that the algorithms AES-GCM-128 and AES-GCM-256 are implemented. If the ST author has selected either AES-CBC-128 or AES-CBC-256 in the requirement, then the evaluator verifies the TSS describes these as well. In addition, the evaluator ensures that the SHA-based HMAC algorithm conforms to the algorithms specified in the relevant iteration of FCS_COP.1 from the Base-PP that applies to keyed-hash message authentication.

Guidance

The evaluator checks the operational guidance to ensure it provides instructions on how the TOE is configured to use the algorithms selected in this component and whether this is performed through direct configuration, defined during initial installation, or defined by acquiring configuration settings from an environmental component.

Tests

- **Test 1**: The evaluator shall configure the TOE/platform as indicated in the operational guidance configuring the TOE/platform to using each of the AES-GCM-128, and AES-GCM-256 algorithms, and attempt to establish a connection using ESP. If the ST Author has selected either AES-CBC-128 or AES-CBC-256, the TOE/platform is configured to use those algorithms and the evaluator attempts to establish a connection using ESP for those algorithms selected.

FCS_IPSEC_EXT.1.5[29]

TSS

The evaluator shall examine the TSS to verify that IKEv1 and/or IKEv2 are implemented. If IKEv1 is implemented, the evaluator shall verify that the TSS indicates whether or not XAUTH is supported, and that aggressive mode is not used for IKEv1 Phase 1 exchanges (i.e. only main mode is used). It may be that these are configurable options.

Guidance

The evaluator shall check the operational guidance to ensure it instructs the administrator how to configure the TOE/platform to use IKEv1 and/or IKEv2 (as selected), and uses the guidance to configure

[29] This protection profile assurance activity was modified as part of NIAP Technical Decision 662.

the TOE/platform to perform NAT traversal for the test below. If XAUTH is implemented, the evaluator shall verify that the operational guidance provides instructions on how it is enabled or disabled.

If the TOE supports IKEv1, the evaluator shall verify that the operational guidance either asserts that only main mode is used for Phase 1 exchanges, or provides instructions for disabling aggressive mode.

Tests

- **Test 1**: The evaluator shall configure the TOE/platform so that it will perform NAT traversal processing as described in the TSS and RFC 7296, section 2.23. The evaluator shall initiate an IPsec connection and determine that the NAT is successfully traversed. If the TOE/platform supports IKEv1 with or without XAUTH, the evaluator shall verify that this test can be successfully repeated with XAUTH enabled and disabled in the manner specified by the operational guidance. If the TOE/platform only supports IKEv1 with XAUTH, the evaluator shall verify that connections not using XAUTH are unsuccessful. If the TOE/platform only supports IKEv1 without XAUTH, the evaluator shall verify that connections using XAUTH are unsuccessful.

  In the case that the VPN gateway enforces the TOE's configuration, the following steps shall be performed to meet the objective of Test 1:

  1. Configure the TOE client and VPN gateway to have XAUTH enabled.
  2. Attempt the connection and observe that the connection succeeds and that XAUTH is used.
  3. Configure the TOE and gateway to have XAUTH disabled.
  4. Attempt the connection and observe that the connection succeeds and that XAUTH is not present.
  5. Attempt to configure a mismatch between the TOE and gateway (i.e. modify a local configuration setting on the client system)
  6. Verify that no IPsec connection is attempted until the gateway corrects the configuration settings
- **Test 2**: [conditional] If the TOE supports IKEv1, the evaluator shall perform any applicable operational guidance steps to disable the use of aggressive mode and then attempt to establish a connection using an IKEv1 Phase 1 connection in aggressive mode. This attempt should fail. The evaluator shall show that the TOE/platform will reject a VPN gateway from initiating an IKEv1 Phase 1 connection in aggressive mode. The evaluator should then show that main mode exchanges are supported.

  In the case that the VPN gateway enforces the TOE's configuration, the following steps should be performed to meet the objective of Test 2:

  1. Configure the gateway and TOE client in the appropriate manner per the guidance documentation. (Gateway rejects Aggressive mode, Client rejects aggressive mode)
  2. Connect the TOE client to the gateway to obtain the configuration settings.
  3. Observe the main mode connection is successful.
  4. Disconnect the TOE from the gateway.
  5. Attempt to modify the setting for main mode locally on the TOE to force the client to attempt to use aggressive mode.

6. Observe that when the initial connection attempt to the gateway is made, the gateway detects the configuration difference and reapplies the main mode setting before the TOE can attempt an IPsec connection.
7. Configure a peer to have equivalent settings to the VPN gateway (Same ciphers/Authentication/Hash/KEX settings)
8. Tell the TOE that there is a VPN gateway at the location of the peer.
9. Observe that the TOE cannot establish a connection with the peer.

FCS_IPSEC_EXT.1.6

TSS

The evaluator shall ensure the TSS identifies the algorithms used for encrypting the IKEv1 and/or IKEv2 payload, and that the algorithms AES-CBC-128, AES-CBC-256 are specified, and if others are chosen in the selection of the requirement, those are included in the TSS discussion.

Guidance

The evaluator checks the operational guidance to ensure it provides instructions on how the TOE is configured to use the algorithms selected in this component and whether this is performed through direct configuration, defined during initial installation, or defined by acquiring configuration settings from an environmental component.

Test

The evaluator shall use the operational guidance to configure the TOE/platform (or to configure the Operational Environment to have the TOE receive configuration) to perform the following test for each ciphersuite selected:

- **Test 1**: The evaluator shall configure the TOE/platform to use the ciphersuite under test to encrypt the IKEv1 and/or IKEv2 payload and establish a connection with a peer device, which is configured to only accept the payload encrypted using the indicated ciphersuite. The evaluator will confirm the algorithm was that used in the negotiation. The evaluator will confirm that the connection is successful by confirming that data can be passed through the connection once it is established. For example, the evaluator may connect to a webpage on the remote network and verify that it can be reached.

FCS_IPSEC_EXT.1.7

TSS

There are no TSS EAs for this requirement.

Guidance

The evaluator shall check the operational guidance to ensure it provides instructions on how the TOE configures the values for SA lifetimes. In addition, the evaluator shall check that the guidance has the option for either the Administrator or VPN Gateway to configure Phase 1 SAs if time-based limits are supported. Currently there are no values mandated for the number of packets or number of bytes, the evaluator shall simply check the operational guidance to ensure that this can be configured if selected in the requirement.

Tests

When testing this functionality, the evaluator needs to ensure that both sides are configured appropriately. From the RFC "A difference between IKEv1 and IKEv2 is that in IKEv1 SA lifetimes were negotiated. In IKEv2, each end of the SA is responsible for enforcing its own lifetime policy on the SA and rekeying the SA when necessary. If the two ends have different lifetime policies, the end with the shorter lifetime will end up always being the one to request the rekeying. If the two ends have the same lifetime policies, it is possible that both will initiate a rekeying at the same time (which will result in redundant SAs). To reduce the probability of this happening, the timing of rekeying requests SHOULD be jittered."

Each of the following tests shall be performed for each version of IKE selected in the FCS_IPSEC_EXT.1.5 protocol selection:

- **Test 1**: [conditional] The evaluator shall configure a maximum lifetime in terms of the # of packets (or bytes) allowed following the operational guidance. The evaluator shall establish an SA and determine that once the allowed # of packets (or bytes) through this SA is exceeded, the connection is closed.
- **Test 2**: [conditional] The evaluator shall construct a test where a Phase 1 SA is established and attempted to be maintained for more than 24 hours before it is renegotiated. The evaluator shall observe that this SA is closed or renegotiated in 24 hours or less. If such an action requires that the TOE be configured in a specific way, the evaluator shall implement tests demonstrating that the configuration capability of the TOE works as documented in the operational guidance.
- **Test 3**: [conditional] The evaluator shall perform a test similar to Test 2 for Phase 2 SAs, except that the lifetime will be 8 hours or less instead of 24 hours or less.
- **Test 4**: [conditional] If a fixed limit for IKEv1 SAs is supported, the evaluator shall establish an SA and observe that the connection is closed after the fixed traffic and/or time value is reached.

FCS_IPSEC_EXT.1.8

TSS

The evaluator shall check to ensure that the DH groups specified in the requirement are listed as being supported in the TSS. If there is more than one DH group supported, the evaluator checks to ensure the TSS describes how a particular DH group is specified/negotiated with a peer.

Guidance

There are no guidance EAs for this requirement.

Tests

The evaluator shall perform the following test:

- **Test 1**: For each supported DH group, the evaluator shall test to ensure that all supported IKE protocols can be successfully completed using that particular DH group.

FCS_IPSEC_EXT.1.9

TSS

The evaluator shall check to ensure that, for each DH group supported, the TSS describes the process for generating "x" (as defined in FCS_IPSEC_EXT.1.9) and each nonce. The evaluator shall verify that the TSS indicates that the random number generated that meets the requirements in this EP is used, and that the length of "x" and the nonces meet the stipulations in the requirement.

Guidance

There are no guidance EAs for this requirement.

Test

There are no test EAs for this requirement.

FCS_IPSEC_EXT.1.10

EAs for this element are tested through EAs for FCS_IPSEC_EXT.1.9.

FCS_IPSEC_EXT.1.11

TSS

The evaluator shall ensures that the TSS whether peer authentication is performed using RSA, ECDSA, or both.

If any selection with pre-shared keys is chosen in the selection, the evaluator shall check to ensure that the TSS describes how those selections work in conjunction with authentication of IPsec connections.

The evaluator shall ensure that the TSS describes how the TOE compares the peer's presented identifier to the reference identifier. This description shall include whether the certificate presented identifier is compared to the ID payload presented identifier, which fields of the certificate are used as the presented identifier (DN, Common Name, or SAN) and, if multiple fields are supported, the logical order comparison. If the ST author assigned an additional identifier type, the TSS description shall also include a description of that type and the method by which that type is compared to the peer's presented certificate.

Guidance

If any selection with "Pre-shared Keys" is selected, the evaluator shall check that the operational guidance describes any configuration necessary to enable any selected authentication mechanisms.

If any method other than no other method is selected, the evaluator shall check that the operational guidance describes any configuration necessary to enable any selected authentication mechanisms.

The evaluator ensures the operational guidance describes how to set up the TOE to use the cryptographic algorithms RSA, ECDSA, or either, depending which is claimed in the ST.

In order to construct the environment and configure the TOE for the following tests, the evaluator will ensure that the operational guidance also describes how to configure the TOE to connect to a trusted CA, and ensure a valid certificate for that CA is loaded into the TOE as a trusted CA.

The evaluator shall also ensure that the operational guidance includes the configuration of the reference identifiers for the peer.

Tests

For efficiency's sake, the testing that is performed here has been combined with the testing for FIA_X509_EXT.2 and FIA_X509_EXT.3 (for IPsec connections and depending on the Base-PP), FCS_IPSEC_EXT.1.12, and FCS_IPSEC_EXT.1.13. The following tests shall be repeated for each peer authentication protocol selected in the FCS_IPSEC_EXT.1.11 selection above:

- **Test 1**: The evaluator shall have the TOE generate a public-private key pair, and submit a CSR (Certificate Signing Request) to a CA (trusted by both the TOE and the peer VPN used to establish a connection) for its signature. The values for the DN (Common Name, Organization, Organizational Unit, and Country) will also be passed in the request. Alternatively, the evaluator may import to the TOE a previously generated private key and corresponding certificate.
- **Test 2**: The evaluator shall configure the TOE to use a private key and associated certificate signed by a trusted CA and shall establish an IPsec connection with the peer.
- **Test 3**: The evaluator shall test that the TOE can properly handle revoked certificates – conditional on whether CRL or OCSP is selected; if both are selected, and then a test is performed for each method. For this current version of the PP-Module, the evaluator has to only test one up in the trust chain (future drafts may require to ensure the validation is done up the entire chain). The evaluator shall ensure that a valid certificate is used, and that the SA is established. The evaluator then attempts the test with a certificate that will be revoked (for each method chosen in the selection) to ensure when the certificate is no longer valid that the TOE will not establish an SA..
- **Test 4**: [conditional]: For each selection made, the evaluator shall verify factors are required, as indicated in the operational guidance, to establish an IPsec connection with the server. For each supported identifier type (excluding DNs), the evaluator shall repeat the following tests:
- **Test 5**: For each field of the certificate supported for comparison, the evaluator shall configure the peer's reference identifier on the TOE (per the administrative guidance) to match the field in the peer's presented certificate and shall verify that the IKE authentication succeeds.
- **Test 6**: For each field of the certificate support for comparison, the evaluator shall configure the peer's reference identifier on the TOE (per the administrative guidance) to not match the field in the peer's presented certificate and shall verify that the IKE authentication fails.

The following tests are conditional:

- **Test 7**: [conditional]: If, according to the TSS, the TOE supports both Common Name and SAN certificate fields and uses the preferred logic outlined in the Application Note, the tests above with the Common Name field shall be performed using peer certificates with no SAN extension. Additionally, the evaluator shall configure the peer's reference identifier on the TOE to not match the SAN in the peer's presented certificate but to match the Common Name in the peer's presented certificate, and verify that the IKE authentication fails.
- **Test 8**: [conditional]: If the TOE supports DN identifier types, the evaluator shall configure the peer's reference identifier on the TOE (per the administrative guidance) to match the subject DN in the peer's presented certificate and shall verify that the IKE authentication succeeds. To demonstrate a bit-wise comparison of the DN, the evaluator shall change a single bit in the DN (preferably, in an Object Identifier (OID) in the DN) and verify that the IKE authentication fails. **To demonstrate a comparison of DN values, the evaluator shall change any one of the four DN values and verify that the IKE authentication fails.**

- **Test 9**: [conditional]: If the TOE supports both IPv4 and IPv6 and supports IP address identifier types, the evaluator must repeat test 1 and 2 with both IPv4 address identifiers and IPv6 identifiers. Additionally, the evaluator shall verify that the TOE verifies that the IP header matches the identifiers by setting the presented identifiers and the reference identifier with the same IP address that differs from the actual IP address of the peer in the IP headers and verifying that the IKE authentication fails.
- **Test 10**: [conditional]: If, according to the TSS, the TOE performs comparisons between the peer's ID payload and the peer's certificate, the evaluator shall repeat the following test for each combination of supported identifier types and supported certificate fields (as above). The evaluator shall configure the peer to present a different ID payload than the field in the peer's presented certificate and verify that the TOE fails to authenticate the IKE peer.

FCS_IPSEC_EXT.1.12

EAs for this element are tested through EAs for FCS_IPSEC_EXT.1.11.

FCS_IPSEC_EXT.1.13

EAs for this element are tested through EAs for FCS_IPSEC_EXT.1.11.

FCS_IPSEC_EXT.1.14

TSS

The evaluator shall check that the TSS describes the potential strengths (in terms of the number of bits in the symmetric key) of the algorithms that are allowed for the IKE and ESP exchanges. The TSS shall also describe the checks that are done when negotiating IKEv1 Phase 2 and/or IKEv2 CHILD_SA suites to ensure that the strength (in terms of the number of bits of key in the symmetric algorithm) of the negotiated algorithm is less than or equal to that of the IKE SA this is protecting the negotiation.

Guidance

There are no guidance EAs for this requirement.

Tests

The evaluator follows the guidance to configure the TOE/platform to perform the following tests.

- **Test 1**: This test shall be performed for each version of IKE supported. The evaluator shall successfully negotiate an IPsec connection using each of the supported algorithms and hash functions identified in the requirements.
- **Test 2**: [conditional] This test shall be performed for each version of IKE supported. The evaluator shall attempt to establish an SA for ESP that selects an encryption algorithm with more strength than that being used for the IKE SA (i.e., symmetric algorithm with a key size larger than that being used for the IKE SA). Such attempts should fail.
- **Test 3**: This test shall be performed for each version of IKE supported. The evaluator shall attempt to establish an IKE SA using an algorithm that is not one of the supported algorithms and hash functions identified in the requirements. Such an attempt should fail.
- **Test 4**: This test shall be performed for each version of IKE supported. The evaluator shall attempt to establish an SA for ESP (assumes the proper parameters where used to establish the

IKE SA) that selects an encryption algorithm that is not identified in FCS_IPSEC_EXT.1.4. Such an attempt should fail.

5.2.4.3 User Data Protection (FDP)

5.2.4.3.1 Spit Tunnel Prevention (FDP_VPN_EXT.1)[30]

TSS

The evaluator shall verify that the TSS section of the ST describes the routing of IP traffic through processes on the TSF when a VPN client is enabled. The evaluator shall ensure that the description indicates which traffic does not go through the VPN and which traffic does and that a configuration exists for each baseband protocol in which only the traffic identified by the ST author is necessary for establishing the VPN connection (IKE traffic and perhaps HTTPS or DNS traffic) is not encapsulated by the VPN protocol (IPsec). The ST author shall also identify in the TSS section any differences in the routing of IP traffic when using any supported baseband protocols (e.g. Wi-Fi or LTE).

Operational Guidance

The evaluator shall verify that the following is addressed by the documentation:

- The description above indicates that if a VPN client is enabled, all configurations route all IP traffic (other than IP traffic required to establish the VPN connection) through the VPN client.
- The AGD guidance describes how the user and/or administrator can configure the TSF to meet this requirement.

Test

The evaluator shall perform the following test:

Step 1 - The evaluator shall use the platform to enable a network connection without using IPsec. The evaluator shall use a packet sniffing tool between the platform and an Internet-connected network. The evaluator shall turn on the sniffing tool and perform actions with the device such as navigating to websites, using provided applications, accessing other Internet resources (Use Case 1), accessing another VPN client (Use Case 2), or accessing an IPsec-capable network device (Use Case 3). The evaluator shall verify that the sniffing tool captures the traffic generated by these actions, turn off the sniffing tool, and save the session data.

Step 2 - The evaluator shall configure an IPsec VPN client that supports the routing specified in this requirement, and if necessary, configure the device to perform the routing specified as described in the AGD guidance. The evaluator shall turn on the sniffing tool, establish the VPN connection, and perform the same actions with the device as performed in the first step. The evaluator shall verify that the sniffing tool captures traffic generated by these actions, turn off the sniffing tool, and save the session data.

[30] This protection profile assurance activity was added as part of NIAP Technical Decision 690.

Step 3 - The evaluator shall examine the traffic from both step one and step two to verify that all IP traffic, aside from and after traffic necessary for establishing the VPN (such as IKE, DNS, and possibly HTTPS), is encapsulated by IPsec.

Step 4 - The evaluator shall attempt to send packets to the TOE outside the VPN connection and shall verify that the TOE discards them.

5.2.4.3.2 Full Residual Information Protection (FDP_RIP.2)

TSS

Requirement met by the platform

The evaluator shall examine the TSS to verify that it describes (for each supported platform) the extent to which the client processes network packets and addresses the FDP_RIP.2 requirement.

Requirement met by the TOE

“Resources” in the context of this requirement are network packets being sent through (as opposed to “to”, as is the case when a security administrator connects to the TOE) the TOE. The concern is that once a network packet is sent, the buffer or memory area used by the packet still contains data from that packet, and that if that buffer is re-used, those data might remain and make their way into a new packet. The evaluator shall check to ensure that the TSS describes packet processing to the extent that they can determine that no data will be reused when processing network packets. The evaluator shall ensure that this description at a minimum describes how the previous data are zeroized/overwritten, and at what point in the buffer processing this occurs.

Guidance

There are no AGD Assurance Activities for this requirement.

Tests

There are no test EAs for this requirement.

5.2.4.4 Identification & Authentication (FIA)

5.2.4.4.1 Pre-Shared Key Composition (FIA_PSK_EXT.1)

TSS

The evaluator shall examine the TSS to ensure that it identifies all protocols that allow pre-shared keys. For each protocol identified by the requirement, the evaluator shall confirm that the TSS states which pre-shared key selections are supported.

Guidance

The evaluator shall examine the operational guidance to determine that it provides guidance to administrators on how to configure all selected pre-shared key options if any configuration is required.

Tests

The evaluator shall also perform the following tests for each protocol (or instantiation of a protocol, if performed by a different implementation on the TOE).

- **Test 1**: For each mechanism selected in FIA_PSK_EXT.1.2, the evaluator shall attempt to establish a connection and confirm that the connection requires the selected factors in the PSK to establish the connection.

5.2.4.4.2 X.509 Certificate Use and Management (FIA_X509_EXT.3)

TSS

The evaluator shall check the TSS to ensure that it describes whether the VPN client or the OS implements the certificate validation functionality, how the VPN client/OS chooses which certificates to use, and any necessary instructions in the administrative guidance for configuring the OS so that desired certificates can be used.

The evaluator shall examine the TSS to confirm that it describes the behavior of the client/OS when a connection cannot be established during the validity check of a certificate used in establishing a trusted channel.

Guidance

If the requirement indicates that the administrator is able to specify the default action, then the evaluator shall ensure that the operational guidance contains instructions on how this configuration action is performed.

Tests

The evaluator shall perform the following test regardless of whether the certificate validation functionality is implemented by the VPN client or by the OS:

Test 1: The evaluator shall demonstrate that using a valid certificate that requires certificate validation checking to be performed in at least some part by communicating with a non-TOE IT entity. The evaluator shall then manipulate the environment so that the TOE is unable to verify the validity of the certificate, and observe that the action selected in FIA_X509_EXT.3.2 is performed. If the selected action is administrator-configurable, then the evaluator shall follow the operational guidance to determine that all supported administrator-configurable options behave in their documented manner.

5.2.4.5 Security Management (FMT)

5.2.4.5.1 Specification of Management Functions (VPN) (FMT_SMF.1(VPN))

Application Note: FMT_SMF.1(VPN) corresponds to FMT_SMF.1/VPN in the VPN Client Module.

TSS

The evaluator shall check to ensure the TSS describes the client credentials and how they are used by the TOE.

Guidance

The evaluator shall check to make sure that every management function mandated in the ST for this requirement is described in the operational guidance and that the description contains the information required to perform the management duties associated with each management function.

Tests

The evaluator shall test the TOE's ability to provide the management functions by configuring the TOE according to the operational guidance and testing each management activity listed in the ST.

The evaluator shall ensure that all management functions claimed in the ST can be performed by completing activities described in the AGD. Note that this may be performed in the course of completing other testing.

5.2.4.6 Protection of the TSF (FPT)

5.2.4.6.1 Self-Test (FPT_TST_EXT.1 (VPN))

Application Note: FPT_TST_EXT.1(VPN) corresponds to FPT_TST_EXT.1 in the VPN Client Module.

Except for where it is explicitly noted, the evaluator is expected to check the following information regardless of whether the functionality is implemented by the TOE or by the TOE platform.

TSS

The evaluator shall examine the TSS to ensure that it details the self-tests that are run by the TSF on start-up; this description should include an outline of what the tests are actually doing (e.g., rather than saying "memory is tested", a description similar to "memory is tested by writing a value to each memory location and reading it back to ensure it is identical to what was written" shall be used). The evaluator shall ensure that the TSS makes an argument that the tests are sufficient to demonstrate that the TSF is operating correctly. If some of the tests are performed by the TOE platform, the evaluator shall check the TSS to ensure that those tests are identified, and that the ST for each platform contains a description of those tests. Note that the tests that are required by this component are those that support security functionality in this PP-Module, which may not correspond to the set of all self-tests contained in the platform STs.

The evaluator shall examine the TSS to ensure that it describes how the integrity of stored TSF executable code is cryptographically verified when it is loaded for execution. The evaluator shall ensure that the TSS makes an argument that the tests are sufficient to demonstrate that the integrity of stored TSF executable code has not been compromised. The evaluator shall check to ensure that the cryptographic requirements listed are consistent with the description of the integrity verification process.

The evaluator also ensures that the TSS (or the operational guidance) describes the actions that take place for successful (e.g. hash verified) and unsuccessful (e.g., hash not verified) cases. For checks implemented entirely by the platform, the evaluator ensures that the operational guidance for the TOE references or includes the platform-specific guidance for each platform listed in the ST.

Guidance

If not present in the TSS, the evaluator ensures that the operational guidance describes the actions that take place for successful (e.g. hash verified) and unsuccessful (e.g., hash not verified) cases. For checks implemented entirely by the platform, the evaluator ensures that the operational guidance for the TOE references or includes the platform-specific guidance for each platform listed in the ST.

Tests

The evaluator shall perform the following tests:

- **Test 1**: The evaluator performs the integrity check on a known good TSF executable and verifies that the check is successful.
- **Test 2**: The evaluator modifies the TSF executable, performs the integrity check on the modified TSF executable and verifies that the check fails.

5.2.4.7 Trusted Path/Channels (FTP)

5.2.4.7.1 Inter-TSF Trusted Channel (FTP_ITC.1(VPN))

Application Note: FTP_ITC.1(VPN) corresponds to FTP_ITC.1 in the VPN Client Module.

TSS

The evaluator shall examine the TSS to determine that it describes the details of the TOE connecting to a VPN gateway, VPN client, or IPsec-capable network device in terms of the cryptographic protocols specified in the requirement, along with TOE-specific options or procedures that might not be reflected in the specification. evaluator shall also confirm that all protocols listed in the TSS are specified and included in the requirements in the ST.

Guidance

The evaluator shall confirm that the operational guidance contains instructions for establishing the connection to a VPN gateway, VPN client, or IPsec-capable network device, and that it contains recovery instructions should a connection be unintentionally broken.

Tests

The evaluator shall perform the following tests:

- Test 1: The evaluator shall ensure that the TOE is able to initiate communications with a VPN gateway, VPN client, IPsec-capable network device using the protocols specified in the requirement, setting up the connections as described in the operational guidance and ensuring that communication is successful.
- Test 2: The evaluator shall ensure, for each communication channel with an IPsec peer, the channel data is not sent in plaintext.
- Test 3: The evaluator shall ensure, for each communication channel with an IPsec peer, modification of the channel data is detected by the TOE.
- Test 4: The evaluator shall physically interrupt the connection from the TOE to the IPsec peer. The evaluators shall ensure that subsequent communications are appropriately protected, at a minimum in the case of any attempts to automatically resume the connection or connect to a new access point.

Further EAs are associated with requirements for FCS_IPSEC_EXT.1.

5.2.5 Bluetooth Module Assurance Activities

This section copies the assurance activities from the Bluetooth PP-Module in order to ease reading and comparisons between the extended package and the security.

5.2.5.1 Security Audit (FAU)

5.2.5.1.1 Audit Data Generation (FAU_GEN.1(BT))

TSS

There are additional auditable events that serve to extend the FAU_GEN.1 SFR found in each Base-PP.

This SFR is evaluated in the same manner as defined by the Evaluation Activities for the claimed Base-PP. The only difference is that the evaluator shall also assess the auditable events required for this PP-Module in addition to those defined in the claimed Base-PP.

5.2.5.2 Cryptographic Support (FCS)

5.2.5.2.1 Bluetooth Key Generation (FCS_CKM_EXT.8)

TSS

The evaluator shall ensure that the TSS describes the criteria used to determine the frequency of generating new ECDH public/private key pairs. In particular, the evaluator shall ensure that the implementation does not permit the use of static ECDH key pairs.

Guidance

There are no guidance evaluation activities for this component.

Tests

The evaluator shall perform the following steps:

Step 1: Pair the TOE to a remote Bluetooth device and record the public key currently in use by the TOE.

(This public key can be obtained using a Bluetooth protocol analyzer to inspect packets exchanged during pairing.)

Step 2: Perform necessary actions to generate new ECDH public/private key pairs. (Note that this test step depends on how the TSS describes the criteria used to determine the frequency of generating new ECDH public/private key pairs.)

Step 3: Pair the TOE to a remote Bluetooth device and again record the public key currently in use by the TOE.

Step 4: Verify that the public key in Step 1 differs from the public key in Step 3.

5.2.5.3 Identification & Authentication (FIA)

5.2.5.3.1 Bluetooth User Authorization (FIA_BLT_EXT.1)

TSS

The evaluator shall examine the TSS to ensure that it contains a description of when user permission is required for Bluetooth pairing; and that this description mandates explicit user authorization via manual input for all Bluetooth pairing; including application use of the Bluetooth trusted channel and situations where temporary (non-bonded) connections are formed.

Guidance

The evaluator shall examine the API documentation provided as a means of satisfying the requirements for the ADV assurance class (see section 5.2.2 in the MDF PP and GPOS PP) and verify that this API documentation does not include any API for programmatic entering of pairing information (e.g. PINs; numeric codes; or "yes/no" responses) intended to bypass manual user input during pairing.

The evaluator shall examine the guidance to verify that these user authorization screens are clearly identified and instructions are given for authorizing Bluetooth pairings.

Tests

The evaluator shall perform the following steps:

Step 1: Initiate pairing with the TOE from a remote Bluetooth device that requests no man-in-the-middle protection; no bonding; and claims to have NoInput/NoOutput (IO) capability. Such a device will attempt to evoke behavior from the TOE that represents the minimal level of user interaction that the TOE supports during pairing.

Step 2: Verify that the TOE does not permit any Bluetooth pairing without explicit authorization from the user (e.g. the user must have to minimally answer "yes" or "allow" in a prompt).

5.2.5.3.2 Bluetooth Mutual Authentication (FIA_BLT_EXT.2)

TSS

The evaluator shall ensure that the TSS describes how data transfer of any type is prevented before the Bluetooth pairing is completed. The TSS shall specifically call out any supported RFCOMM and L2CAP data transfer mechanisms. The evaluator shall ensure that the data transfers are only completed after the Bluetooth devices are paired and mutually authenticated.

Guidance

There are no guidance evaluation activities for this component.

Tests

The evaluator shall use a Bluetooth tool to attempt to access TOE files using the OBEX Object Push service (OBEX Push) and verify that pairing and mutual authentication are required by the TOE before allowing access. If the OBEX Object Push service is unsupported on the TOE; a different service that

transfers data over Bluetooth L2CAP and/or RFCOMM may be used in this test.

5.2.5.3.3 Rejection of Duplicate Bluetooth Connections (FIA_BLT_EXT.3)

TSS

The evaluator shall ensure that the TSS describes how Bluetooth sessions are maintained such that at least two devices with the same Bluetooth device address are not simultaneously connected and such that the initial session is not superseded by any following session initialization attempts.

Guidance

There are no guidance evaluation activities for this component.

Tests

The evaluator shall perform the following steps:

Step 1: Pair the TOE with a remote Bluetooth device (DEV1) with a known address BD_ADDR. Establish an active session between the TOE and DEV1 with the known address BD_ADDR.

Step 2: Attempt to pair a second remote Bluetooth device (DEV2) claiming to have a Bluetooth device address matching DEV1 BD_ADDR to the TOE. Using a Bluetooth protocol analyzer, verify that the pairing attempt by DEV2 is not completed by the TOE and that the active session to DEV1 is unaffected.

Step 3: Attempt to initialize a session to the TOE from DEV2 containing address DEV1 BD_ADDR. Using a Bluetooth protocol analyzer, verify that the session initialization attempt by DEV2 is ignored by the TOE and that the initial session to DEV1 is unaffected.

5.2.5.3.4 Secure Simple Pairing (FIA_BLT_EXT.4)

TSS

The evaluator shall verify that the TSS describes the secure simple pairing process.

Guidance

There are no guidance evaluation activities for this component.

Tests

The evaluator shall perform the following steps:

Step 1: Initiate pairing with the TOE from a remote Bluetooth device that supports Secure Simple Pairing.

Step 2: During the pairing process; observe the packets in a Bluetooth protocol analyzer and verify that the TOE claims support for both "Secure Simple Pairing (Host Support)" and "Secure Simple Pairing (Controller Support)" during the LMP Features Exchange.

Step 3: Verify that Secure Simple Pairing is used during the pairing process.

5.2.5.3.5 Trusted Bluetooth Device User Authorization (FIA_BLT_EXT.6)

TSS

The evaluator shall verify that the TSS describes all Bluetooth profiles and associated services for which explicit user authorization is required before a remote device can gain access. The evaluator shall also verify that the TSS describes any difference in behavior based on whether or not the device has a trusted relationship with the TOE for that service (i.e. whether there are any services that require explicit user authorization for untrusted devices that do not require such authorization for trusted devices). The evaluator shall also verify that the TSS describes the method by which a device can become 'trusted'.

Guidance

There are no guidance evaluation activities for this component.

Tests

The evaluator shall perform the following tests:

- **Test 1**: While the service is in active use by an application on the TOE, the evaluator shall attempt to gain access to a "protected" Bluetooth service (as specified in the assignment in FIA_BLT_EXT.6.1) from a "trusted" remote device. The evaluator shall verify that the user is explicitly asked for authorization by the TOE to allow access to the service for the particular remote device. The evaluator shall deny the authorization on the TOE and verify that the remote attempt to access the service fails due to lack of authorization.
- **Test 2**: The evaluator shall repeat Test 1, this time allowing the authorization and verifying that the remote device successfully accesses the service.

5.2.5.3.6 Untrusted Bluetooth Device User Authorization (FIA_BLT_EXT.7)

TSS

The TSS evaluation activities for this component are addressed by FIA_BLT_EXT.6.

Guidance

There are no guidance evaluation activities for this component.

Tests

The evaluator shall perform the following tests if the TSF differentiates between "trusted" and "untrusted" devices for the purpose of granting access to services. If it does not, then the test evaluation activities for FIA_BLT_EXT.6 are sufficient to satisfy this component.

- **Test 1**: While the service is in active use by an application on the TOE, the evaluator shall attempt to gain access to a "protected" Bluetooth service (as specified in the assignment in FIA_BLT_EXT.7.1) from an "untrusted" remote device. The evaluator shall verify that the user is explicitly asked for authorization by the TOE to allow access to the service for the particular remote device. The evaluator shall deny the authorization on the TOE and verify that the remote attempt to access the service fails due to lack of authorization.

- **Test 2**: The evaluator shall repeat Test 1, this time allowing the authorization and verifying that the remote device successfully accesses the service.
- **Test 3**: (conditional): If there exist any services that require explicit user authorization for access by untrusted devices but not by trusted devices (i.e. a service that is listed in FIA_BLT_EXT.7.1 but not FIA_BLT_EXT.6.1), the evaluator shall repeat Test 1 for these services and observe that the results are identical. That is, the evaluator shall use these results to verify that explicit user approval is required for an untrusted device to access these services, and failure to grant this approval will result in the device being unable to access them.
- **Test 4**: (conditional): If test 3 applies, the evaluator shall repeat Test 2 using any services chosen in Test 3 and observe that the results are identical. That is, the evaluator shall use these results to verify that explicit user approval is required for an untrusted device to access these services, and granting this approval will result in the device being able to access them.
- **Test 5**: (conditional): If test 3 applies, the evaluator shall repeat Test 3 except this time designating the device as "trusted" prior to attempting to access the service. The evaluator shall verify that access to the service is granted without explicit user authorization (because the device is now trusted and therefore FIA_BLT_EXT.7.1 no longer applies to it). That is, the evaluator shall use these results to demonstrate that the TSF will grant a device access to different services depending on whether or not the device is trusted.

5.2.5.4 Security Management (FMT)

5.2.5.4.1 Management of Security Functions Behavior for Bluetooth (FMT_MOF_EXT.1(BT))

TSS

The evaluator shall examine the TSS to ensure that it identifies the Bluetooth-related management functions that are supported by the TOE and the roles that are authorized to perform each function.

Guidance

The evaluator shall examine the operational guidance to ensure that it provides sufficient guidance on each supported Bluetooth management function to describe how the function is performed and any role restrictions on the subjects that are authorized to perform the function.

Tests

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 will 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.

5.2.5.4.2 Specification of Management Functions for VPN (FMT_SMF_EXT.1(BT))

TSS

The evaluator shall ensure that the TSS includes a description of the Bluetooth profiles and services supported and the Bluetooth security modes and levels supported by the TOE.

If function BT-4, "Allow/disallow additional wireless technologies to be used with Bluetooth," is selected, the evaluator shall verify that the TSS describes any additional wireless technologies that may be used with Bluetooth, which may include Wi-Fi with Bluetooth High Speed and/or NFC as an Out of Band pairing mechanism.

If function BT-5, "Configure allowable methods of Out of Band pairing (for BR/EDR and LE)," is selected, the evaluator shall verify that the TSS describes when Out of Band pairing methods are allowed and which ones are configurable.

If function BT-8, "Disable/enable the Bluetooth services and/or profiles available on the OS (for BR/EDR and LE)," is selected, the evaluator shall verify that all supported Bluetooth services are listed in the TSS as manageable and, if the TOE allows disabling by application rather than by service name, that a list of services for each application is also listed.

If function BT-9, "Specify minimum level of security for each pairing (for BR/EDR and LE)," is selected, the evaluator shall verify that the TSS describes the method by which the level of security for pairings are managed, including whether the setting is performed for each pairing or is a global setting.

Guidance

The evaluator shall ensure that the management functions defined in the PP-Module are described in the guidance to the same extent required for the Base-PP management functions.

Tests

The evaluator shall use a Bluetooth-specific protocol analyzer to perform the following tests:

- **Test 1**: The evaluator shall disable the Discoverable mode and shall verify that other Bluetooth BR/EDR devices cannot detect the TOE. The evaluator shall use the protocol analyzer to verify that the TOE does not respond to inquiries from other devices searching for Bluetooth devices. The evaluator shall enable Discoverable mode and verify that other devices can detect the TOE and that the TOE sends response packets to inquiries from searching devices.

  The following tests are conditional on if the corresponding function is included in the ST:

- **Test 2**: (conditional): The evaluator shall examine Bluetooth traffic from the TOE to determine the current Bluetooth device name, change the Bluetooth device name, and verify that the Bluetooth traffic from the TOE lists the new name. The evaluator shall examine Bluetooth traffic from the TOE to determine the current Bluetooth device name for BR/EDR and LE. The evaluator shall change the Bluetooth device name for LE independently of the device name for BR/EDR. The evaluator shall verify that the Bluetooth traffic from the TOE lists the new name.
- **Test 3**: (conditional): The evaluator shall disable Bluetooth BR/EDR and enable Bluetooth LE. The evaluator shall examine Bluetooth traffic from the TOE to confirm that only Bluetooth LE traffic is present. The evaluator shall repeat the test with Bluetooth BR/EDR enabled and Bluetooth LE disabled, confirming that only Bluetooth BR/EDR is present.
- **Test 4**: (conditional): For each additional wireless technology that can be used with Bluetooth as claimed in the ST, the evaluator shall revoke Bluetooth permissions from that technology. If the set of supported wireless technologies includes Wi-Fi, the evaluator shall verify that Bluetooth High Speed is not able to send Bluetooth traffic over Wi-Fi when disabled. If the set of supported

wireless technologies includes NFC, the evaluator shall verify that NFC cannot be used for pairing when disabled. For any other supported wireless technology, the evaluator shall verify that it cannot be used with Bluetooth in the specified manner when disabled. The evaluator shall then re-enable all supported wireless technologies and verify that all functionality that was previously unavailable has been restored.

- **Test 5**: (conditional): The evaluator shall attempt to pair using each of the Out of Band pairing methods, verify that the pairing method works, iteratively disable each pairing method, and verify that the pairing method fails.
- **Test 6**: (conditional): The evaluator shall enable Advertising for Bluetooth LE, verify that the advertisements are captured by the protocol analyzer, disable Advertising, and verify that no advertisements from the device are captured by the protocol analyzer
- **Test 7**: (conditional): The evaluator shall enable Connectable mode and verify that other Bluetooth devices may pair with the TOE and (if the devices were bonded) re-connect after pairing and disconnection. For BR/EDR devices: The evaluator shall use the protocol analyzer to verify that the TOE responds to pages from the other devices and permits pairing and re-connection. The evaluator shall disable Connectable mode and verify that the TOE does not respond to pages from remote Bluetooth devices, thereby not permitting pairing or re-connection. For LE: The evaluator shall use the protocol analyzer to verify that the TOE sends connectable advertising events and responds to connection requests. The evaluator shall disable Connectable mode and verify that the TOE stops sending connectable advertising events and stops responding to connection requests from remote Bluetooth devices.
- **Test 8**: (conditional): For each supported Bluetooth service and/or profile listed in the TSS, the evaluator shall verify that the service or profile is manageable. If this is configurable by application rather than by service and/or profile name, the evaluator shall verify that a list of services and/or profiles for each application is also listed.
- **Test 9**: (conditional): The evaluator shall allow low security modes/levels on the TOE and shall initiate pairing with the TOE from a remote device that allows only something other than Security Mode 4/Level 3 or Security Mode 4/Level 4 (for BR/EDR), or Security Mode 1/Level 3 (for LE). (For example, a remote BR/EDR device may claim Input/Output capability "NoInputNoOutput" and state that man-in-the-middle (MiTM) protection is not required. A remote LE device may not support encryption.) The evaluator shall verify that this pairing attempt succeeds due to the TOE falling back to the low security mode/level. The evaluator shall then remove the pairing of the two devices, prohibit the use of low security modes/levels on the TOE, then attempt the connection again. The evaluator shall verify that the pairing attempt fails. With the low security modes/levels disabled, the evaluator shall initiate pairing from the TOE to a remote device that supports Security Mode 4/Level 3 or Security Mode 4/Level 4 (for BR/EDR) or Security Mode 1/Level 3 (for LE). The evaluator shall verify that this pairing is successful and uses the high security mode/level.

5.2.5.5 Trusted Path/Channels (FTP)

5.2.5.5.1 Bluetooth Encryption (FTP_BLT_EXT.1)

TSS

The evaluator shall verify that the TSS describes the use of encryption, the specific Bluetooth protocol(s) it applies to, and whether it is enabled by default.

The evaluator shall verify that the TSS includes the protocol used for encryption of the transmitted data and the key generation mechanism used.

Guidance

The evaluator shall verify that the operational guidance includes instructions on how to configure the TOE to require the use of encryption during data transmission (unless this behavior is enforced by default).

Tests

There are no test EAs for this component. Testing for this SFR is addressed through the evaluation of FTP_BLT_EXT.3/BR and, if claimed, FTP_BLT_EXT.3/LE.

5.2.5.5.2 Persistence of Bluetooth Encryption (FTP_BLT_EXT.2)

TSS

The evaluator shall verify that the TSS describes the TSF's behavior if a remote device stops encryption while connected to the TOE.

Guidance

The evaluator shall verify that the operational guidance describes how to enable/disable encryption (if configurable).

Tests

The evaluator shall perform the following steps using a Bluetooth protocol analyzer to observe packets pertaining to the encryption key size:

Step 1: Initiate pairing with the TOE from a remote Bluetooth device that has been configured to have a minimum encryption key size that is equal to or greater than that of the TOE.

Step 2: After pairing has successfully finished and while a connection exists between the TOE and the remote device; turn off encryption on the remote device. This can be done using commercially-available tools.

Step 3: Verify that the TOE either restarts encryption with the remote device or terminates the connection with the remote device.

5.2.5.5.3 Bluetooth Encryption Parameters (BR/EDR) (FTP_BLT_EXT.3(BR))

TSS

The evaluator shall examine the TSS and verify that it specifies the minimum key size for BR/EDR encryption, whether this value is configurable, and the mechanism by which the TOE will not negotiate keys sizes smaller than the minimum.

Guidance

The evaluator shall verify that the guidance includes instructions on how to configure the minimum encryption key size for BR/EDR encryption, if configurable.

Tests

The evaluator shall perform the following tests:

- **Test 1**: The evaluator shall perform the following steps using a Bluetooth protocol analyzer to observe packets pertaining to the encryption key size:

  Step 1: Initiate BR/EDR pairing with the TOE from a remote Bluetooth device that has been configured to have a minimum encryption key size that is equal to or greater than that of the TOE. This can be done using certain commercially-available tools that can send the appropriate command to certain commercially-available Bluetooth controllers.

  Step 2: Use a Bluetooth packet sniffer to verify that the encryption key size negotiated for the connection is at least as large as the minimum encryption key size defined for the TOE.

- **Test 2**: (conditional): If the encryption key size is configurable, configure the TOE to support a different minimum key size, then repeat Test 1 and verify that the negotiated key size is at least as large as the new minimum value
- **Test 3**: The evaluator shall perform the following steps using a Bluetooth protocol analyzer to observe packets pertaining to the encryption key size:

  Step 1: Initiate BR/EDR pairing with the TOE from a remote Bluetooth device that has been configured to have a maximum encryption key size of 1 byte. This can be done using certain commercially-available tools that can send the appropriate command to certain commercially-available Bluetooth controllers.

  Step 2: Verify that the encryption key size suggested by the remote device is not accepted by the TOE and that the connection is not completed.

5.2.5.5.4 Bluetooth Encryption Parameters (LE) (FTP_BLT_EXT.3(LE))

TSS

The evaluator shall examine the TSS and verify that it specifies the minimum key size for LE encryption, whether this value is configurable, and the mechanism by which the TOE will not negotiate keys sizes smaller than the minimum.

Guidance

The evaluator shall verify that the guidance includes instructions on how to configure the minimum encryption key size for LE encryption, if configurable.

Tests

The evaluator shall perform the following tests:

- **Test 1**: The evaluator shall perform the following steps using a Bluetooth protocol analyzer to observe packets pertaining to the encryption key size:

Step 1: Initiate LE pairing with the TOE from a remote Bluetooth device that has been configured to have a minimum encryption key size that is equal to or greater than that of the TOE. This can be done using certain commercially-available tools that can send the appropriate command to certain commercially-available Bluetooth controllers.

Step 2: Use a Bluetooth packet sniffer to verify that the encryption key size negotiated for the connection is at least as large as the minimum encryption key size defined for the TOE.

- **Test 2**: (conditional): If the encryption key size is configurable, configure the TOE to support a different minimum key size, then repeat Test 1 and verify that the negotiated key size is at least as large as the new minimum value.
- **Test 3**: The evaluator shall perform the following steps using a Bluetooth protocol analyzer to observe packets pertaining to the encryption key size:

  Step 1: Initiate LE pairing with the TOE from a remote Bluetooth device that has been configured to have a maximum encryption key size of 1 byte. This can be done using certain commercially-available tools that can send the appropriate command to certain commercially-available Bluetooth controllers.

  Step 2: Verify that the encryption key size suggested by the remote device is not accepted by the TOE and that the connection is not completed.

5.2.6 TLS Module Assurance Activities

This section copies the assurance activities from the TLS Module in order to ease reading and comparisons between the extended package and the security target.

5.2.6.1 Cryptographic Support (FCS)

5.2.6.1.1 TLS Protocol (FCS_TLS_EXT.1)

TSS

The evaluator shall examine the TSS to verify that the TLS and DTLS claims are consistent with those selected in the SFR.

Guidance

The evaluator shall ensure that the selections indicated in the ST are consistent with selections in the dependent components.

Tests

There are no test activities for this SFR; the following information is provided as an overview of the expected functionality and test environment for all subsequent SFRs.

5.2.6.1.2 TLS Client Protocol (FCS_TLSC_EXT.1)

TSS

The evaluator shall check the description of the implementation of this protocol in the TSS to ensure the supported TLS versions, features, ciphersuites, and extensions are specified in accordance with RFC 5246

(TLS 1.2) and RFC 8446 (TLS 1.3 and updates to TLS 1.2) and as refined in FCS_TLSC_EXT.1 as appropriate.

The evaluator shall verify that ciphersuites indicated in FCS_TLSC_EXT.1.2 are included in the description, and that none of the following ciphersuites are supported: ciphersuites indicating 'NULL,' 'RC2,' 'RC4,' 'DES,' 'IDEA,' or 'TDES' in the encryption algorithm component, indicating 'anon,' or indicating MD5 or SHA in the message digest algorithm component.

The evaluator shall verify that the TLS implementation description includes the extensions as required in FCS_TLSC_EXT.1.4.

The evaluator shall verify that the ST describes applications that use the TLS functions and how they establish reference identifiers. The evaluator shall verify that the ST includes a description of matching methods used for each supported name type to the supported application defined reference identifiers. The evaluator shall verify that the ST includes a description of wildcards recognized for each name type claimed in FCS_TLSC_EXT.1.5 and shall verify that the matching rules meet or exceed best practices. In particular, the evaluator shall ensure that the matching rules are as restrictive as, or more restrictive than the following:

- DNS names: The ‘*’ character used in the complete leftmost label of a DNS name represents any valid name that has the same number of labels, and that matches all remaining labels. The ‘*’ character must only be used in the leftmost complete label of a properly formatted DNS name. The ‘*’ must not be used to represent a public suffix, or in the leftmost label immediately following a public suffix.
- URI or SRV names: The ‘*’ character can only occur in the domain name portion of the name represented as a DNS name. All restrictions for wildcards in DNS names apply to the DNS portion of the name. URI host names presented as an IP address are matched according to IP address matching rules – see best practices for IP addresses below. In accordance with RFC 6125, it is preferred that such URIs are presented a matching name of type IP address in the SAN.
- IP addresses: RFC 5280 does not support IP address ranges as presented names, but indicates that presented names may be compared to IP address ranges present in name constraints. If the TSF supports IP address ranges as reference identifiers, the reference identifier matches if the presented name is in the range. IP ranges in name constraints (including reference identifiers) should be presented in CIDR format. RFC 2822 names: RFC 5280 and updates RFC 8398 and RFC 8399 do not support special indicators representing more than a single mailbox as a presented name, but indicates that presented names may be compared to a single mailbox, ‘any’ email address at a host, or ‘any’ email address on a domain (e.g., “example.com” matches any email address on the host example.com and “.example.com” matches any email address in the domain example.com, but does not match email addresses at the host “example.com”). Such matching is prohibited for internationalized RFC 2822 names.
- Embedded CN name types: The CN relative distinguished name of a DNS name type included in the subject field is not strongly typed. Attempts to match both the name type and wildcard specifications can result in matches not intended, and therefore, not authoritatively asserted by a certification authority. It is preferred that no matching of CN embedded names be supported, but if necessary for backward compatibility, the description should clearly indicate how different name types are interpreted in the matching algorithm. In particular, the ‘*’ character in a CN is

not to be interpreted as representing more than a single entity unless the entirety of the RDN is properly formatted as a DNS, URI, or SVR name, and represents a wildcard meeting best practices as described above.

Guidance

The evaluator shall check the operational guidance to ensure that it contains instructions on configuring the product so that TLS conforms to the description in the TSS and that it includes any instructions on configuring the version, ciphersuites, or optional extensions that are supported. The evaluator shall verify that all configurable features for matching identifiers in certificates presented in the TLS handshake to application specific reference identifiers are described.

Tests

The evaluator shall perform the following tests:

- Test 3: (supported configurations) For each supported version, and for each supported ciphersuite associated with the version:

  The evaluator shall establish a TLS connection between the TOE and a test TLS server that is configured to negotiate the tested version and ciphersuite in accordance with the RFC for the version.

  The evaluator shall observe that the TSF presents a client hello with the highest version of TLS 1.2 or the legacy version (value '03 03') and shall observe that the supported version extension is not included for TLS 1.2, and, if TLS 1.3 is supported, is present and contains the value '03 04' for TLS 1.3.

  The evaluator shall observe that the client hello indicates the supported ciphersuites in the order indicated, and that it includes only the extensions supported, with appropriate values, for that version in accordance with the requirement.

  The evaluator shall observe that the TOE successfully completes the TLS handshake.

  Note: TOEs supporting TLS 1.3, but allowing a server to negotiate TLS 1.2, should include all ciphersuites and all extensions as required for either version. If such a TOE is configurable to support only TLS 1.2, only TLS 1.3, or both TLS 1.2 and TLS 1.3, Test 3 should be performed in each configuration – with advertised ciphersuites appropriate for the configuration.

  The connection in Test 3 may be established as part of the establishment of a higher-level protocol, e.g., as part of an EAP session.

  It is sufficient to observe the successful negotiation of a ciphersuite to satisfy the intent of the test; it is not necessary to examine the characteristics of the encrypted traffic in an attempt to discern the ciphersuite being used (for example, that the cryptographic algorithm is 128-bit AES and not 256-bit AES).

- Test 4: (obsolete versions) The evaluator shall perform the following tests:
  - Test 4.1: For each of SSL version 2, SSL version 3, TLS version 1.0, and TLS version 1.1, the evaluator shall initiate a TLS connection from the TOE to a test TLS server that is

configured to negotiate the obsolete version and observe that the TSF terminates the connection.

Note: It is preferred that the TSF sends a fatal error alert message (e.g., protocol version, insufficient security) in response to this, but it is acceptable that the TSF terminates the connection silently (i.e., without sending a fatal error alert).

- Test 4.2: The evaluator shall attempt to establish a connection with a test TLS server that is configured to send a server hello message indicating the selected version (referred to as the legacy version in RFC 8446) with a value corresponding to an undefined TLS (legacy) version (e.g., '03 04') and observe that the TSF terminates the connection.

  Note: It is preferred that the TSF sends a fatal error alert message (e.g., protocol version) in response to this, but it is acceptable that the TSF terminates the connection silently (i.e., without sending a fatal error alert).

  Test 4.2 is intended to test the TSF response to non-standard versions, including early proposals for ‘beta TLS 1.3’ versions. RFC 8446 requires the legacy version to have the value '03 03' and specifies TLS 1.3 in the supported versions extension with the value '03 04'. While not a preferred approach, if continued support for a beta TLS 1.3 version is desired and the TSF cannot be configured to reject such versions, another value (e.g., '03 05') can be used in Test 4.2. Implementations of non-standard versions are not tested.

- Test 5: (ciphersuites) The evaluator shall perform the following tests on handling unexpected ciphersuites using a test TLS server sending handshake messages compliant with the negotiated version except as indicated in the test:
  - Test 5.1: (ciphersuite not offered) For each supported version, the evaluator shall attempt to establish a connection with a test TLS server configured to negotiate the supported version and a ciphersuite not included in the client hello and observe that the TOE rejects the connection.

    Note: It is preferred that the TSF sends a fatal error alert message (e.g., handshake failure) in response to this, but it is acceptable that the TSF terminates the connection silently (i.e., without sending a fatal error alert).

    This test is intended to test the TSF’s generic ability to recognize non-offered ciphersuites. If the ciphersuites in the client hello are configurable, the evaluator shall configure the TSF to offer a ciphersuite outside those that are supported and use that ciphersuite in the test. If the TSF ciphersuite list is not configurable, it is acceptable to use a named ciphersuite from the IANA TLS protocols associated with the tested version. Additional special cases of this test for special ciphersuites are performed separately.

  - Test 5.2: (version confusion) For each supported version, the evaluator shall attempt to establish a connection with a test TLS server that is configured to

negotiate the supported version and a ciphersuite that is not associated with that version and observe that the TOE rejects the connection.

Note: It is preferred that the TSF sends a fatal error alert message (e.g., handshake failure) in response to this, but it is acceptable that the TSF terminates the connection silently (i.e., without sending a fatal error alert).

It is intended that Test 5.2 use TLS 1.3 ciphersuites for a server negotiating TLS 1.2. If TLS 1.3 is supported, the test server negotiating TLS 1.3 should select a TLS 1.2 ciphersuite supported by the TOE for TLS 1.2 and matching the client's supported groups and signature algorithm indicated by extensions in the TLS 1.3 client hello. If the TOE is configurable to allow both TLS 1.2 and TLS 1.3 servers, the test server should use ciphersuites offered by the TSF in its client hello message.

- Test 5.3: (null ciphersuite) For each supported version, the evaluator shall attempt to establish a connection with a test TLS server configured to negotiate the null ciphersuite (TLS_NULL_WITH_NULL_NULL) and observe that the TOE rejects the connection.

  Note: It is preferred that the TSF sends a fatal error alert message (e.g., handshake failure, insufficient security) in response to this, but it is acceptable that the TSF terminates the connection silently (i.e., without sending a fatal error alert).

- Test 5.4: (anon ciphersuite) The evaluator shall attempt to establish a TLS 1.2 connection with a test TLS server configured to negotiate a ciphersuite using the anonymous server authentication method and observe that the TOE rejects the connection.

  Note: It is preferred that the TSF sends a fatal error alert message (e.g., handshake failure, insufficient security) in response to this, but it is acceptable that the TSF terminates the connection silently (i.e., without sending a fatal error alert).

  See IANA TLS parameters for available ciphersuites to be selected by the test TLS server. The test ciphersuite should use supported cryptographic algorithms for as many of the other components as possible. For example, if the TSF only supports the ciphersuite TLS_ECDHE_ECDSA_WITH_AES_256_GCM_SHA384, the test server could select TLS_DH_ANON_WITH_AES_256_GCM_SHA_384. Test 5.5: (deprecated encryption algorithm) For each deprecated encryption algorithm (NULL, RC2, RC4, DES, IDEA, and TDES), the evaluator shall attempt to establish a TLS 1.2 connection with a test TLS server configured to negotiate a ciphersuite using the deprecated encryption algorithm and observe that the TOE rejects the connection. Note: It is preferred that the TSF sends a fatal error alert message (e.g., handshake failure, insufficient security) in response to this, but it is acceptable that the TSF terminates the connection silently (i.e., without

sending a fatal error alert). See IANA TLS parameters for available ciphersuites to be tested. The test ciphersuite should use supported cryptographic algorithms for as many of the other components as possible. For example, if the TSF only supports TLS_ECDHE_ECDSA_WITH_AES_256_GCM_SHA384, the test server could select TLS_ECDHE_PSK_WITH_NULL_SHA_384, TLS_RSA_EXPORT_WITH_RC2_CBC_40_MD5, TLS_ECDHE_RSA_WITH_RC4_128_SHA, TLS_DHE_DSS_WITH_DES_CBC_SHA, TLS_RSA_WITH_IDEA_CBC_SHA, and TLS_ECDHE_RSA_WITH_3DES_EDE_CBC_SHA.

- Test 5.5: (deprecated encryption algorithm) For each deprecated encryption algorithm (NULL, RC2, RC4, DES, IDEA, and TDES), the evaluator shall attempt to establish a TLS 1.2 connection with a test TLS server configured to negotiate a ciphersuite using the deprecated encryption algorithm and observe that the TOE rejects the connection.

  Note: It is preferred that the TSF sends a fatal error alert message (e.g., handshake failure, insufficient security) in response to this, but it is acceptable that the TSF terminates the connection silently (i.e., without sending a fatal error alert).

  See IANA TLS parameters for available ciphersuites to be tested. The test ciphersuite should use supported cryptographic algorithms for as many of the other components as possible. For example, if the TSF only supports TLS_ECDHE_ECDSA_WITH_AES_256_GCM_SHA384, the test server could select TLS_ECDHE_PSK_WITH_NULL_SHA_384, TLS_RSA_EXPORT_WITH_RC2_CBC_40_MD5, TLS_ECDHE_RSA_WITH_RC4_128_SHA, TLS_DHE_DSS_WITH_DES_CBC_SHA, TLS_RSA_WITH_IDEA_CBC_SHA, and TLS_ECDHE_RSA_WITH_3DES_EDE_CBC_SHA.

- Test 6: (extensions) For each supported version indicated in the following tests, the evaluator shall establish a connection from the TOE with a test server negotiating the tested version and providing server handshake messages as indicated when performing the following tests for validating proper extension handling:
  - Test 6.1: (signature_algorithms) [conditional] If the TSF supports certificate-based server authentication, the evaluator shall perform the following tests:
    - Test 6.1.1: For each supported version, the evaluator shall initiate a TLS session with a TLS test server and observe that the TSF's client hello includes the signature_algorithms extension with values in conformance with the ST.
    - Test 6.1.2: (TLS 1.2 only) [conditional] If the TSF supports an ECDHE or DHE ciphersuite, the evaluator shall ensure the test TLS server sends a compliant server hello message selecting TLS 1.2 and one of the supported ECDHE or DHE ciphersuites, a compliant server certificate message, and a key exchange message signed using a signature algorithm and hash combination not included

in the client's hello message (e.g., RSA with SHA-1). The evaluator shall observe that the TSF terminates the handshake.

Note: It is preferred that the TSF sends a fatal error alert message (e.g., handshake failure, illegal parameter, decryption error) in response to this, but it is acceptable that the TSF terminates the connection silently (i.e., without sending a fatal error alert).

- Test 6.1.3: [conditional] If TLS 1.3 is supported, the evaluator shall configure the test TLS server to respond to the TOE with a compliant server hello message selecting TLS 1.3 and a server certificate message, but then also sends a certificate verification message that uses a signature algorithm method not included in the signature_algorithms extension. The evaluator shall observe that the TSF terminates the TLS handshake.
- Note: It is preferred that the TSF sends a fatal error alert message (e.g., handshake failure, illegal parameter, bad certificate, decryption error) in response to this, but it is acceptable that the TSF terminates the connection silently (i.e., without sending a fatal error alert).
- Test 6.1.4: [conditional] For all supported versions for which signature_algorithms_cert is not supported, the evaluator shall ensure the test TLS server sends a compliant server hello message for the tested version and a server certificate message containing a valid certificate that represents the test TLS server, but which is signed using a signature and hash combination not included in the TSF's signature_algorithms extension (e.g., a certificate signed using RSA and SHA-1). The evaluator shall observe that the TSF terminates the TLS session.

  Note: It is preferred that the TSF sends a fatal error alert message (e.g., unsupported certificate, bad certificate, decryption error, handshake failure) in response to this, but it is acceptable that the TSF terminates the connection silently (i.e., without sending a fatal error alert).

  Certificate-based server authentication is required unless the TSF only supports TLS with shared PSK. For TLS 1.2, this is the case if only TLS_ECDHE_PSK_WITH_AES_256_GCM_SHA384 as defined in RFC 8442, TLS_DHE_PSK_WITH_AES_256_GCM_SHA384 as defined in RFC 5487, TLS_ECDHE_PSK_WITH_AES_128_GCM_SHA256 as defined in RFC 8442, or TLS_DHE_PSK_WITH_AES_128_GCM_SHA256 as defined in RFC 5487, are supported. For TLS 1.3, this is the case if only PSK handshakes are supported.

- Test 6.2: (signature_algorithms_cert) [conditional] If signature_algorithms_cert is supported, then for each version that uses the signature_algorithms_cert extension, the evaluator shall ensure that the test TLS server sends a compliant server hello message selecting the tested version and indicating certificate-based server authentication.

  The evaluator shall ensure that the test TLS server forwards a certificate message containing a valid certificate that represents the test TLS server, but which is signed by a valid

Certification Authority using a signature and hash combination not included in the TSF's signature_algorithms_cert extension (e.g., a certificate signed using RSA and SHA-1). The evaluator shall confirm the TSF terminates the session.

Note: Support for certificate-based authentication is assumed if the signature_algorithms_cert is supported. For TLS 1.2, a non-PSK ciphersuite, or one of TLS_RSA_PSK_WITH_AES_256_GCM_SHA384 or TLS_RSA_PSK_WITH_AES_128_GCM_SHA256 as defined in RFC 5487 is used to indicate certificate-based server authentication. For TLS 1.3, the test server completes a full handshake, even if a PSK is offered to indicate certificate-based server authentication. If the TSF only supports shared PSK authentication, Test 6.2 is not performed.

For TLS 1.3, the server certificate message is encrypted. The evaluator will configure the test TLS server with the indicated certificate and ensure that the certificate is indeed sent by observing the buffer of messages to be encrypted, or by inspecting one or both sets of logs from the TSF and test TLS server.

It is preferred that the TSF sends a fatal error alert message (e.g., unsupported certificate, bad certificate, decryption error, handshake failure) in response to this, but it is acceptable that the TSF terminates the connection silently (i.e., without sending a fatal error alert).

- Test 6.3: (extended_master_secret) (TLS 1.2 only) The evaluator shall initiate a TLS 1.2 session with a test TLS server configured to compute a master secret according to RFC 5246, section 8.
  The evaluator shall observe that the TSF's client hello includes the extended master secret extension in accordance with RFC 7627, and ensures that the test TLS server does not include the extended master secret extension in its server hello. The evaluator shall observe that the TSF terminates the session.
  Note: It is preferred that the TSF sends a fatal error alert message (e.g., handshake failure) in response to this, but it is acceptable that the TSF terminates the connection silently (i.e., without sending a fatal error alert).
- Test 6.4: (supported_groups) (TLS 1.2 only – for TLS 1.3, testing is combined with testing of the keyshare extension)
  - Test 6.4.1: For each supported group, the evaluator shall initiate a TLS session with a compliant test TLS 1.2 server supporting RFC 7919. The evaluator shall ensure that the test TLS server is configured to select TLS 1.2 and a ciphersuite using the supported group. The evaluator shall observe that the TSF's client hello lists the supported groups as indicated in the ST, and that the TSF successfully establishes the TLS session.
  - Test 6.4.2: [conditional on TLS 1.2 support for ECDHE ciphersuites] The evaluator shall initiate a TLS session with a test TLS server that is configured to use an explicit version of a named EC group supported by the client. The evaluator shall ensure that the test TLS server key exchange message includes the explicit formulation of the group in its key exchange message as indicated in RFC 4492 section 5.4. The evaluator shall confirm that the TSF terminates the session.

Note: It is preferred that the TSF sends a fatal error alert message (e.g., illegal parameter) in response to this, but it is acceptable that the TSF terminates the connection silently (i.e., without sending a fatal error alert).

- Test 7: (TLS 1.3 extensions) [conditional] If the TSF supports TLS 1.3, the evaluator shall perform the following tests. For each test, the evaluator shall observe that the TSF's client hello includes the supported versions extension with the value '03 04' indicating TLS 1.3:
    - Test 7.1: (supported versions) The evaluator shall initiate TLS 1.3 sessions in turn from the TOE to a test TLS server configured as indicated in the sub-tests below:
        - Test 7.1.1: The evaluator shall configure the test TLS server to include the supported versions extension in the server hello containing the value '03 03.' The evaluator shall observe that the TSF terminates the TLS session.
          Note: It is preferred that the TSF sends a fatal error alert message (e.g., illegal parameter, handshake failure, protocol version) in response to this, but it is acceptable that the TSF terminates the connection silently (i.e., without sending a fatal error alert).
        - Test 7.1.2: The evaluator shall configure the test TLS server to include the supported versions extension in the server hello containing the value '03 04' and complete a compliant TLS 1.3 handshake. The evaluator shall observe that the TSF completes the TLS 1.3 handshake successfully.
        - Test 7.1.3: [conditional] If the TSF is configurable to support both TLS 1.2 and TLS 1.3, the evaluator shall follow operational guidance to configure this behavior. The evaluator shall ensure that the test TLS server sends a TLS 1.2 compliant server handshake and observe that the server random does not incidentally include any downgrade messaging. The evaluator shall observe that the TSF completes the TLS 1.2 handshake successfully.
          Note: Enhanced downgrade protection defined in RFC 8446 is optional, and if supported, is tested separately. The evaluator may configure the test server's random, or may repeat the test until the server's random does not match a downgrade indicator.
    - Test 7.2: (supported groups, key shares) The evaluator shall initiate TLS 1.3 sessions in turn with a test TLS server configured as indicated in the following sub-tests:
        - Test 7.2.1: For each supported group, the evaluator shall configure the compliant test TLS 1.3 server to select a ciphersuite using the group. The evaluator shall observe that the TSF sends an element of the group in its client hello key shares extension (after a hello retry message from the test server, if the key share for the group is not included in the initial client hello). The evaluator shall ensure the test TLS server sends an element of the group in its server hello and observes that the TSF completes the TLS handshake successfully.
        - Test 7.2.2: For each supported group, the evaluator shall modify the server hello sent by the test TLS server to include an invalid key share value claiming to be an element the group indicated in the supported groups extension. The evaluator shall observe that the TSF terminates the TLS session.

Note: It is preferred that the TSF sends a fatal error alert message (e.g., illegal parameter) in response to this, but it is acceptable that the TSF terminates the connection silently (i.e., without sending a fatal error alert).
For DHE ciphersuites, a zero value, or a value greater or equal to the modulus is not a valid element. For ECDHE groups, an invalid point contains x and y coordinates of the correct size, but represents a point not on the curve. The evaluator can construct such an invalid point by modifying a byte in the y coordinate of a valid point and verify that the coordinates do not satisfy the curve equation.

- Test 7.3: (PSK support) [conditional] If the TSF supports pre-shared keys, the evaluator shall follow the operational guidance to use pre-shared keys, shall establish a pre-shared key between the TSF and the test TLS server, and initiate TLS 1.3 sessions in turn between the TSF and the test TLS server configured as indicated in the following sub-tests:
  - Test 7.3.1: The evaluator shall configure the TSF to use the pre-shared key and ensure that the test TLS server functions as a compliant TLS 1.3 server. The evaluator shall observe that the TSF's client hello includes the pre_shared_key extension with the valid PSK indicator shared with the test server. The evaluator shall also observe that the TSF's client hello also includes the psk_key_exchange_mode and the post_handshake_auth extensions and that the psk_key_exchange_mode indicates one or more of DHE or ECDHE modes but does not include the PSK-only mode. The evaluator shall observe that the TSF completes the TLS 1.3 handshake successfully in accordance with RFC 8446, to include the TSF sending appropriate key shares for one or more of the supported groups.
    Once the handshake is successful, the evaluator shall cause the test TLS server to send a certificate request and observe that the TSF provides a certificate message and certificate verify message.
    Note: It may be necessary to complete a standard handshake and send a new ticket message from the test TLS server to establish a pre-shared key, or it might be possible to configure the pre-shared key manually via out-of-band mechanisms. This can be performed in conjunction with other testing that is not tested as part of this SFR. It is not required at this time to support emerging standards on establishing PSK, but as such standards are finalized, this FP may be updated to require such support.
    TLS messages after the handshake are encrypted so it may not be possible to observe the certificate and certificate verify messages sent by the TSF directly. The evaluator may need to configure the test TLS server to use an application that requires post-handshake client authentication and terminates the session or otherwise has an observable effect if the certificate is not provided.
  - Test 7.3.2: The evaluator shall attempt to configure the TSF to send early data. If there is no indication from the TSF that this is blocked, the evaluator shall repeat test 5.3.1 with the TSF so configured and observe that the TSF does not send application data prior to receiving the server hello.

Note: Early data will be encrypted under the PSK and received by the test TLS server prior to it sending a server hello message.

- Test 8: (corrupt finished message) For each supported version, the evaluator shall initiate a TLS session from the TOE to a test TLS server that sends a compliant set of server handshake messages, except for sending a modified finished message (modify a byte of the finished message that would have been sent by a compliant server). The evaluator shall observe that the TSF terminates the session and does not complete the handshake by observing that the TSF does not send application data provided to the TLS channel.
- Test 9: (missing finished message) For each supported version, the evaluator shall initiate a session from the TOE to a test TLS server providing a compliant handshake, except for sending a random TLS message (the five byte header indicates a correct TLS message for the negotiated version, but not indicating a finished message) as the final message. The evaluator shall observe that the TSF terminates the session and does not send application data.
  Note: It is preferred that the TSF sends a fatal error alert message (e.g., decryption error) in response to this, but it is acceptable that the TSF terminates the connection silently (i.e., without sending a fatal error alert).
  For TLS 1.2, the modified message is sent after the change_cipher_spec message. For TLS 1.3, the modified message is sent as the last message of the server's second flight of messages.
- Test 10: (unexpected/corrupt signatures within handshake) The evaluator shall perform the following tests, according to the versions supported.
  - Test 10.1: (TLS 1.2 only) [conditional] If the ST indicates support for ECDSA or DSA ciphersuites, the evaluator shall initiate a TLS session with a compliant test TLS server and modify the signature in the server key exchange. The evaluator shall observe that the TSF terminates the session with a fatal alert message (e.g., decrypt error, handshake error).
  - Test 10.2: [conditional] If the ST indicates support for TLS 1.3, the evaluator shall initiate a TLS session between the TOE and a test TLS server that is configured to send a compliant server hello message, encrypted extension message, and certificate message, but will send a certificate verify message with an invalid signature (e.g., by modifying a byte from a valid signature). The evaluator shall confirm that the TSF terminates the session with a fatal error alert message (e.g., bad certificate, decrypt error, handshake error).
  - Test 10.3: (TLS 1.2 only) [conditional] If the ST indicates support for both RSA and ECDSA methods in the signature_algorithm (or, if supported, the signature_algorithms_cert) extension, and if the ST indicates one or more TLS 1.2 ciphersuites indicating each of the RSA and ECDSA methods in its signature components, the evaluator shall choose two ciphersuites: one indicating an RSA signature (cipher 1) and one indictaing an ECDSA signature (cipher 2). The evalutor shall then establish two certificates that are trusted by the TOE: one representing the test TLS 1.2 server using an RSA signature (cert 1) and one representing the test TLS 1.2 server using an ECDSA signature (cert 2). The evaluator shall initiate a TLS session between the TOE and the test TLS 1.2 server that is configured to select cipher 1 and to send cert 2. The evaluator shall verify that the TSF terminates this TLS session. The evaluator shall then initiate a TLS session between the

TOE and the test TLS 1.2 server that is configured to select cipher 2 and to send cert 1. The evaluator shall verify that the TSF also terminates this TLS session.
Note: It is preferred that the TSF sends a fatal error alert message (e.g., bad certificate, decryption error, handshake failure) in response to this, but it is acceptable that the TSF terminates the connection silently (i.e., without sending a fatal error alert).

- Test 11: [conditional] If the TSF supports certificate-based server authentication, then for each supported version, the evaluator will initiate a TLS session from the TOE to the compliant test TLS server configured to negotiate the tested version, and to authenticate using a certificate trusted by the TSF as specified in the following:
    - Test 11.1: (certificate extended key usage purpose) The evaluator shall send a server certificate that contains the Server Authentication purpose in the ExtendedKeyUsage extension and verify that a connection is established. The evaluator shall repeat this test using a different certificate that is otherwise valid and trusted but lacks the Server Authentication purpose in the ExtendedKeyUsage extension and observe the TSF terminates the session.
      Note: This test may be performed as part of certificate validation testing (FIA_X509_EXT.1).
      It is preferred that the TSF sends a fatal error alert message (e.g., bad certificate, decryption error, handshake failure) in response to this, but it is acceptable that the TSF terminates the connection silently (i.e., without sending a fatal error alert).
      Ideally, the two certificates should be similar in regards to structure, the types of identifiers used, and the chain of trust.
    - Test 11.2: (certificate identifiers) For each supported method of matching presented identifiers, and for each name type for which the TSF parses the presented identifiers from the server certificate for the method, the evaluator shall establish a valid certificate trusted by the TSF to represent the test server using only the tested name type. The evaluator shall perform the following sub-tests:
        - Test 11.2.1: The evaluator shall prepare the TSF as necessary to use the matching method and establish reference identifiers for the test server for the tested name type. The evaluator shall ensure the test TLS server sends a certificate with a matching name of the tested name type and observe that the TSF completes the connection.
        - Test 11.2.2: The evaluator shall prepare the TSF as necessary to use the matching method and establish reference identifiers that do not match the name representing the test server. The evaluator shall ensure the test TLS server sends a certificate with a name of the type tested, and observe the TSF terminates the session.

          Note: It is preferred that the TSF sends a fatal error alert message (e.g., bad certificate, unknown certificate) in response to this, but it is acceptable that the TSF terminates the connection silently (i.e., without sending a fatal error alert).

        - Test 11.2.3: [conditional] If the TSF supports wildcards for a DNS, URI, or SVR name type, the evaluator shall prepare the TSF as necessary to use the matching method for the name type, and establish a reference identifier. The evaluator

shall establish a certificate for the test server that includes a wildcard name for the DNS portion of the appropriate name type which matches the reference identifier. The evaluator shall ensure the TLS server sends the certificate containing the wildcard name of the type tested, and observe that the TSF completes the connection.

- Test 11.2.4: [conditional] If the TSF supports a DNS, URI, or SVR name type, but does not support wildcards (in general, or specifically for internationalized names of the specified type), the evaluator shall prepare the TSF as necessary to use the matching method and establish a reference identifier that matches a wildcard name for the DNS portion of the appropriate name type, in accordance with the appropriate RFC, in a certificate representing the server. The evaluator shall ensure the TLS server sends the certificate containing the wildcard name of the type tested, and observe that the TSF terminates the connection.
  Note: If the TSF's ability to support wildcard certificates is configurable, both Test 11.2.3 and Test 11.2.4 are performed under the appropriate configuration. This test is required if the TSF supports internationalized names of the specified type – in this case, the reference identifier only includes an internationalized encoding in the leftmost label. The certificate used is intended to match the certificate as if wildcards were supported and if the wildcard extended to internationalized names.
- Test 11.2.5: [conditional] If the TSF supports wildcards for a DNS, URI, or SVR name type, the evaluator shall prepare the TSF as necessary to use the matching method. The evaluator shall establish a reference identifier and a certificate for the server as indicated in each of the subtests described below. The evaluator shall in turn, ensure the TLS server sends the certificate associated with the reference identifier and observe that the TSF terminates the session.
  - Test 11.2.5.1: The reference identifier contains a DNS portion with two labels, and the certificate includes a name whose DNS portion includes a matching rightmost label and a wildcard in the leftmost label.
  - Test 11.2.5.2: The reference identifier contains a DNS portion with four labels, and the certificate includes a name whose DNS portion includes two rightmost labels matching the reference identifier, and a wildcard in the third (leftmost) label.
  - Test 11.2.5.3: The reference identifier contains a DNS portion with three labels, and the certificate includes a name whose DNS portion includes two rightmost labels matching the reference identifier, and a wildcard in the third (leftmost) label.
- Test 11.2.6: [conditional] If the TSF supports wildcards and supports embedded DNS, URI, or SVR name types in the CN, then for each supported name type, the evaluator shall repeat Test 11.2.3, Test 11.2.4, and Test 11.2.5 using certificates with the prescribed name embedded in the CN.
- Test 11.2.7: [conditional] If the TSF supports IP addresses as an embedded name type in the CN, the evaluator shall establish an IP address as a reference identifier and establish a certificate with a valid DNS name in the subject field,

including a CN whose value is the digital formatting of the octets of the reference identifier. The evaluator shall ensure the server sends the certificate and observe that the TSF successfully completes the session.

- Test 11.2.8: [conditional] If the TSF supports IP addresses and any embedded name type in the CN, the evaluator shall establish an IP address as a reference identifier and establish a certificate with a valid DNS name in the subject field, including a CN whose value is the digital formatting of the octets of the reference identifier (as in Test 11.2.7) except that one of the octets is replaced by the '*' character. The evaluator shall ensure the server sends the certificate and observe that the TSF terminates the session.

- Test 11.3: (mixed identifiers)[conditional] If the TSF supports a name matching method where the TSF performs matching of both CN-encoded name types and SAN names of the same type, then for each such method, and for each such name type, the evaluator shall establish a valid certificate trusted by the TSF to represent the test server using one name for the CN-encoded name type and a different name for the SAN name type The evaluator shall perform the following tests:
  - Test 11.3.1: The evaluator shall follow the operational guidance to configure the TSF to use the name matching method and establish reference identifiers matching only the SAN. The evaluator shall ensure that the test server sends the certificate with the matching SAN and non-matching CN-encoded name, and observe that the TSF completes the connection.
    Note: Configuration of the TSF may depend on the application using TLS.
  - Test 11.3.2: The evaluator shall follow the operational guidance to configure the TSF to use the name matching method and establish reference identifiers matching only the CN-encoded name. The evaluator shall ensure that the test server sends the certificate with the matching SAN name and non-matching CN encoded name, and observe that the TSF terminates the session.
    It is preferred that the TSF sends a fatal error alert message (e.g., bad certificate, unknown certificate) in response to this, but it is acceptable that the TSF terminates the connection silently (i.e., without sending a fatal error alert).
- Test 11.4: (empty certificate) The evaluator shall configure the test TLS server to supply an empty certificate message and verify that the TSF terminates the session. Note: It is preferred that the TSF sends a fatal error alert message (e.g., bad certificate, unknown certificate) in response to this, but it is acceptable that the TSF terminates the connection silently (i.e., without sending a fatal error alert).
- Test 11.5: (invalid certificate) [conditional] If validity exceptions are supported, then for each exception for certificate validity supported, the evaluator shall configure the TSF to allow the exception and ensure the test TLS server sends a certificate that is valid and trusted, except for the allowed exception. The evaluator shall observe that the TSF completes the session.
  Without modifying the TSF configuration, the evaluator shall initiate a new session with the test TLS server that includes an additional validation error, and observe that the TSF terminates the session.

Note: It is preferred that the TSF sends a fatal error alert message (e.g., decode error, bad certificate) in response to this, but it is acceptable that the TSF terminates the connection silently (i.e., without sending a fatal error alert).
The intent of this test is to verify the scope of the exception processing. If verifying certificate status information is claimed as an exception, then this test will verify that a TLS session succeeds when all supported methods for obtaining certificate status information is blocked from the TSF, to include removing any status information that might be cached by the TSF. If the exception is limited to specific certificates (e.g., only leaf certificates are exempt, or only certain leaf certificates are exempt) the additional validation error could be unavailable revocation information for a non exempt certificate (e.g., revocation status information from an intermediate CA is blocked for the issuing CA of an exempt leaf certificate, or revocation information from the issuing CA is blocked for a non-exempt leaf certificate). If the only option for the exception is for all revocation information for all certificates, another validation error from FIA_X509_EXT.1 (e.g., certificate expiration, extended key usage, etc.) may be used.

5.2.6.1.3 TLS Client Support for Mutual Authentication (FCS_TLSC_EXT.2)

TSS

The evaluator shall 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 shall also ensure that the TSS describes any factors beyond configuration that are necessary in order for the client to engage in mutual authentication using X.509v3 certificates.

Guidance

The evaluator shall ensure that the operational guidance includes any instructions necessary to configure the TOE to perform mutual authentication. The evaluator shall also verify that the operational guidance required per FIA_X509_EXT.2.1 includes instructions for configuring the client-side certificates for TLS mutual authentication.

Tests

Tests For each supported TLS version, the evaluator shall perform the following tests:

- Test 12: The evaluator shall establish a TLS connection from the TSF to a test TLS server that negotiates the tested version and which is not configured for mutual authentication (i.e., does not send a Server's Certificate Request (type 13) message). The evaluator observes negotiation of a TLS channel and confirms that the TOE did not send a Client's Certificate message (type 11) during handshake.
- Test 13: The evaluator shall establish a connection to a test TLS server with a shared trusted root that is configured for mutual authentication (i.e., it sends a Server's Certificate Request (type 13) message). The evaluator observes negotiation of a TLS channel and confirms that the TOE responds with a non-empty Client's Certificate message (type 11) and Certificate Verify (type 15) message.
- Test 14: [conditional] If the TSF supports post-handshake authentication, the evaluator shall establish a pre-shared key between the TSF and a test TLS 1.3 server. The evaluator shall initiate

a TLS session using the pre-shared key and confirm the TSF and test TLS 1.3 server successfully complete the TLS handshake and both support post-handshake authentication. After the session is successfully established, the evaluator shall initiate a certificate request message from the test TLS 1.3 server. The evaluator shall observe that the TSF receives that authentication request and shall take necessary actions, in accordance with the operational guidance, to complete the authentication request. The evaluator shall confirm that the test TLS 1.3 server receives certificate and certificate verification messages from the TSF over the channel that authenticates the client.
Note: TLS 1.3 certificate requests from the test server and client certificate and certificate verify messages are encrypted. The evaluator confirms that the TSF sends the appropriate messages by examining the messages received at the test TLS 1.3 server and by inspecting any relevant server logs. The evaluator may also take advantage of the calling application to demonstrate that the TOE receives data configured at the test TLS server.

5.2.6.1.4 TLS Client Support Downgrade Protection (FCS_TLSC_EXT.3)

TSS

The evaluator shall review the TSS and confirm that the description of the TLS client protocol includes the downgrade protection mechanism in accordance with RFC 8446 and identifies any configurable features of the TSF needed to meet the requirements. If the ST claims that the TLS 1.1 and below indicator is processed, the evaluator shall confirm that the TSS indicates which configurations allow processing of the downgrade indicator and the specific response of the TSF when it receives the downgrade indicator as opposed to simply terminating the session for the unsupported version.

Guidance

The evaluator shall review the operational guidance and confirm that any instructions to configure the TSF to meet the requirements are included.

Tests

The evaluator shall perform the following tests to confirm the response to downgrade indicators from a test TLS 1.3 server:

- Test 15: [conditional] If the TSF supports TLS 1.3, the evaluator shall initiate a TLS 1.3 session with a test TLS 1.3 server configured to send a compliant TLS 1.2 server hello (not including any TLS 1.3 extensions) but including the TLS 1.2 downgrade indicator ‘44 4F 57 4E 47 52 44 01’ in the last eight bytes of the server random field. The evaluator shall confirm that the TSF terminates the session.
  Note: It is preferred that the TSF send a fatal error alert message (e.g., illegal parameter), but it is acceptable that the TSF terminate the session without sending an error alert.
- Test 16: [conditional] If the TSF supports the TLS 1.1 or below downgrade indicator and if the ST indicates a configuration where the indicator is processed, the evaluator shall follow operational guidance instructions to configure the TSF so it parses a TLS 1.1 handshake to detect and process the TLS downgrade indicator. The evaluator shall initiate a TLS session between the TOE and a test TLS server that is configured to send a TLS 1.1 server hello message with the downgrade indicator ‘44 4F 57 4E 47 52 44 00’ in the last eight bytes of the server random field,

but which is otherwise compliant with RFC 4346. The evaluator shall observe that the TSF terminates the session as described in the ST.
Note: It is preferred that the TSF send a fatal error alert message (illegal parameter or unsupported version), but it is acceptable that the TSF terminate the session without sending an error alert.
Use of the TLS 1.1 and below indicator as a redundant mechanism where there is no configuration that actually processes the value does not require additional testing, since this would be addressed by Test 4.1 for FCS_TLSC_EXT.1.1. This test is only required if the TSF responds differently (e.g., a different error alert) when the downgrade indicator is present than when TLS 1.1 or below is negotiated and the downgrade indicator is not present.

5.2.6.1.5 TLS Client Support for Renegotiation (FCS_TLSC_EXT.4)

TSS

The evaluator shall examine the ST to ensure that TLS renegotiation protections are described in accordance with the requirements. The evaluator shall ensure that any configurable features of the renegotiation protections are identified.

Guidance

The evaluator shall examine the operational guidance to confirm that instructions for any configurable features of the renegotiation protection mechanisms are included.

Tests

The evaluator shall perform the following tests as indicated. One or both of "tls-client-accepts renegotiation" or Test 18 is required, depending on whether the TSF is configurable to reject renegotiation or supports secure renegotiation methods defined for TLS 1.2. If TLS 1.3 is supported, Test 18 is required.

- Test 17: [conditional] If the TSF supports a configuration to accept renegotiation requests for TLS 1.2, the evaluator shall follow any operational guidance to configure the TSF. The evaluator shall perform the following tests:
    - Test 17.1: The evaluator shall initiate a TLS connection with a test server configured to negotiate a compliant TLS 1.2 handshake. The evaluator shall inspect the messages received by the test TLS 1.2 server. The evaluator shall observe that either the “renegotiation_info” field or the SCSV ciphersuite is included in the client hello message during the initial handshake.
    - Test 17.2: For each of the following sub-tests, the evaluator shall initiate a new TLS connection with a test TLS 1.2 server configured to send a renegotiation_info extension as specified, but otherwise complete a compliant TLS 1.2 session:
        - Test 17.2.1: The evaluator shall configure the test TLS 1.2 server to send a renegotiation_info extension whose value indicates a non-zero length. The evaluator shall confirm that the TSF terminates the connection.
          Note: It is preferred that the TSF sends a fatal error alert message (e.g., illegal parameter) in response to this, but it is acceptable that the TSF terminates the connection silently (i.e., without sending a fatal error alert).

- Test 17.2.2: The evaluator shall configure the test TLS 1.2 server to send a compliant renegotiation_info extension and observe the TSF successfully completes the TLS 1.2 connection.
- Test 17.2.3: The evaluator shall initiate a session renegotiation after completing a successful handshake with a test TLS 1.2 server that completes a successful TLS 1.2 handshake (as in Test 17.1) and then sends a hello reset request from the test TLS server with a "renegotiation_info" extension that has an unexpected "client_verify_data" or "server_verify_data" value (modify a byte from a compliant response). The evaluator shall verify that the TSF terminates the connection.

  Note: It is preferred that the TSF sends a fatal error alert message (e.g., illegal parameter, handshake error) in response to this, but it is acceptable that the TSF terminates the connection silently (i.e., without sending a fatal error alert).

- Test 18: [conditional] if the TSF supports a configuration that prevents renegotiation, the evaluator shall perform the following tests:
  - Test 18.1: (TLS 1.2) [conditional] If the TLS supports a configuration to reject TLS 1.2 renegotiation, the evaluator shall follow the operational guidance as necessary to prevent renegotiation. The evaluator shall initiate a TLS session between the so configured TSF and a test TLS 1.2 server that is configured to perform a compliant handshake, followed by a hello reset request. The evaluator shall confirm that the TSF completes the initial handshake successfully but terminates the TLS session after receiving the hello reset request.
    Note: It is preferred that the TSF sends a fatal error alert message (e.g., unexpected message) in response to this, but it is acceptable that the TSF terminates the connection silently (i.e., without sending a fatal error alert).
  - Test 18.2: [conditional] If the TSF supports TLS 1.3, the evaluator shall initiate a TLS session between the TSF and a test TLS 1.3 server that completes a compliant TLS 1.3 handshake, followed by a hello reset message. The evaluator shall observe that the TSF completes the initial TLS 1.3 handshake successfully, but terminates the session on receiving the hello reset message.
    It is preferred that the TSF sends a fatal error alert message (e.g., unexpected message) in response to this, but it is acceptable that the TSF terminates the connection silently (i.e., without sending a fatal error alert).

5.2.6.1.6 TLS Client Support for Session Resumption (FCS_TLSC_EXT.5)

TSS

The evaluator shall examine the ST and confirm that the TLS client protocol description includes a description of the supported resumption mechanisms.

Guidance

The evaluator shall ensure the operational guidance describes instructions for any configurable features of the resumption mechanism.

Tests

The evaluator shall perform the following tests:

- Test 19: For each supported TLS version and for each supported resumption mechanism that is supported for that version, the evaluator shall establish a new TLS session between the TSF and a compliant test TLS server that is configured to negotiate the indicated version and perform resumption using the indicated mechanism. The evaluator shall confirm that the TSF completes the initial TLS handshake and shall cause the TSF to close the session normally. The evaluator shall then cause the TSF to resume the session with the test TLS server using the indicated method and observe that the TSF successfully establishes the session.
  Note: For each method, successful establishment refers to proper use of the mechanism, to include compliant extensions and behavior, as indicated in the referenced RFC.
- Test 20: (TLS 1.3 session id echo) [conditional] If the TSF supports TLS 1.3, the evaluator shall initiate a new TLS 1.3 session with a test TLS server. The evaluator shall cause the test TLS server to send a TLS 1.3 server hello message (or a hello retry request if the TSF doesn't include the key share extension) that contains a different value in the legacy_session_id field, and observe that the TSF terminates the session.
  Note: It is preferred that the TSF sends a fatal error alert message (e.g., illegal parameter) in response to this, but it is acceptable that the TSF terminates the connection silently (i.e., without sending a fatal error alert).

5.2.6.1.7 TLS Client 1.3 Resumption Refinements (FCS_TLSC_EXT.6)

TSS

The evaluator shall examine the TSS to verify that the TLS client protocol description indicates that the PSK exchange requires DHE mode and prohibits sending early data. The evaluator shall examine the TSS to verify it lists all applications that can be secured by TLS 1.3 using pre shared keys and describes how each TLS 1.3 client application ensures data for the application is not sent using early data.

Guidance

The evaluator shall examine the operational guidance to verify that instructions for any configurable features that are required to meet the requirement are included. The evaluator shall ensure the operational guidance includes any instructions required to configure applications so the TLS 1.3 client implementation does not send early data.

Tests

[conditional] For each application that is able to be secured via TLS 1.3 using PSK, the evaluator shall follow operational guidance to configure the application not to send early data. The evaluator shall cause the application to initiate a resumed TLS 1.3 session between the TSF and a compliant test TLS 1.3 server as in Test 19 in FCS_TLSC_EXT.5. The evaluator shall observe that the TSF client hello for TLS 1.3 includes the psk_mode extension with the value psk_dhe_ke and sends a key share value for a supported group. The evaluator shall confirm that early data is not received by the test TLS server. Note: If no applications supported by the TOE provide data to TLS 1.3 that can be sent using PSK, this test is omitted.

5.2.6.1.8 TLS Server Protocol (FCS_TLSS_EXT.1)

TSS

The evaluator shall check the description of the implementation of this protocol in the TSS to ensure the supported TLS versions, features, ciphersuites, and extensions, are specified in accordance with RFC 5246 (TLS 1.2) and RFC 8446 (TLS 1.3 and updates to TLS 1.2) as appropriate. The evaluator shall check the description to see if beta TLS 1.3 versions are supported.

The evaluator shall verify that ciphersuites indicated in FCS_TLSS_EXT.1.2 are included in the description, and that none of the following ciphersuites are supported: ciphersuites indicating 'NULL,' 'RC2,' 'RC4,' 'DES,' 'IDEA,' or 'TDES' in the encryption algorithm component, indicating 'anon,' or indicating MD5 or SHA in the message digest algorithm component.

The evaluator shall verify that the TLS implementation description includes the extensions as required in FCS_TLSS_EXT.1.4.

The evaluator shall confirm that the TLS description includes the number and types of certificates that can be installed to represent the TOE.

Guidance

The evaluator shall check the operational guidance to ensure that it contains instructions on configuring the product so that the TSF conforms to the requirements. If the ST indicates that beta versions of TLS 1.3 are supported for backward compatibility, the evaluator shall ensure that the operational guidance provides instructions for disabling these versions. The evaluator shall review the operational guidance to ensure instructions on installing certificates representing the TOE are provided.

Tests

The evaluator shall perform the following tests:

Test 21: (supported TLS 1.2 configurations) The evaluator shall perform the following tests: Test 21.1: For each supported TLS 1.2 ciphersuite, the evaluator shall send a compliant TLS 1.2 client hello with the highest version or legacy version of 1.2 (value '03 03'), a single entry in the ciphersuites field consisting of the specific ciphersuite, and no supported version extension or key share extension. The evaluator shall observe the TSF’s server hello indicates TLS 1.2 in the highest version or legacy version field, does not include a supported version or key share extension, and indicates the specific ciphersuite in the ciphersuite field. If the ciphersuite requires certificate-based authentication, the evaluator shall observe that the TSF sends a valid certificate representing the TOE and successfully completes the TLS handshake. Note: The ciphersuites TLS_ECDHE_PSK_WITH_AES_256_GCM_SHA384 as defined in RFC 8442, TLS_DHE_PSK_WITH_AES_256_GCM_SHA384 as defined in RFC 5487, TLS_ECDHE_PSK_WITH_AES_128_GCM_SHA256 as defined in RFC 8442, and TLS_DHE_PSK_WITH_AES_128_GCM_SHA256 as defined in RFC 5487, if supported, do not require certificate-based authentication of the server. Test 21.2: (TLS 1.2 support for TLS 1.3 clients) [conditional] If the TSF is configurable to support only TLS 1.2 (or if TLS 1.3 is not supported), and if the TSF supports DHE or ECDHE ciphersuites, the evaluator shall follow any operational guidance instructions necessary to configure the TSF to only support TLS 1.2. For each supported TLS 1.2 ciphersuite with DHE or ECDHE indicated as the key exchange method, the evaluator shall send a client

hello with the highest version or legacy version of 1.2 (value '03 03'), a list of ciphersuites consisting of one or more TLS 1.3 ciphersuites followed by the specific TLS 1.2 ciphersuite and no other TLS 1.2 ciphersuites in the ciphersuites field, and including a TLS 1.3 supported group and key share extension with consistent values. The evaluator shall observe that the TSF's server hello indicates TLS 1.2 in the highest version or legacy version field, does not include a supported version or key share extension, and indicates the specific TLS 1.2 ciphersuite in the ciphersuite field. The evaluator shall observe that the TSF completes the TLS 1.2 handshake successfully. Note: Supported ciphersuites using RSA key exchange should not be included in this test. The supported groups extension sent by the test TLS client should be consistent with the TLS 1.2 ciphersuite (e.g., it should be an EC group if the ciphersuite is ECDHE). Test 21.3: (TLS 1.3 support) [conditional] If the TSF supports TLS 1.3, then for each supported TLS 1.3 ciphersuite and key exchange group, the evaluator shall send a compliant TLS 1.3 client hello indicating a list of one or more TLS 1.2 ciphersuites followed by the specific TLS 1.3 ciphersuite and no other ciphersuites in the ciphersuites field, a supported version extension indicating TLS 1.3 (value '03 04') only, a supported groups extension indicating the selected group, and a key share extension containing a value representing an element of the specific group. The evaluator shall observe the TSF's server hello contains the supported versions extension indicating TLS 1.3, the specific ciphersuite in the selected ciphersuite field, and a key share extension containing an element of the specific supported group. The evaluator shall observe that the TSF completes the TLS 1.3 handshake successfully.

Test 22: (obsolete versions) The evaluator shall perform the following tests: Test 22.1: For each of SSL version 2, SSL version 3, TLS version 1.0, and TLS version 1.1, the evaluator shall send a client hello to the TSF indicating the selected version as the highest version. The evaluator shall observe the TSF terminates the connection. Note: It is preferred that the TSF sends a fatal error alert message (e.g., protocol version, insufficient security) in response to this, but it is acceptable that the TSF terminates the connection silently (i.e., without sending a fatal error alert). Test 22.2: The evaluator shall follow the operational guidance to configure the TSF to ensure any supported beta TLS 1.3 versions are disabled, as necessary. The evaluator shall send the TSF a client hello message indicating the supported version (referred to as the legacy version in RFC 8446) with the value '03 04' and observe that the TSF responds with a server hello indicating the highest version supported. Note: Test 22.2 is intended to test the TSF response to non-standard versions, including beta versions of TLS 1.3. If the TSF supports such beta versions, the evaluator shall follow the operational guidance instructions to disable them prior to conducting Test 22.2. Some TLS 1.3 implementations ignore the legacy version field and only check for the supported_versions extension to determine TLS 1.3 support by a client. It is preferred that the legacy version field should still be set to a standard version ('03 03') in the server hello, but it is acceptable that presence of the supported_versions indicating TLS 1.3 (value '03 04') overrides the legacy_version indication to determine highest supported version. Test 23: (ciphersuites) The evaluator shall perform the following tests on handling unexpected ciphersuites using a test TLS client sending handshake messages compliant with the negotiated version except as indicated in the test: Test 23.1: (ciphersuite not supported) For each supported version, the evaluator shall follow the operational guidance, if available, to configure the TSF to disable a supported ciphersuite. The evaluator shall send a compliant client hello to the TSF indicating support for the specific version and a ciphersuites field containing this single disabled ciphersuite. The evaluator shall observe that the TOE rejects the connection. Note: It is preferred that the TSF sends a fatal error alert message (e.g., handshake failure) in response to this, but it is acceptable that the TSF terminates the connection silently (i.e., without sending a fatal error alert). If the TSF's ciphersuites are not configurable, it is acceptable to use a named

ciphersuite from the IANA TLS protocols associated with the tested version. Additional special cases of this test for special ciphersuites are performed separately. Test 23.2: (version confusion) For each supported version, the evaluator shall send a client hello that is compliant for the specific version that includes a list of ciphersuites consisting of a single ciphersuite not associated with that version. The evaluator shall observe that the TOE rejects the connection. Note: It is preferred that the TSF sends a fatal error alert message (e.g., handshake failure) in response to this, but it is acceptable that the TSF terminates the connection silently (i.e., without sending a fatal error alert). It is preferred that Test 23.2 use TLS 1.3 ciphersuites for a server negotiating TLS 1.2. If TLS 1.3 is supported, Test 23.2 also includes a server negotiating TLS 1.3 with a TLS 1.2 ciphersuite – in this case, the negotiated ciphersuite should be chosen to be one supported by the TOE if negotiating TLS 1.2. If the TOE is configurable to allow both TLS 1.2 and TLS 1.3 clients (or does so by default), this configuration is used for both the TLS 1.2 and TLS 1.3 iteration of this test; otherwise the TOE is configured to support the negotiated version in each iteration. Test 23.3: (null ciphersuite) For each supported version, the evaluator shall send a client hello indicating support for the version and include a ciphersuite list consisting of only the null ciphersuite (TLS_NULL_WITH_NULL_NULL, with the value '00 00') and observe that the TOE rejects the connection. Note: It is preferred that the TSF sends a fatal error alert message (e.g., handshake failure, insufficient security) in response to this, but it is acceptable that the TSF terminates the connection silently (i.e., without sending a fatal error alert). Test 23.4: (anon ciphersuite) The evaluator shall send the TSF a TLS 1.2 handshake that is compliant, except that the ciphersuites field includes a ciphersuite list consisting only of ciphersuites using the anonymous server authentication method and observe that the TOE rejects the connection. Note: It is preferred that the TSF sends a fatal error alert message (e.g., handshake failure, insufficient security) in response to this, but it is acceptable that the TSF terminates the connection silently (i.e., without sending a fatal error alert). See IANA TLS parameters for available ciphersuites to be included in the client hello. The test ciphersuites list should include ciphersuites using supported cryptographic algorithms in as many of the other components as possible. For example, if the TSF supports the ciphersuite TLS_ECDHE_ECDSA_WITH_AES_256_GCM_SHA384, the evaluator should include TLS_DH_ANON_WITH_AES_256_GCM_SHA_384. Test 23.5: (deprecated encryption algorithm) The evaluator shall send the TSF a TLS 1.2 client hello that is compliant, except that the ciphersuites field is a list consisting only of ciphersuites indicating a deprecated encryption algorithm, including at least one each of NULL, RC2, RC4, DES, IDEA, and TDES. The evaluator shall observe that the TOE rejects the connection. Note: It is preferred that the TSF sends a fatal error alert message (e.g., handshake failure, insufficient security) in response to this, but it is acceptable that the TSF terminates the connection silently (i.e., without sending a fatal error alert). See IANA TLS parameters for available ciphersuites to be included. The test ciphersuite should use supported cryptographic algorithms for as many of the other components as possible. For example, if the TSF supports TLS_ECDHE_ECDSA_WITH_AES_256_GCM_SHA384, the test could include TLS_ECDHE_PSK_WITH_NULL_SHA_384, TLS_RSA_EXPORT_WITH_RC2_CBC_40_MD5, TLS_ECDHE_RSA_WITH_RC4_128_SHA, TLS_DHE_DSS_WITH_DES_CBC_SHA, TLS_RSA_WITH_IDEA_CBC_SHA, and TLS_ECDHE_RSA_WITH_3DES_EDE_CBC_SHA. Test 24: (extensions) Test 24.1: (signature algorithms) [conditional] If the TSF supports certificate-based authentication, then for each supported signature algorithm indicated in the ST, the evaluator shall perform the following sub-tests with certificates that represent the TOE. For each sub-test, the evaluator shall establish a certificate representing the TOE and using a public-private key pair suitable for the specific signature algorithm value, and signed by a certification authority that uses the same signature algorithm. If the

TSF also supports the signature_algorithms_cert extension, then for each value of the signature_algorithms_cert extension, the evaluator shall repeat the sub-tests using a certificate representing the TOE and using a key pair consistent with the signature algorithm, but signed by a certification authority using the signature algorithm specified in the signature_algorithms_cert extension. Note: The TSF supports certificate-based server authentication if the TLS 1.2 supported ciphersuites include ciphersuites other than TLS_ECDHE_PSK_WITH_AES_256_GCM_SHA384 as defined in RFC 8442, TLS_DHE_PSK_WITH_AES_256_GCM_SHA384 as defined in RFC 5487, TLS_ECDHE_PSK_WITH_AES_128_GCM_SHA256 as defined in RFC 8442, and TLS_DHE_PSK_WITH_AES_128_GCM_SHA256 as defined in RFC 5487. If these are the only supported ciphersuites, this test is omitted. For TLS 1.3, certificate-based server authentication, the client hello should not include the PSK extension. The evaluator shall follow operational guidance instructions to provision the TSF with one or more of these certificates as indicated in the following sub-tests: Test 24.1.1: (TLS 1.2) For each supported value of the signature_algorithms extension, the evaluator shall provision a certificate with a key pair compatible with the specific signature_algorithm value and send the TSF a TLS 1.2 client hello that indicates all supported ciphersuites and has a signature_algorithms extension consisting of a single value matching the specific signature algorithm. If the TSF supports signature_algorithms_cert extension, the client hello also contains the value consistent with the previsioned certificate. The evaluator shall observe that the TSF negotiates TLS 1.2 with a TLS 1.2 ciphersuite that is compatible with the signature algorithm, and that it sends a certificate message containing the provisioned certificate with a key pair that is consistent with the specific signature_algorithm value (and signed using the signature_algorithms_cert extension value, if supported). Note: For TLS 1.2, the ciphersuite describes the signature algorithm as RSA or ECDSA and is compatible with the certificate used if the signature algorithm component of the ciphersuite is of the same type as the signature value of the signature_algorithms extension. Test 24.1.2: [conditional] If the TSF supports TLS 1.3, then for each supported value of the signature_algorithm, the evaluator shall provision a certificate with a key pair that is compatible with the specific signature_algorithm value, send a TLS 1.3 client hello that indicates a supported ciphersuite and has a signature_algorithms extension consisting of a single value matching the specific signature algorithm. If the TSF supports the signature_algorithms_cert extension, the client hello also contains a signature_algorithms_cert extension with a value consistent with the provisioned certificate. The evaluator shall observe that the TSF sends a certificate message containing the provisioned certificate consistent with the specific signature_algorithm value (and signed using the signature_algorithms_cert extension value) and a certificate verify message using the signature_algorithms extension value. Note: For TLS 1.3, the certificate message and certificate verify is encrypted. The evaluator confirms the values of these messages as received at the test TLS client, using logs, or using a test TLS client designed to expose the certificates after they are decrypted. It is not necessary to manually verify the signature used in the key exchange message (TLS 1.2) or certificate verify message (TLS 1.3). Test 24.1.3: [conditional] If the ST indicates that the TSF supports provisioning of multiple certificates, the evaluator shall conduct the following sub-tests: Test 24.1.3.1: The evaluator shall repeat Test 24.1.1 with both the provisioned certificate indicated for Test 24.1.1 and a provisioned certificate using a public key that is not consistent with the signature_algorithm value, but signed by a CA using the signature algorithm specified in the client hello. The evaluator shall observe that the TSF's certificate message does not include the certificate that does not match the signature_algorithm value in the client hello. Test 24.1.3.2: [conditional] If the ST also indicates support for TLS 1.3, the evaluator shall similarly repeat Test 24.1.2

with both the provisioned certificate indicated for test Test 24.1.2 and a provisioned certificate with public keys that are not consistent with the signature_algorithm value but which are signed by a CA using the signature_algorithm value specified in the client hello, and observe that the certificate message sent by the TSF does not include the certificate that does not match the value of the signature_algorithm entry in the client hello. Test 24.1.3.3: [conditional] If the ST also indicates support for the signature_algorithms_cert extension, the evaluator shall repeat Test 24.1.3.1 and Test 24.1.3.2 (if TLS 1.3 is supported) using additional provisioned certificates representing the TOE that use public keys consistent with the signature_algorithm value, but which are signed by CAs using signature algorithms that do not match the value of the signature_algorithms_cert in the client hello and observe that the TSF's certificate message does not include the certificate that does not match the signature_algorithms_cert values in the client hello. Test 24.1.4: (TLS 1.2) The evaluator shall provision a certificate as in Test 24.1.1 but shall send a client hello that only offers ciphersuites whose signature component does not match the value of the signature_algorithms extension. The evaluator shall observe that the TSF terminates the handshake. Note: It is preferred that the TSF sends a fatal error alert message (e.g., handshake failure, illegal parameter) in response to this, but it is acceptable that the TSF terminates the connection silently (i.e., without sending a fatal error alert). Test 24.2: (extended master secret): The evaluator shall initiate a TLS 1.2 session with the TSF from a test TLS client for which the client hello does not include the extended master secret extension and observe that the TSF terminates the session. Note: It is preferred that the TSF sends a fatal error alert message (e.g., handshake error) in response to this, but it is acceptable that the TSF terminates the connection silently (i.e., without sending a fatal error alert).

Test 25: (key exchange) The evaluator shall perform the following tests to confirm compliant key exchange: Test 25.1: (TLS 1.2 RSA key exchange) [conditional] If any of the supported TLS 1.2 ciphersuites in the ST includes RSA for the key exchange method, the evaluator shall perform the following sub-tests: Test 25.1.1: For each supported RSA key size, the evaluator shall provision the TSF with a valid certificate that has an RSA public key of that size. The evaluator shall initiate a valid TLS 1.2 handshake from a compliant test TLS 1.2 client and observe that the server certificate message matches the provisioned certificate. Test 25.1.2: For each supported RSA key size, the evaluator shall send the TSF a compliant TLS 1.2 client hello, but in place of the client's key exchange message, the evaluator shall send the TSF a (non-compliant) key exchange message that is properly formatted but uses an invalid EncryptedPreMasterSecret field in the TLS handshake (e.g., modify a byte of a properly computed value). The evaluator shall attempt to complete the handshake using compliant client change cipher spec and finished messages and verify that the TSF terminates the handshake in a manner that is indistinguishable from a finished message error and does not send application data. Note: Mitigations for oracle attacks described in RFC 5246 Appendix D require the TSF to exhibit the same behavior for key exchange failures as it does for finished message failures. It is preferred that the TSF send a fatal decrypt failure error alert at the end of the handshake in both this case and for a finished message error, but it is acceptable that the TSF terminate the session with another error alert, or without sending an error alert in either case. If the failure error alert is not for a decryption failure, the evaluator shall note that the TSF's response agrees with the response observed in the TLS 1.2 iteration of Test 25.2. Test 25.2: For each supported version, the evaluator shall initiate a compliant handshake up through the (implied for TLS 1.3) change cipher spec message. The evaluator shall then send a (non-compliant) client finished handshake message with an invalid 'verify data' value and verify that the server terminates the session and does not send any application data. Note: TLS 1.2 handshakes include explicit change cipher spec

messages, but TLS 1.3 omits the change cipher spec message. If TLS 1.3 is supported, the modified finished message is sent as the final message from the client after receiving the server's second flight of handshake messages [encrypted extensions, (new ticket), (certificate, certificate verify), (certificate request)]. It is preferred that the TSF send a fatal decryption failure error alert, but it is acceptable that the TSF terminate the session using another error alert or without sending an error alert. The finished message is encrypted. The invalid 'verify data' can be constructed by modifying a byte of a compliant finished message payload.

Test 25.3: (TLS 1.2 DHE or ECDHE key exchange) [conditional] If the ST indicates support for DHE or ECDHE ciphersuites for TLS 1.2, then the evaluator shall perform the following sub-tests: Test 25.3.1: [conditional] If the TSF supports DHE ciphersuites and supports DHE parameters that are not specified in the supported groups extension, then for each supported DHE parameter set, the evaluator shall follow the operational guidance to configure the TSF to use the DHE parameters in its key exchange. The evaluator shall then initiate a TLS 1.2 handshake from a test client with a client hello indicating a single DHE ciphersuite. The evaluator shall observe that the TSF key exchange message indicates the configured parameters and ensure that the client key exchange is a valid point for the parameter set. The evaluator shall confirm that the TSF successfully completes the session. The evaluator shall close the session and resend the client hello. After the TSF responds with a valid key exchange message, the evaluator shall send an empty client key exchange message and observe that the TSF terminates the session. Note: It is preferred that the TSF sends a fatal error alert message (e.g., decryption failure, illegal parameter, handshake error) in response to this, but it is acceptable that the TSF terminates the connection silently (i.e., without sending a fatal error alert). Test 25.3.2: [conditional] If the TSF supports DHE ciphersuites and supports DHE groups in the supported groups extension, then for each supported DHE group, the evaluator shall send the TSF a compliant TLS 1.2 client hello indicating a single ciphersuite that is compatible with the group and indicating the group in the supported groups extension. The evaluator shall observe that the TSF negotiates TLS 1.2 using the indicated ciphersuite and that the server key exchange message indicates the specific group. The evaluator shall send the TOE a client key exchange with a valid point in the group and observe that the TSF successfully completes the session. The evaluator shall close the session and resend the client hello. After the TSF responds with a valid key exchange message, the evaluator shall send the TSF a client key exchange with the public key value '0.' The evaluator shall observe that the TSF terminates the session. The evaluator shall send a new client hello including the same ciphersuite but indicating a group not supported by the TSF in the supported groups extension. The evaluator shall observe that the TSF terminates the session. Note: It is preferred that the TSF sends a fatal error alert message (e.g., decryption failure, illegal parameter, handshake error) in response to this, but it is acceptable that the TSF terminates the connection silently (i.e., without sending a fatal error alert). Test 25.3.3: [conditional] If the TSF supports ECDHE ciphersuites (and therefore supports ECDHE groups in the supported groups extension), the evaluator shall send a client hello message indicating a single supported ECDHE ciphersuite and including the supported ECDHE group in the supported groups extension. The evaluator shall observe that the TSF sends a key exchange message with a valid point of the specified group. The evaluator shall send the TSF a client key exchange message to the TSF consisting of a valid element in the supported group and observe that the TSF successfully completes the session. The evaluator shall close the session and resend the client hello. After the TSF sends the valid key exchange message, the evaluator shall send a client key exchange message consisting of an invalid element of the supported group and observe that the TSF terminates the handshake.

The evaluator shall send a third client hello to the TSF indicating the supported ECDHE ciphersuite and including an ECDHE group that is not supported. The evaluator shall observe that the TSF terminates the session. Note: It is preferred that the TSF sends a fatal error alert message (e.g., decryption failure, illegal parameter, handshake error, insufficient security) in response to this, but it is acceptable that the TSF terminates the connection silently (i.e., without sending a fatal error alert). An invalid ECDSA point consists of properly formatted x and y components, but for which the equation of the curve is not satisfied. To obtain an invalid point, the evaluator can modify a byte of the y coordinate value of a valid point and confirm that the point is not on the curve. The IANA TLS parameters website lists registered ECDHE groups for use in selecting a non-supported group. If the TSF supports all registered ECDHE groups, it is acceptable to send the client hello without a supported groups extension. The TSF should reject such a client hello, but it is acceptable for the TSF to default to a supported group. In this case, the TSF passes the test. Test 25.4: (TLS 1.3 key exchange) [conditional] If the TSF supports TLS 1.3, then for each supported group the evaluator shall perform the following sub-tests: Test 25.4.1: The evaluator shall send the TSF a compliant TLS 1.3 client hello indicating a single key share value from the supported group and shall observe that the server hello includes valid elements of the supported group. Test 25.4.2: The evaluator shall send the TSF a TLS 1.3 client hello indicating a supported groups value supported by the TSF but containing a key share extension indicating an element claiming to be in the supported group that does not represent a valid element of the group. The evaluator shall observe that the TSF terminates the session. Note: It is preferred that the TSF sends a fatal error alert message (e.g., illegal parameter, handshake failure, decryption failure) in response to this, but it is acceptable that the TSF terminates the connection silently (i.e., without sending a fatal error alert). For DHE groups, the invalid element may be of the wrong length; for ECDHE groups, the invalid element has coordinates (x and y) that do not satisfy the equation of the elliptic curve. To obtain an invalid ECDHE point, the evaluator can modify a byte of the y coordinate value of a valid point and confirm that the point is not on the curve.

Test 25.5: For each supported version, the evaluator shall initiate a TLS handshake from a test TLS client with compliant handshake messages negotiating the version and supported parameters to include the change cipher spec message (implied for TLS 1.3), but which omits the finished message and instead sends an application message containing random data. The evaluator shall observe that the TSF terminates the connection. Note: It is preferred that the TSF sends a fatal error alert message (e.g., decryption failure) in response to this, but it is acceptable that the TSF terminates the connection silently (i.e., without sending a fatal error alert). Application data is indicated by the TLSCipherText ContentType field having value 23 (application data). The legacy record version '03 03' and length fields should match a valid TLSCipherText message of the same size,

5.2.6.1.9 TLS Server Support for Mutual Authentication (FCS_TLSS_EXT.2)

TSS

The evaluator shall ensure that the TSS description required per FIA_X509_EXT.2.1 includes the use of client-side certificates for TLS mutual authentication, and that the description includes any certificate validation exception rules and the name types supported for matching to reference identifiers for all applications that use TLS. The evaluator shall examine the TSS to ensure that any CN-embedded name types that are used include a description of the encoding and matching rules.

Guidance

The evaluator shall verify that the operational guidance includes instructions for configuring trust stores for client-side certificates used in TLS mutual authentication. The evaluator shall ensure that the operational guidance includes instructions for configuring the server to require mutual authentication of clients using these certificates and for configuring any certificate validation exception rules. The evaluator shall ensure that the operational guidance includes instructions for configuring reference identifiers normalized or matched by the TSF and matching rules for the supported name types.

Tests

The evaluator shall use TLS as a function to verify that the validation rules in FIA_X509_EXT.1 are adhered to and shall perform the tests listed below. The evaluator shall apply the operational guidance to configure the server to require TLS mutual authentication of clients for these tests unless overridden by instructions in the test activity.

Note: TLS 1.3 is a fundamentally different protocol than TLS 1.2, so even though the certificate validation and name checking tests are identical for both versions, it is likely that early deployments of TLS 1.3 may use a different code-base that warrants independent testing. If TLS 1.3 is supported and the evaluator can verify that the TSF uses the same code-base for certificate validation and name checking for both TLS 1.3 and TLS 1.2, it is acceptable that testing be performed for only one version for these tests.

- Test 26: For each supported version, the evaluator shall follow the operational guidance to configure the TOE to require valid client authentication with no exceptions and initiate a TLS session from a compliant TLS test client supporting that version. The evaluator shall ensure that the test client sends a certificate_list structure which has a length of zero. The evaluator shall verify the TSF terminates the session and no application data flows.
  Note: It is preferred that the TSF sends a fatal error alert message (e.g., handshake failure, bad certificaate, unknown certificate, unknown CA) in response to this, but it is acceptable that the TSF terminates the connection silently (i.e., without sending a fatal error alert).
- Test 27: [conditional] If the ST indicates that the TSF supports establishment of a TLS session for missing or invalid certificates, then for each supported version, and for each supported response option for a missing or invalid certificate indicated in FCS_TLSS_EXT.2.3, the evaluator shall configure the TSF according to the operational guidance to respond as indicated for the calling application. The evaluator shall send client handshake messages from a test TLS client as indicated for each sub-test. The evaluator shall perform the following sub-tests:
  - Test 27.1: [conditional]: If the TSF supports non-authenticated session establishment when receiving an empty certificate message, the evaluator shall initiate a TLS handshake from a compliant test TLS client supporting the version and providing a certificate message containing a certificate_list structure of length zero. The evaluator shall confirm that the TSF notifies the calling application that the user is unauthenticated.
    Note: Specific procedures for determining that the calling application is notified will vary based on the application. If an API to the calling application is not available, the evaluator may attempt to configure the calling application to provide a different response (e.g., require authentication for flagged data) for authenticated and non

authenticated users and make a request at the test client that results in a response indicating the application is treating the client as non-authenticated.

- Test 27.2: [conditional] If the TSF supports exceptions for when revocation status information is unavailable, then the evaluator shall follow the operational guidance to attempt to establish a narrowly defined exception for which both exempt and non exempt certificates can be established. The evaluator shall establish a primary certificate chain for the test client that only exhibits the allowed exception and one or more alternate certificate chains for the test client that do not pass the exception rule, as necessary to test the boundaries of the exception rules.
  The evaluator shall follow the operational guidance to remove any cached revocation status information for the test client's primary certificate chain. The evaluator shall initiate a valid TLS session from the test client that presents the primary certificate for the test client, provide any feedback requested by the TSF to confirm the exception, and observe that the TSF allows the certificate and completes the TLS handshake successfully.
  For each alternate certificate chain, the evaluator shall repeat the session initiation from the test client but present the alternate certificate chain and observe that the TSF terminates the session.
  Note: It is preferred that the TSF sends a fatal error alert message (e.g., bad certificate, unknown certificate, access denied, handshake error) in response to this, but it is acceptable that the TSF terminates the connection silently (i.e., without sending a fatal error alert).
  The alternate certificate chains are intended to test the boundaries of the exception rules. For example, if the exception rule indicates that only leaf certificates are exempt, the evaluator will include an alternate certificate chain for which a CA certificate's revocation information is advertised but is not available; if the exception can be configured for an explicit leaf certificate, or particular subjects, an alternate chain will be included that does not include an excepted certificate or subject. If the exception rules can be configured for all certificates having advertised revocation information, an alternate certificate chain can include an expired certificate – only one additional validity failure (e.g., expired certificate) is required in this case. More comprehensive validity failure handling is addressed by testing for FIA_X509_EXT.1.

- Test 28: For each supported version, the evaluator shall configure the TSF to negotiate the version and require client authentication and perform the following steps:
  - For each supported name matching method indicated in the outer selection of FCS_TLSS_EXT.2.4, and for each name type supported by the matching method as indicated in the inner-selections claimed in each outer selection, the evaluator shall establish a valid primary certificate chain with single names for a test client containing only the supported name types and a valid alternate certificate chain with single names indicating a different name of the same type.
  - [conditional] If any of the supported name types include CN encoding of a name type also supported as a SAN entry, the evaluator shall establish additional certificate chains:
    - The evaluator shall establish a primary certificate chain with multiple names, to include a leaf certificate with:

- a SAN entry that matches the name in the primary certificate chain with single names, of the same SAN name type; and
- a CN entry encoding the same SAN type which matches the name in the alternate certificate chain with single names of the CN encoding of the same SAN name type;

▪ The evaluator shall establish an alternate certificate chain with multiple names, to include a leaf certificate with:

- A SAN entry that matches the name in the alternate certificate chain with single names, of the same SAN name type; and
- a CN entry encoding the same SAN type which matches the name in the primary certificate chain with single names, of the CN encoding of the same SAN name type.

o [conditional] If any of the supported name types include CN encoding, the evaluator shall follow the operational guidance to configure the TSF, establishing trust in the root CA for all primary and alternate certificate chains. The evaluator shall configure the TSF and any relevant TOE applications that use TLS for client authentication as necessary to establish reference identifiers that match the names in the client's primary certificate chains with single names, but not matching any of the names in the alternate certificate chains with single names.

o For each primary certificate chain (with single or multiple names), the evaluator shall initiate a TLS session from the test TLS client that is configured to present the primary certificate chain in a certificate message and a valid certificate verify message in response to the server's certificate request message. The evaluator shall confirm that the TSF accepts the certificate and completes the authenticated TLS session successfully.

o For each alternate certificate chain (with single or multiple names), the evaluator shall initiate a TLS session from the test TLS client that is configured to present the alternate certificate chain in a certificate message and a valid certificate verify message in response to the server's certificate request message. The evaluator shall confirm that the TSF terminates the session.

Note: It is preferred that the TSF sends a fatal error alert message (e.g., access denied) in response to this, but it is acceptable that the TSF terminates the connection silently (i.e., without sending a fatal error alert).

The intent of this test is to confirm that for each method that the TSF uses to match name types presented in validated certificates, it is able to recognize both matching and non matching names. Names of special types implicitly encoded in the CN entry of the certificate subject name are especially prone to error since they may only be validated by the issuing CA as a directory name (RDN) type, especially if the issuing CA is unaware of the intended encoding as a different name type. It is a best practice that when the CN is interpreted as an embedded name type other than RDN, an explicitly encoded SAN entry should take precedence.

5.2.6.1.10 TLS Server Support Downgrade Protection (FCS_TLSS_EXT.3)

TSS

The evaluator shall examine the ST and confirm that the TLS description includes details on the session downgrade protections that are supported.

Guidance

The evaluator shall examine the operational guidance to confirm that instructions are included to configure the TSF to support only TLS 1.3 and to provide the associated downgrade indications.

Tests

The evaluator shall follow the operational guidance as necessary to configure the TSF to negotiate only TLS 1.3 and to provide the associated downgrade indications. The evaluator shall send a TLS client hello to the TOE that indicates support for only TLS 1.2. The evaluator shall observe that the TSF sends a server hello with the last eight bytes of the server random value equal to 44 4F 57 4E 47 52 44 01.

5.2.6.1.11 TLS Server Support for Session Resumption (FCS_TLSS_EXT.5)

TSS

The evaluator shall examine the ST and confirm that the TLS server protocol description includes a description of the supported resumption mechanisms.

Guidance

The evaluator shall ensure the operational guidance describes instructions for any configurable features of the resumption mechanism.

Tests

The evaluator shall perform the following tests:

- Test 31: For each supported version, and for each supported resumption method for that version, the evaluator shall establish a compliant initial TLS session with the TOE for the version using the specified method. The evaluator shall close the successful session and initiate resumption using the specified mechanism. The evaluator shall observe that the TSF successfully establishes the resumed session in accordance with the requirements.
- Test 32: For each supported version and each supported resumption method for that version, the evaluator shall send a compliant client hello message supporting only the specific version and indicating support for the resumption method. The evaluator shall allow the TOE and test client to continue with the compliant handshake until resumption information is established but then cause a fatal error to terminate the session. The evaluator shall then send a new client hello in an attempt to resume the session with the resumption information provided and verify that the TSF does not resume the session, but instead either terminates the session or completes a full handshake, ignoring the resumption information.

  Note: For TLS 1.2, resumption information should be established at the point the TSF sends a server hello, either acknowledging the session-based resumption or acknowledging support for ticket-based resumption and sending a new_ticket message. A TLS 1.2 session can then be terminated by sending a modified finished message. For TLS 1.3, the new_ticket message is sent

after the finished message; once received by the client, the session can be terminated by modifying a byte of the encrypted application data.

5.2.6.1.12 TLS Server TLS 1.3 Resumption Requirements (FCS_TLSS_EXT.6)

TSS

The evaluator shall examine the ST to confirm that the TLS description includes details on session resumption for TLS 1.3, describes each application capable of using TLS 1.3 with PSK, and describes how the TSF and application respond to client attempts to use early data (including via logging or observable responses). The evaluator shall confirm that the TLS description shows that only the psk_dhe_ke psk_key_exchange_mode is supported and that early information is ignored.

Guidance

The evaluator shall examine the operational guidance to verify that instructions for any configurable features that are required to meet the requirement are included.

Tests

The evaluator shall follow the operational guidance to configure the TSF to negotiate TLS 1.3 and shall perform the following tests:

- Test 33: The evaluator shall attempt a resumed session (as for FCS_TLSS_EXT.5 Test 31) but using psk_ke mode as the value for the psk_key_exchange_mode in the resumption client hello. The evaluator shall observe that the TSF refuses to resume the session, either by completing a full TLS 1.3 handshake or by terminating the session.

  Note: It is preferred that the TSF sends a fatal error alert message (e.g., illegal parameter) in response to this, but it is acceptable that the TSF terminates the connection silently (i.e., without sending a fatal error alert).

- Test 34: The evaluator shall initiate a resumed session (as for FCS_TLSS_EXT.5 Test 31) with a test TLS 1.3 client attempting to provide early data that provokes a known reaction at the TOE if received. The evaluator shall observe that the TSF does not react to the early data, indicating that the data was ignored.

  Note: The specific early data used may depend on the applications calling the TLS session and should be selected to initiate an observable response in the TSF or calling application as described in the ST. For HTTPS, for example, the early data can be an HTTP POST that updates data at the TOE, which can then be observed via a user interface for the application if the data was posted or via application logging indicating that the operation failed.

5.2.6.1.13 DTLS Client Protocol (FCS_DTLSC_EXT.1)

TSS

The evaluator shall verify that the TSS describes the actions that take place if a message received from the DTLS server fails the integrity check. If both selections are chosen in FCS_DTLSC_EXT.1.7, the evaluator shall verify that the TSS describes when each method is used and whether the behavior is configurable.

Guidance

If the ST indicates the behavior of the TSF on receiving a message from the DTLS server that fails the MAC integrity check is configurable, the evaluator shall verify that the guidance documentation describes instructions for configuring the behavior.

Tests

For each version supported, the evaluator shall establish a connection using a compliant handshake negotiating the version. The evaluator will then cause the test server to send application data with at least one byte in a record message modified from what a compliant test server would send, and verify that the client discards the record or terminates the DTLS session as described in the TSS. If multiple behaviors are supported, the evaluator shall repeat the test for each behavior.

5.2.6.1.14 DTLS Client Support for Mutual Authentication (FCS_DTLSC_EXT.2)

The evaluator shall perform all evaluation activities listed for FCS_TLSC_EXT.2 while ensuring that DTLS (and not TLS) is used in each evaluation activity.

5.2.6.1.15 DTLS Client Downgrade Protection (FCS_DTLSC_EXT.3)

The evaluator shall perform all evaluation activities listed for FCS_TLSC_EXT.3, with the following modifications:

- DTLS (and not TLS) is used in each evaluation activity
- References to FCS_TLSC_EXT.1.1 are replaced with references to FCS_DTLSC_EXT.1.1.
- DTLS clients may silently drop flawed or unexpected messages from a DTLS test server. Therefore, it might be necessary to resend the message multiple times from the DTLS test server according to the appropriate DTLS RFC to get the desired response.
- DTLS clients do not send fatal error alerts, but should generate them for diagnostics if the test DTLS server repeatedly sends the flawed messages indicated in the tests. If the product generates alerts, the evaluator may observe them in logs of the TSF rather than observing them on the line. Otherwise, the evaluator observes the termination of a session (connection state) by verifying that the TSF does not continue to resend messages after the last timeout expires.

5.2.6.1.16 [D]TLS Client Support for Renegotiation (FCS_DTLSC_EXT.4)

The evaluator shall perform all evaluation activities listed for FCS_TLSC_EXT.4, with the following modifications:

- DTLS (and not TLS) is used in each evaluation activity, with references to TLS replaced by references to DTLS.
- DTLS clients may silently drop flawed or unexpected messages from a DTLS test server. Therefore, it might be necessary to resend the message multiple times from the DTLS test server according to the appropriate DTLS RFC to get the desired response.

5.2.6.1.17 DTLS Client Support for Session Resumption (FCS_DTLSC_EXT.5)

The evaluator shall perform all evaluation activities listed for FCS_TLSC_EXT.5, with the following modifications:

- DTLS (and not TLS) is used in each evaluation activity.

- DTLS clients may silently drop flawed or unexpected messages from a DTLS test server. Therefore, it might be necessary to resend the message multiple times from the DTLS test server according to the appropriate DTLS RFC to get the desired response.
- DTLS clients do not send fatal error alerts, but should generate them for diagnostics if the test DTLS server repeatedly sends the flawed messages indicated in the tests. If the product generates alerts, the evaluator may observe them in logs of the TSF rather than observing them on the line. Otherwise, the evaluator observes the termination of a session (connection state) by verifying that the TSF does not continue to resend messages after the last timeout expires.

5.2.6.1.18 DTLS Server Protocol (FCS_DTLSS_EXT.1)

TSS

The evaluator shall verify that the TSS describes how the DTLS client IP address is validated prior to issuing a server hello message.

Guidance

There are no guidance evaluation activities for this element.

Tests

- Test 1: The evaluator shall send a TLS 1.2 client hello message from a test client and observe that the TSF sends a HelloVerifyRequest message. The evaluator shall modify at least one byte in the cookie from the server's HelloVerifyRequest message and include the modified value as a cookie in the test client's second client hello message. The evaluator shall verify that the server rejects the client's handshake message.
- Test 2: [conditional] If the TSF supports DTLS 1.3, the evaluator shall send a TLS 1.3 client hello message from a test client and observe that the TSF sends a HelloRetryRequest message. The evaluator shall modify at least one byte in the cookie from the server's HelloRetryRequest message and include the modified value as a cookie in the test client's second client hello message. The evaluator shall verify that the server rejects the client's handshake message.

5.2.6.1.19 DTLS Server Support for Mutual Authentication (FCS_DTLSS_EXT.2)

The evaluator shall perform all evaluation activities listed for FCS_TLSS_EXT.2, with the following modifications:

- DTLS (and not TLS) is used in each evaluation activity – 'TLS' is replaced with 'DTLS' and references to FCS_TLSS_EXT.2 elements are replaced with the corresponding reference to the FCS_DTLSS_EXT.2 element.
- DTLS servers may silently drop flawed or unexpected messages from a DTLS test client. Therefore, it might be necessary to resend the message multiple times from the DTLS test client according to the appropriate DTLS RFC to get the desired response.
- DTLS servers do not send fatal error alerts, but should generate them for diagnostics if the test DTLS client repeatedly sends the flawed messages indicated in the tests. If the product generates alerts, the evaluator may observe them in logs of the TSF rather than observing them on the line. Otherwise, the evaluator observes the termination of a session (connection state) by verifying that the TSF does not continue to resend messages after the last timeout expires.

5.2.6.1.20 DTLS Server Downgrade Protection (FCS_DTLSS_EXT.3)

The evaluator shall perform the evaluation activities listed for FCS_TLSS_EXT.3, with references to TLS replaced by the equivalent reference to DTLS.

5.2.6.1.21 DTLS Server Support for Session Resumption (FCS_DTLSS_EXT.5)

The evaluator shall perform the evaluation activities listed for FCS_TLSS_EXT.5, with the following modifications:

- DTLS (and not TLS) is used in each evaluation activity.
- DTLS clients may silently drop flawed or unexpected messages from a DTLS test server. Therefore, it might be necessary to resend the message multiple times from the DTLS test server according to the appropriate DTLS RFC to get the desired response.
- DTLS clients do not send fatal error alerts, but should generate them for diagnostics if the test DTLS server repeatedly sends the flawed messages indicated in the tests. If the product generates alerts, the evaluator may observe them in logs of the TSF rather than observing them on the line. Otherwise, the evaluator observes the termination of a session (connection state) by verifying that the TSF does not continue to resend messages after the last timeout expires.

6 TOE Summary Specification (TSS)

This chapter describes the Windows 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:

Table 29 Standard Fields in a Windows 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.

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.[31] 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, FAU_GEN.1(WAN), FAU_GEN.1(VPN), FAU_GEN.1(BT)**: The TOE audit collection is capable of generating audit events for items identified in section **Error! Reference source not found.**, Error! Reference source not found., and Error! Reference source not found.. 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.
- **FAU_SEL.1**: The TSF provides the ability for the authorized administrator to select the events to be audited based upon object identity, user identity, workstation (host identity), event type, and success or failure of the event.

[31] 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.

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 implementation of the Suite B algorithms, including algorithms for AES (128, 192, 256 key sizes)[32], 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). [33]

A calculated value based on the high-resolution CPU cycle counter which fires after every 1024 interrupts (a continuous source providing 16384 bits of entropy).

Random values gathered periodically from the Trusted Platform Module (TPM), (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

[32] Note that the 192-bit key size is not used by Windows but is available to developers.
[33] The Windows OS Loader implements a SP 800-90 AES-CTR-DRBG and passes along 384 bits of entropy to the kernel for CNG to be use during initialization. This DBRG uses the same algorithms to obtain entropy from the CPU cycle counter, TPM, and RDRAND as described above.

computer, effectively serving as an on-demand self-test.[34] 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.[35]

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 30 Cryptographic Algorithm Standards and Validation for Windows 11 (version 24H2)

Cryptographic Operation	Standard	Requirement	Evaluation Method
Encryption/Decryption	FIPS 197 AES	FCS_COP.1(SYM)	NIST CAVP # A7253, # A7249, # A7250, # A7254
	NIST SP 800-38A CBC mode		NIST CAVP # A7253
	NIST SP 800-38C CCM mode		NIST CAVP # A7253, # A7249

[34] 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. Windows will check the entropy returned from the registered sources and halt the machine if it has insufficient quality.
[35] Running Windows in FIPS validated mode is required according to the administrative guidance.

	NIST SP 800-38E XTS mode		NIST CAVP # A7253
	NIST SP 800-38F KW mode		NIST CAVP # A7250
	NIST SP 800-38D GCM mode		NIST CAVP # A7253,
Digital signature (key generation)	FIPS 186-5 RSA	FCS_CKM.1	NIST CAVP # A7253, # A7251
Digital signature (generation)	FIPS 186-5 RSA	FCS_COP.1(SIGN)	NIST CAVP #A7253, # A7254, #A7251
Digital signature (verification)	FIPS 186-5 RSA	FCS_COP.1(SIGN)	NIST CAVP # A7253, # A7251, # A7252, # A7254
Digital signature (key generation)	FIPS 186-4 DSA	FCS_CKM.1 FCS_CKM.1(VPN)	NIST CAVP # A7253
Digital signature (generation and verification)	FIPS 186-4 DSA	Added as a prerequisite of NIST CAVP KAS # A7253, # A7254	NIST CAVP # A7253, # A7254
Digital signature (key generation)	FIPS 186-5 ECDSA	FCS_CKM.1, FCS_CKM.1(WPA), FCS_CKM.1(VPN)	NIST CAVP # A7253, # A7254, # A7251
Digital signature (key generation, signature generation and verification)	FIPS 186-5 ECDSA	FCS_CKM.1, FCS_CKM.1(WPA)	NIST CAVP # A7253, # A7251
Hashing	FIPS 180-4 SHA-1 and SHA-256, SHA-384, SHA-512	FCS_COP.1 (HASH)	NIST CAVP # A7253
Keyed-Hash Message Authentication Code	FIPS 198-2 HMAC	FCS_COP.1(HMAC)	NIST CAVP # A7253, # A7254
Random number generation	NIST SP 800-90 CTR_DRBG	FCS_RBG_EXT.1	NIST CAVP # A7253, # A7254,
Key agreement	NIST SP 800-56A ECDH	FCS_CKM.2	NIST CAVP # A7253, # A7254
Key establishment	NIST SP 800-56B RSA	FCS_CKM.2, FCS_CKM.2(WLAN)	NIST CVL # A7253, # A7251, Tested by the CC evaluation lab[36]
Key-based key derivation	SP800-108		NIST CAVP # A7250, # A7254
IKEv1	SP800-135	FCS_IPSEC_EXT.1	NIST CAVP # A7253
IKEv2	SP800-135	FCS_IPSEC_EXT.1	NIST CAVP # A7253
TLS	SP800-135	FCS_TLSC_EXT.1,	NIST CAVP # A7253

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

		FCS_TLSC_EXT.2(WLAN) FCS_TLSS_EXT.2,FCS_DTLSC_EXT.1 FCS_DTLSS_EXT.1	

Table 31 Cryptographic Algorithm Standards and Validation for Windows 11 (version 23H2)

Cryptographic Operation	Standard	Requirement	Evaluation Method
Encryption/Decryption	FIPS 197 AES	FCS_COP.1(SYM)	NIST CAVP # A7259, # A7255, # A7256, # A7260
	NIST SP 800-38A CBC mode		NIST CAVP # A7259
	NIST SP 800-38C CCM mode		NIST CAVP # A7259, # A7255
	NIST SP 800-38E XTS mode		NIST CAVP #A7259
	NIST SP 800-38F KW mode		NIST CAVP # A7256
	NIST SP 800-38D GCM mode		NIST CAVP # A7259
Digital signature (key generation)	FIPS 186-5 RSA	FCS_CKM.1	NIST CAVP # A7259, # A7257
Digital signature (generation)	FIPS 186-5 RSA	FCS_COP.1(SIGN)	NIST CAVP # A7259, # A7257 # A7260
Digital signature (verification)	FIPS 186-5 RSA	FCS_COP.1(SIGN)	NIST CAVP #A7259, # A7257, # A7258, # A7260
Digital signature (key generation)	FIPS 186-4 DSA	FCS_CKM.1 FCS_CKM.1(VPN)	NIST CAVP # A7259
Digital signature (generation and verification)	FIPS 186-4 DSA	Added as a prerequisite of NIST CAVP KAS # A7259, # A7260	NIST CAVP # A7259, # A7260
Digital signature (key generation)	FIPS 186-5 ECDSA	FCS_CKM.1, FCS_CKM.1(WPA), FCS_CKM.1(VPN)	NIST CAVP # A7259, # A7257, # A7260
Digital signature (key generation, signature generation and verification)	FIPS 186-5 ECDSA	FCS_CKM.1, FCS_CKM.1(WPA)	NIST CAVP # A7259, # A7257
Hashing	FIPS 180-4 SHA-1 and SHA-256, SHA-384, SHA-512	FCS_COP.1 (HASH)	NIST CAVP # A7259
Keyed-Hash Message Authentication Code	FIPS 198-2 HMAC	FCS_COP.1(HMAC)	NIST CAVP # A7259, # A7260

Random number generation	NIST SP 800-90 CTR_DRBG	FCS_RBG_EXT.1	NIST CAVP # A7259, # A7260
Key agreement	NIST SP 800-56A ECDH	FCS_CKM.2	NIST CAVP # A7259, # A7260
Key establishment	NIST SP 800-56B RSA	FCS_CKM.2, FCS_CKM.2(WLAN)	NIST CVL # A7259, # A7257, Tested by the CC evaluation lab[37]
Key-based key derivation	SP800-108		NIST CAVP # A7256 # A7260
IKEv1	SP800-135	FCS_IPSEC_EXT.1	NIST CAVP # A7259
IKEv2	SP800-135	FCS_IPSEC_EXT.1	NIST CAVP #A7259
TLS	SP800-135	FCS_TLSC_EXT.1, FCS_TLSC_EXT.2(WLAN) FCS_TLSS_EXT.2,FCS_DTLSC_EXT.1 FCS_DTLSS_EXT.1	NIST CAVP # A7259

Table 32 Cryptographic Algorithm Standards and Validation for Windows Server 2025

Cryptographic Operation	Standard	Requirement	Evaluation Method
Encryption/Decryption	FIPS 197 AES	FCS_COP.1(SYM)	NIST CAVP # A7265, # A7261, # A7262, # A7266
	NIST SP 800-38A CBC mode		NIST CAVP # A7265
	NIST SP 800-38C CCM mode		NIST CAVP # A7265, # A7261
	NIST SP 800-38E XTS mode		NIST CAVP # A7265
	NIST SP 800-38F KW mode		NIST CAVP # A7262
	NIST SP 800-38D GCM mode		NIST CAVP # A7265
Digital signature (key generation)	FIPS 186-5 RSA	FCS_CKM.1	NIST CAVP # A7265, # A7263
Digital signature (generation)	FIPS 186-5 RSA	FCS_COP.1(SIGN)	NIST CAVP # A7265, # A7263, # A7266
Digital signature (verification)	FIPS 186-5 RSA	FCS_COP.1(SIGN)	NIST CAVP # A7265, # A7263, # A7264, # A7266

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

Digital signature (key generation)	FIPS 186-4 DSA	FCS_CKM.1 FCS_CKM.1(VPN)	NIST CAVP # A7265
Digital signature (generation and verification)	FIPS 186-4 DSA	Added as a prerequisite of NIST CAVP KAS # A7265, # A7266	NIST CAVP # A7265
Digital signature (key generation)	FIPS 186-5 ECDSA	FCS_CKM.1, FCS_CKM.1(WPA), FCS_CKM.1(VPN)	NIST CAVP # A7265, # A7263, # A7266
Digital signature (key generation, signature generation and verification)	FIPS 186-5 ECDSA	FCS_CKM.1, FCS_CKM.1(WPA)	NIST CAVP # A7265, # A7263
Hashing	FIPS 180-4 SHA-1 and SHA-256, SHA-384, SHA-512	FCS_COP.1 (HASH)	NIST CAVP # A7265
Keyed-Hash Message Authentication Code	FIPS 198-2 HMAC	FCS_COP.1(HMAC)	NIST CAVP # A7265, # A7266
Random number generation	NIST SP 800-90 CTR_DRBG	FCS_RBG_EXT.1	NIST CAVP # A7265, # A7266
Key agreement	NIST SP 800-56A ECDH	FCS_CKM.2	NIST CAVP # A7265
Key establishment	NIST SP 800-56B RSA	FCS_CKM.2, FCS_CKM.2(WLAN)	NIST CVL # A7265, # A7263, Tested by the CC evaluation lab[38]
Key-based key derivation	SP800-108		NIST CAVP # A7262, # A7266
IKEv1	SP800-135	FCS_IPSEC_EXT.1	NIST CAVP # A7265
IKEv2	SP800-135	FCS_IPSEC_EXT.1	NIST CAVP # A7265
TLS	SP800-135	FCS_TLSC_EXT.1, FCS_TLSC_EXT.2(WLAN) FCS_TLSS_EXT.2,FCS_DTLSC_EXT.1 FCS_DTLSS_EXT.1	NIST CAVP # A7265

Table 33 Cryptographic Algorithm Standards and Validation for Azure Local (version 24H2)

Cryptographic Operation	Standard	Requirement	Evaluation Method
Encryption/Decryption	FIPS 197 AES	FCS_COP.1(SYM)	NIST CAVP # A7271, # A7267, # A7268, # A7272
	NIST SP 800-38A CBC mode		NIST CAVP # A7271

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

	NIST SP 800-38C CCM mode		NIST CAVP # A7271, # A7267
	NIST SP 800-38E XTS mode		NIST CAVP # A7271
	NIST SP 800-38F KW mode		NIST CAVP # A7268
	NIST SP 800-38D GCM mode		NIST CAVP # A7271
Digital signature (key generation)	FIPS 186-5 RSA	FCS_CKM.1	NIST CAVP # A7271, # A7269
Digital signature (generation)	FIPS 186-5 RSA	FCS_COP.1(SIGN)	NIST CAVP # A7271, # A7269, # A7272
Digital signature (verification)	FIPS 186-5 RSA	FCS_COP.1(SIGN)	NIST CAVP # A7271, A7269, # A7270, # A7272
Digital signature (key generation)	FIPS 186-4 DSA	FCS_CKM.1 FCS_CKM.1(VPN)	NIST CAVP # A7271
Digital signature (generation and verification)	FIPS 186-4 DSA	Added as a prerequisite of NIST CAVP KAS # A7271, # A7272	NIST CAVP # A7271, # A7272
Digital signature (key generation)	FIPS 186-5 ECDSA	FCS_CKM.1, FCS_CKM.1(WPA), FCS_CKM.1(VPN)	NIST CAVP # A7271, # A7269, # A7272
Digital signature (key generation, signature generation and verification)	FIPS 186-5 ECDSA	FCS_CKM.1, FCS_CKM.1(WPA)	NIST CAVP # A7271, # A7269
Hashing	FIPS 180-4 SHA-1 and SHA-256, SHA-384, SHA-512	FCS_COP.1 (HASH)	NIST CAVP # A7271
Keyed-Hash Message Authentication Code	FIPS 198-2 HMAC	FCS_COP.1(HMAC)	NIST CAVP # A7271
Random number generation	NIST SP 800-90 CTR_DRBG	FCS_RBG_EXT.1	NIST CAVP # A7271, # A7272
Key agreement	NIST SP 800-56A ECDH	FCS_CKM.2	NIST CAVP # A7271, A7272
Key establishment	NIST SP 800-56B RSA	FCS_CKM.2, FCS_CKM.2(WLAN)	NIST CVL # A7271, # A7269, Tested by the CC evaluation lab[39]
Key-based key derivation	SP800-108		NIST CAVP # A7268 # A7272
IKEv1	SP800-135	FCS_IPSEC_EXT.1	NIST CAVP # A7271
IKEv2	SP800-135	FCS_IPSEC_EXT.1	NIST CAVP # A7271

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

TLS	SP800-135	FCS_TLSC_EXT.1, FCS_TLSC_EXT.2(WLAN) FCS_TLSS_EXT.2,FCS_DTLSC_EXT.1 FCS_DTLSS_EXT.1	NIST CAVP # A7271

Table 34 Cryptographic Algorithm Standards and Validation for Azure Local (version 23H2)

Cryptographic Operation	Standard	Requirement	Evaluation Method
Encryption/Decryption	FIPS 197 AES	FCS_COP.1(SYM)	NIST CAVP # A7277, # A7273, # A7274, # A7278
	NIST SP 800-38A CBC mode		NIST CAVP # A7277
	NIST SP 800-38C CCM mode		NIST CAVP # A7277, #A7273
	NIST SP 800-38E XTS mode		NIST CAVP # A7277
	NIST SP 800-38F KW mode		NIST CAVP # A7274
	NIST SP 800-38D GCM mode		NIST CAVP # A7277
Digital signature (key generation)	FIPS 186-5 RSA	FCS_CKM.1	NIST CAVP # A7277, # A7275
Digital signature (generation)	FIPS 186-5 RSA	FCS_COP.1(SIGN)	NIST CAVP # A7277, # A7275, # A7278
Digital signature (verification)	FIPS 186-5 RSA	FCS_COP.1(SIGN)	NIST CAVP # A7277, # A7275, # A7276, # A7278
Digital signature (key generation)	FIPS 186-4 DSA	FCS_CKM.1 FCS_CKM.1(VPN)	NIST CAVP # A7277
Digital signature (generation and verification)	FIPS 186-4 DSA	Added as a prerequisite of NIST CAVP KAS # A7277, # A7278	NIST CAVP # A7277, # A7278
Digital signature (key generation)	FIPS 186-5 ECDSA	FCS_CKM.1, FCS_CKM.1(WPA), FCS_CKM.1(VPN)	NIST CAVP # A7277, # A7275, # A7278
Digital signature (key generation, signature generation and verification)	FIPS 186-5 ECDSA	FCS_CKM.1, FCS_CKM.1(WPA)	NIST CAVP # A7277, # A7275
Hashing	FIPS 180-4 SHA-1 and SHA-256, SHA-384, SHA-512	FCS_COP.1 (HASH)	NIST CAVP # A7277
Keyed-Hash Message Authentication Code	FIPS 198-2 HMAC	FCS_COP.1(HMAC)	NIST CAVP # A7277, # A7278

Random number generation	NIST SP 800-90 CTR_DRBG	FCS_RBG_EXT.1	NIST CAVP # A7277, # A7278
Key agreement	NIST SP 800-56A ECDH	FCS_CKM.2	NIST CAVP # A7277, # A7278
Key establishment	NIST SP 800-56B RSA	FCS_CKM.2, FCS_CKM.2(WLAN)	NIST CVL # A7277, # A7275, Tested by the CC evaluation lab[40]
Key-based key derivation	SP800-108		NIST CAVP # A7274, # A7278
IKEv1	SP800-135	FCS_IPSEC_EXT.1	NIST CAVP # A7277
IKEv2	SP800-135	FCS_IPSEC_EXT.1	NIST CAVP # A7277
TLS	SP800-135	FCS_TLSC_EXT.1, FCS_TLSC_EXT.2(WLAN) FCS_TLSS_EXT.2,FCS_DTLSC_EXT.1 FCS_DTLSS_EXT.1	NIST CAVP # A7277

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 **Error! Reference source not found.**. Note that the administrative guidance precludes hibernating the computer and so these keys are not persisted into volatile storage.

Table 35 Types of Keys Used by Windows

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

[40] 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 using the P-256, P-384, and P-521 curves (FIPS 186-5) for TLS, IPsec traffic, and peer authentication.
Asymmetric ECDSA Private Keys	Keys used for the calculation of ECDSA digital signatures using the P-256, P-384, and P-521 curves (FIPS 186-5) for TLS, IPsec traffic and peer authentication.
Asymmetric RSA Public Keys	Keys used for the verification of RSA digital signatures (FIPS 186-5) for IPsec, TLS, Wi-Fi and signed product updates.
Asymmetric RSA Private Keys	Keys used for the calculation of RSA digital signatures (FIPS 186-5) 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 using MODP-2048, MODP-3072, MODP-4096 for Diffie-Hellman key establishment for IKE with only MODP-2048; and ffdhe2048, ffdhe3072, ffdhe4096, ffdhe6144 Diffie-Hellman key establishment for TLS.
ECDH Private and Public values	Private and public values using the P-256, P-384, and P-521 curves in EC Diffie-Hellman key establishment for TLS and IKE.
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 esncryption key used by DPAPI
DRBG seed	eed for the main DRBG, zeroized during reseeding

6.2.3 Networking

6.2.3.1 TLS, HTTPS, DTLS, EAP-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:

Table 36 TLS RFCs Implemented by Windows

RFC #	Name	How Used
2246	The TLS Protocol Version 1.0	Specifies requirements for TLS 1.0.
3268	Advanced Encryption Standard (AES) Ciphersuites for Transport Layer Security (TLS)	Specifies additional ciphersuites implemented by Windows.
3546	Transport Layer Security (TLS) Extensions	Updates RFC 2246 with TLS 1.0 extensions implemented by Windows.

4346	The Transport Layer Security (TLS) Protocol Version 1.1	Specifies requirements for TLS 1.1.
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.
5216	The EAP-TLS Authentication Protocol	The core Extensible Authentication Protocol implementation.
5246	The Transport Layer Security (TLS) Protocol Version 1.2	Obsoletes 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.
8996	Deprecating TLS 1.0 and TLS 1.1	Recommendation to restrict TLS 1.0 and 1.1 versions.
SSL3	The SSL Protocol Version 3	Specifies requirements for SSL3.

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 as defined in RFC 5246,
- TLS_RSA_WITH_AES_256_CBC_SHA as defined in RFC 5246,
- TLS_RSA_WITH_AES_128_CBC_SHA256 as defined in RFC 5246,
- TLS_RSA_WITH_AES_256_CBC_SHA256 as defined in RFC 5246,
- TLS_RSA_WITH_AES_128_GCM_SHA256 as defined in RFC 5288,
- TLS_RSA_WITH_AES_256_GCM_SHA384 as defined in RFC 5288,
- TLS_DHE_RSA_WITH_AES_128_GCM_SHA256 as defined in RFC 5288,
- TLS_DHE_RSA_WITH_AES_256_GCM_SHA384 as defined in RFC 5288,
- TLS_ECDHE_ECDSA_WITH_AES_128_CBC_SHA256 as defined in RFC 5289,

- TLS_ECDHE_ECDSA_WITH_AES_128_GCM_SHA256 as defined in RFC 5289,
- TLS_ECDHE_ECDSA_WITH_AES_256_CBC_SHA384 as defined in RFC 5289,
- TLS_ECDHE_ECDSA_WITH_AES_256_GCM_SHA384 as defined in RFC 5289,
- TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA256 as defined in RFC 5289,
- TLS_ECDHE_RSA_WITH_AES_128_GCM_SHA256 as defined in RFC 5289,
- TLS_ECDHE_RSA_WITH_AES_256_CBC_SHA384 as defined in RFC 5289,
- TLS_ECDHE_RSA_WITH_AES_256_GCM_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 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 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.[41]

The Extensible Authentication Protocol for TLS (EAP-TLS) protocol implementation in Windows is the same implementation as for the TLS client and server in Windows, thus using the same set of options and sources for random numbers. In particular the EAP Master Session Key (MSK) is derived from the TLS master key, with the MSK then being used as the shared key in an IKEv2 connection.

6.2.3.2 Wireless Networking

Windows has native implementations of IEEE 802.11-2012 and IEEE 802.11ac-2013 to provide secure wireless local area networking (Wi-Fi). Windows can use PRF-384 in WPA2 Wi-Fi sessions and generate

[41] The Windows Update client will not include the TLS_DHE_RSA_WITH_AES_128_GCM_SHA256 and TLS_DHE_RSA_WITH_AES_256_GCM_SHA384 ciphersuites in the available ciphersuites when establishing a TLS session.

AES 128-bit keys or use PRF-704 to generate AES 256-bit keys, both utilize the Windows RBG. Windows complies with the IEEE 802.11-2012 and IEEE 802.11ac-2013 standards and interoperates with other devices that implement the standard. Computers running a Windows OS typically have Wi-Fi CERTIFIED Interoperability Certificates from the Wi-Fi Alliance.

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

6.2.3.3 IPsec

The Windows IPsec implementation is an integral part of the Windows operating system ; it conforms to RFC 4301, Security Architecture for the Internet Protocol. This is documented publicly in the Windows protocol documentation at section 7.5.1 IPsec Overview and covers Windows 8, Windows RT, and Server 2012.[42]

Windows implements both RFCS 2409, Internet Key Exchange (IKEv1), and RFC 4306, Internet Key Exchange version 2, (IKEv2).[43] Windows IPsec supports both tunnel mode and transport mode and provides an option for NAT transversal (reference: section 7.5.5, IPsec Encapsulations).[44] The RAS VPN interface uses tunnel mode only.

The Windows IPsec implementation includes a security policy database (SPD), which states how Windows should process network packets. The SPD uses the traffic source, destination and transport protocol to determine if a packet should be transmitted or received, blocked, or protected with IPsec, (reference: 7.5.3, Security Policy Database Structure), based on firewall processing rules.[45] These rules are described in Understanding Firewall Rules and the "Managing IPsec and VPN Connections" section of the Common Criteria *Operational and Administrative Guidance* for this evaluation. In order to prevent unsolicited inbound traffic, an authorized administrator does not need to define a final catch-all rule which will discard a network packet when no other rules in the SPD apply because Windows will discard the packet. The security policy database also includes configuration settings to limit the time and number of sessions before a new key needs to be generated.

Windows implements AES-GCM-128, AES-GCM-256, AES-CBC-128, and AES-CBC-256 as encryption algorithms for the encapsulating security payload (ESP) (reference: section 6, Appendix A, Product Behavior).[46] . However only AES-CBC-128 and AES-CBC-256 can be used for IKEv1 and IKEv2 to protect the encrypted payload. The resulting potential strength of the symmetric key will be 128 or 256 bits of security depending on whether the IPsec VPN client and IPsec VPN server agreed to use a 128 or 256 AES symmetric key to protect the network traffic. Windows implements HMAC-SHA1, HMAC-SHA-256 and HMAC-SHA-384[47] as authentication algorithms for key exchange as well as Diffie-Hellman Groups

---

[42] Also available as [MS-WSO], *Windows System Overview*, page 43 for offline reading.
[43] [MS-IKEE], *Internet Key Exchange Protocol Extensions*, page 8.
[44] [MS-WSO], page 45.
[45] [MS-WPO], page 44.
[46] [MS-IKEE], pages 74 – 75.
[47] Windows truncates the HMAC output as described in RFC 4868 for HMAC-SHA-256 and HMAC-SHA-384 and for HMAC-SHA1-96 as described in RFC 2404.

14, 19, and 20 (reference: section 6, Appendix A, Product Behavior).[48] The IPsec VPN client will propose a cryptosuite to the IPsec VPN server; if the server responds with a cryptosuite that the client supports, the client will use the server's proposed cryptosuite instead. If the IPsec VPN client and server cannot agree on a cryptosuite, either side may terminate the connection attempt.

In order to prevent security being reduced while transitioning from IKE Phase 1 / IKEv2 SA, an authorized administrator must configure the IPsec VPN client such that algorithms with same strength are used for both IKE Phase 1 and Phase 2 as well as for IKEv2 SA and IKEv2 Child SA.

Windows constructs nonces, which are 32-bit random values, as specified in RFC 2408, Internet Security Association and Key Management Protocol (ISAKMP) section 3.13.[49] When a random number is needed for either a nonce or for key agreement, Windows uses a FIPS-validated random bit generator. When requested, the Windows random bit generator can generate 256 or 512 bits for the caller, the probability of guessing a 256 bit value is 1 in 2^{256} and a 512 bit value is 1 in 2^{512}. When generating the security value *x* used in the IKE Diffie-Hellman key exchange, g^x mod p, Windows uses a FIPS validated random number generator to generate 'x' with length 224, 256, or 384 bits for DH groups 14, 19, and 20 respectively. [50] See the TSS section for Error! Reference source not found. for the NIST CAVP validation numbers.

Windows implements peer authentication using 2048 bit RSA certificates,[51] or ECDSA certificates using the P-256 and P-384 curves for both IKEv1 and IKEv2.[52]

While Windows supports pre-shared IPsec keys, it is not recommended due to the potential use of weak pre-shared keys.[53] Windows simply uses the pre-shared key that was entered by the authorized administrator, there is no additional processing on the input data.

Windows operating systems do not implement the IKEv1 aggressive mode option during a Phase 1 key exchange.

Windows will validate certificates as described in section 6.4.1 by comparing the distinguished name (DN) in the certificate to the expected distinguished name in the X.509v3 certificate presented by the VPN gateway and does not require additional configuration. This comparison occurs in the encrypted and authenticated IKE identification payload. The reference identifiers of the remote computer is compared against the presented identifier in either the Subject Alternative Name (SAN) or the Subject Name of the certificate. The reference identifier may be any of the IP address, Distinguished Name (DN) or Fully Qualified Domain Name (FQDN) of the VPN gateway.

Table 37 Windows Implementation of IPsec RFCs

RFC #	Name	How Used

[48] *Ibid*.
[49] [MS-IKEE], page 51.
[50] http://technet.microsoft.com/en-us/library/cc962035.aspx.
[51] [MS-IKEE], page 73.
[52] http://technet.microsoft.com/en-us/library/905aa96a-4af7-44b0-8e8f-d2b6854a91e6.
[53] http://technet.microsoft.com/en-us/library/cc782582(v=WS.10).aspx.

2407	The Internet IP Security Domain of Interpretation for ISAKMP	Integral part of the Windows Internet Key Exchange (IKE) implementation.
2408	Internet Security Association and Key Management Protocol (ISAKMP)	Integral part of the Windows Internet Key Exchange (IKE) implementation.
2409	The Internet Key Exchange (IKE)	Integral part of the Windows Internet Key Exchange (IKE) implementation.
2986	PKCS #10: Certification Request Syntax Specification; Version 1.7	Public key certification requests issued by Windows.
4106	The Use of Galois/Counter Mode (GCM) in IPsec Encapsulating Security Payload (ESP)	Certain IPsec cryptosuites implemented by Windows.
4109	Algorithms for Internet Key Exchange version 1 (IKEv1)	Certain IPsec cryptosuites implemented by Windows.
4301	Security Architecture for the Internet Protocol	Description of the general security architecture for IPsec.
4303	IP Encapsulating Security Payload (ESP)	Specifies the IP Encapsulating Security Payload (ESP) implemented by Windows.
4304	Extended Sequence Number (ESN) Addendum to IPsec Domain of Interpretation (DOI) for Internet Security Association and Key Management Protocol (ISAKMP)	Specifies a sequence number high-order extension that is implemented by Windows.
4306	Internet Key Exchange (IKEv2) Protocol	Integral part of the Windows Internet Key Exchange (IKE) implementation.
4307	Cryptographic Algorithms for Use in the Internet Key Exchange Version 2 (IKEv2)	Certain IPsec cryptosuites implemented by Windows.
4868	Using HMAC-SHA-256, HMAC-SHA-384, and HMAC-SHA-512 with IPsec	Certain IPsec cryptosuites implemented by Windows.
4945	The Internet IP Security PKI Profile of IKEv1/ISAKMP, IKEv2, and PKIX	Integral part of the Windows Internet Key Exchange (IKE) implementation.
5280	Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile	Specifies PKI support implemented by Windows.
5282	Using Authenticated Encryption Algorithms with the Encrypted Payload of the Internet Key Exchange version 2 (IKEv2) Protocol	Certain IPsec cryptosuites implemented by Windows.
5881	Bidirectional Forwarding Detection (BFD) for IPv4 and IPv6 (Single Hop)	Interoperability between IPv4 and IPv6 networks.
5996	Internet Key Exchange Protocol Version 2 (IKEv2)	Integral part of the Windows Internet Key Exchange (IKE) implementation.
6379	Suite B Cryptographic Suites for IPsec	Certain IPsec cryptosuites implemented by Windows.

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,[54] FCS_CKM.1(WPA), FCS_CKM.1(VPN), FCS_CKM.1(VPN), FCS_CKM.2,[55] FCS_CKM.2(WLAN), FCS_CKM.2(VPN), FCS_COP.1(SYM), FCS_COP.1(HASH), FCS_COP.1(SIGN), FCS_COP.1(HMAC), FCS_RBG_EXT.1**: See **Table 30 Cryptographic Algorithm Standards and** through **Table 34 Cryptographic Algorithm Standards and Validation for Azure Local (version 23H2)**.
- **FCS_CKM_EXT.2, FCS_CKM.2(WLAN)**: Windows provides secure key storage for private (asymmetric) keys and other data deemed by an authorized subject, such as the pre-shared key, to require secure storage using DPAPI and the NTFS discretionary access control policy.[56]
- **FCS_CKM_EXT.4**: Windows overwrites critical cryptographic parameters immediately after that data is no longer needed.
- **FCS_CKM_EXT.8**: When Windows initiates a new Bluetooth association it will generate a new key pair for the association.
- **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.1(WLAN), FCS_TLS_EXT.2, FCS_TLS_EXT.2(WLAN), FCS_TLS_EXT.3, FCS_TLS_EXT.4, FCS_EAP_EXT.1**: 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.
- **FCS_DTLS_EXT.1**: The Windows implementation of DTLS 1.0 and DTLS 1.2 is based on underlying SChannel component which implements TLS.
- **FCS_IPSEC_EXT.1**: Windows provides an IPsec implementation as described about in section 6.2.3.3.

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

[54] In the context of this evaluation, Windows will generate RSA and ECC key pairs as part of establishing a TLS session.
[55] In the context of this evaluation, Windows will generate RSA and ECC key pairs as part of establishing a TLS session.
[56] See https://www.niap-ccevs.org/st/st_vid10677-st.pdf and http://www.commoncriteriaportal.org/files/epfiles/st_windows10.pdf.

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:

Table 38 DAC Access Rights and Named Objects

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

Named Object	Access Rights
	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

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 IPsec network connection is authenticated as described in X.509 Certificate Validation and Generation, IPsec and Pre-shared Keys, and IPsec.

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.[57]

6.3.3 Memory Management and Object Reuse

Windows ensures that any previous information content is unavailable upon allocation to subjects and objects. The TSF ensures that resources processed by the kernel or are exported to user-mode processes do not have residual information in the following ways:

- All objects are based on memory and disk storage. Memory allocated for objects, which includes memory allocated for network packets, is either overwritten with all zeros or overwritten with the provided data before being assigned to an object. Read/write pointers prevent reading beyond the space used by the object. Only the exact value of what is most recently written can be read and no more. For varying length objects, subsequent reads only return the exact value that was set, even though the actual allocated size of the object may be greater than this. Objects stored on disk are restricted to only disk space used for that object.
- Subject processes using the IPsec client have associated memory and an execution context. The TSF ensures that the memory associated with subjects is either overwritten with all zeros or overwritten with user data before allocation as described in the previous point for memory allocated to objects. In addition, the execution context (processor registers) is initialized when new threads within a process are created and restored when a thread context switch occurs.
- Network packets processed by IPsec are encrypted in place. In other words, the data to be encrypted is not copied to a separate buffer and then encrypted. The encrypted network packet is encrypted into the same buffer and overwrites the plaintext network packet. The buffers allocated to hold network packets are allocated with enough space to accommodate padding required for encryption. Each network packet is held in its own buffer. There is a list of buffers,

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

one for each packet. A buffer that holds a network packet is not reused for another network packet. After a buffer holding a network packet is no longer in use the memory allocated for the buffer is freed and released back to the TSF.

The above, in combination, will ensure that the memory used for inbound and outbound network packets does not contain data from previous use.

6.3.4 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, FDP_VPN_EXT.1**: Windows provides a VPN client and interfaces for developers to implement their own VPN client.
- **FDP_RIP.2**: The TSF ensures that previous information contents of resources used for new objects are not discernible in the new object via zeroing or overwriting of memory and tracking read/write pointers for disk storage. Every process is allocated new memory and an execution context. Memory is zeroed or overwritten before allocation.

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 can authenticate users based on username and password as well as using a Windows Hello PIN which is backed by a TPM. Windows 11 and Windows Server can also use physical or virtual smart card thus supporting multiple user authentication.

Password-based authentication to Windows 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.

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,[58] which validates certificates as described in RFC 5280, and particular, the specific validation listed in section**Error! Reference source not found.**, 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, e.g. IPsec, 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.

6.4.2 Certificate Storage

In a Windows OS, stored certificates known as *trusted root certificates* are contained in certificate stores. Each user has their own certificate store and there is a certificate store for the computer account; access to a certificate store is managed by the discretionary access control policy in Windows such that only the authorized administrator, i.e., the user or the local administrator, can add or remove entries. Certificates which are used by applications, for example, TLS, are also placed in certificate stores for the user.

In addition to the standard certificate revocation processes, application certificates can be loaded by either using administrative tools such as **certutil.exe**, changes to the trusted root certificates can be made using Certificate Trust Lists.

6.4.3 IPsec and Pre-shared Keys

IPsec is the only protocol in this evaluation which supports the use of pre-shared keys. These keys can range from a-z, A-Z, the numbers 0 – 9, and any special character entered from the keyboard. The length

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

of the pre-shared key can range from 1 to 256 characters, and so the specific length of 22 characters which the protection profile requires is supported.

The IPsec pre-shared key is used as-is without modification by Windows and so the pre-shared key does not use the Windows random number generator. The reasoning for this is that if the user needs to supply a particular key, that specific key should be used. If the user desires a randomized bit string, then the solution is to use a X.509 certificate which will contain a bit string of suitable length and randomness.

6.4.4 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_BLT_EXT.1, FIA_BLT_EXT.2, FIA_BLT_EXT.3, FIA_BLT_EXT.4, FIA_BLT_EXT.6, FIA_BLT_EXT.7**: Windows requires Bluetooth mutual authentication between the Windows device and the remote device prior to any data transfer over the Bluetooth connection because all Bluetooth profiles are disabled without an explicit authorization by the user. After the user explicitly authorizes the Bluetooth pairing then Windows deems the device to be trusted. Windows will also reject any attempts from another Bluetooth device if the address is the same as a device which is already paired. The collection of supported Bluetooth profiles for Windows 11 is documented at *Bluetooth version and profile support in Windows 11*", the profiles for the other Windows operating systems in this evaluation is documented at *Supported Bluetooth profiles*. Windows operates at security mode 2, service level enforced security, and Bluetooth services proffered by Windows are at the “authorization and authentication” level.
- **FIA_PAE_EXT.**1: Windows conforms to IEEE 802.1X as a Port Access Entity acting in the Supplicant role.
- **FIA_PSK_EXT.1**: Windows allows for the use of pre-shared IPsec keys which are directly used to create an IPsec connection. The set of characters for the pre-shared key is a-z, A-Z, the numbers 0 – 9, and any special character entered from the keyboard.
- **FIA_UAU.5**: Windows provides authentication using a username and password.
- **FIA_X509_EXT.1, FIA_X509_EXT.1(WLAN)**: 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, FIA_X509_EXT.2(WLAN), FIA_X509_IPSEC.3:(IPSEC)**: Windows uses X.509 certificates for EAP-TLS exchanges, TLS, DTLS, HTTPS, IPsec, code signing for system software updates, code signing for mobile applications, and code signing for integrity verification.
- **FIA_X509_EXT.4, FIA_X509_EXT.6:** Windows stores trusted certificates in the certificate stores which controls access based on the Windows Discretionary Access Control policy.

6.5 Security Management

The complete set of management functions are described in **Error! Reference source not found.**, 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.

Table 39 General Purpose OS Windows Security Management Functions

#	Management Function	Administrator	User
1.	Enable/disable screen lock and session timeout	√	√
2.	Configure screen lock inactivity timeout and session timeout	√	√
3.	Import keys/secrets into the secure key storage	√	√
4.	Configure local audit storage capacity	√	
5.	Configure minimum password Length	√	
6.	~~Configure minimum number of special characters in password~~	N.A.	N.A.
7.	~~Configure minimum number of numeric characters in password~~	N.A.	N.A.
8.	~~Configure minimum number of uppercase characters in password~~	N.A.	N.A.
9.	~~Configure minimum number of lowercase characters in password~~	N.A.	N.A.
10.	Configure lockout policy for unsuccessful authentication attempts through by implementing timeouts between attempts and by limiting number of attempts during a time period	√	
11.	Configure host-based firewall	√	
12.	Configure name/address of directory server to bind with	√	
13.	Configure name/address of remote management server from which to receive management settings	√	
14.	~~Configure name/address of audit/logging server to which to send audit/logging records~~	N.A.	N.A.
15.	Configure audit rules	√	
16.	Configure name/address of network time server	√	
17.	Enable/disable automatic software update	√	
18.	Configure Wi-Fi interface	√	
19.	Enable/disable Bluetooth interface	√	
20.	Enable/disable local area network interface, configure USB interfaces	√	
21.	Manage Windows Diagnostics settings	√	√
	Configure remote connection inactivity timeout	√	

Table 40 WLAN Client Windows Security Management Functions

#	Management Function	Implemented	Administrator	User
WL-1	configure security policy for each wireless network: • specify the CA(s) from which the TSF will accept WLAN	√	√	√

	authentication server certificate(s), specify the Fully Qualified Domain Names (FQDNs) of acceptable WLAN authentication server certificate(s), • security type, • authentication protocol, • client credentials to be used for authentication, •			
WL-2	specify wireless networks (SSIDs) to which the TSF may connect	√	√	
WL-3	enable/disable wireless network bridging capability (for example, bridging a connection between the WLAN and cellular radios to function as a hotspot) authenticated by pre-shared key, passcode, no authentication	√	√	
WL-4	enable/disable certificate revocation list checking	√	√	O
WL-5	disable ad hoc wireless client-to-client connection capability	√	√	√
WL-6	disable roaming capability	√	√	√
WL-7	enable/disable IEEE 802.1X pre-authentication	√	√	
WL-8	loading X.509 certificates into the TOE	√	√	
WL-9	revoke X.509 certificates loaded into the TOE	√	√	
WL-10	enable/disable and configure PMK caching: • set the amount of time (in minutes) for which PMK entries are cached, • set the maximum number of PMK entries that can be cached	√	√	
WL-11	configure security policy for each wireless network: set wireless frequency band to 2.4 GHz, 5 GHz, and 6 GHz	√	√	√

Table 41 IPsec VPN Client Windows Security Management Functions

Management Task	Local Administrative Interface	Remote Administrative Interface
Specify VPN gateways to use	• PowerShell • User Interface	• Group Policy • MDM

Specify client credentials to use	• PowerShell • User Interface	• Group Policy • MDM
Configuration of IKE protocol versions	• PowerShell • User Interface	• Group Policy • MDM
Configure IKE authentication techniques	• PowerShell • User Interface	• Group Policy • MDM
Configure the cryptoperiod for the established session keys	• PowerShell	• Group Policy • VPN Gateway
Configure certificate revocation check	• PowerShell	• Group Policy
Specify the algorithm suites that may be proposed and accepted during the IPsec exchanges	• PowerShell	• Group Policy
Load X.509v3 certificates	• PowerShell • User Interface	• Group Policy • MDM

Table 42 Bluetooth Windows Security Management Functions

Function	Implemented?	Standard user	Local administrator	Admin Only
BT-1. Configure the Bluetooth trusted channel. • Disable/enable the Discoverable (for BR/EDR) and Advertising (for LE) modes;	Yes	Yes	Yes	No
~~BT-2. Change the Bluetooth device name (separately for BR/EDR and LE);~~	No	N.A.	N.A.	N.A.
~~BT-3. Provide separate controls for turning the BR/EDR and LE radios on and off;~~	No	N.A.	N.A.	N.A.
~~BT-4. Allow/disallow the following additional wireless technologies to be used with Bluetooth: [Wi-Fi, NFC, [assignment: other wireless technologies]];~~	No	N.A.	N.A.	N.A.
~~BT-5. Configure allowable methods of Out of Band pairing (for BR/EDR and LE);~~	No	N.A.	N.A.	N.A.
BT-6. Disable/enable the Discoverable (for BR/EDR) and Advertising (for LE) modes separately;	Yes	No	Yes	Yes
BT-8. Disable/enable the Bluetooth for all Bluetooth services using the Windows Settings pages. (See BT-1 for details.)	Yes	No	Yes	No
~~BT-7. Disable/enable the Connectable mode (for BR/EDR and LE);~~	No	N.A.	N.A.	N.A.
BT-8. Disable/enable all Bluetooth services using the Windows Device Manager and enabling / disabling the BT radio;	Yes	No	Yes	Yes
BT-8. Disable/enable all Bluetooth services using the ServicesAllowedList from the Bluetooth Policy	Yes	No	Yes	Yes

Configuration Service Provider (CSP) managed by a MDM				
~~BT-9. Specify minimum level of security for each pairing (for BR/EDR and LE)~~	No	N.A.	N.A.	N.A.

6.5.1 SFR Summary

- **FMT_MOF_EXT.1, FMT_SMF_EXT.1, FMT_SMF.1(WLAN), FMT_SMF_EXT.1(VPN), FMT_MOF_EXT.1(BT), FMT_SMF_EXT.1(BT)**: 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

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.[59] 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. [60]

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,[61] 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

[59] 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.
[60] The PRNG is seeded by the TPM RBG, the RDRAND instruction and other sources.
[61] Winload.exe, winresume.exe, tcblaunch.exe, tcbloader.dll, 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"
- A physical memory testing application (memtest.exe) to check the physical memory ICs for the machine are working correctly.[62]

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.[63]

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.

[62] This is considered to be a non-operational mode for the evaluation.
[63] 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.[64]

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 fileswhich 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.

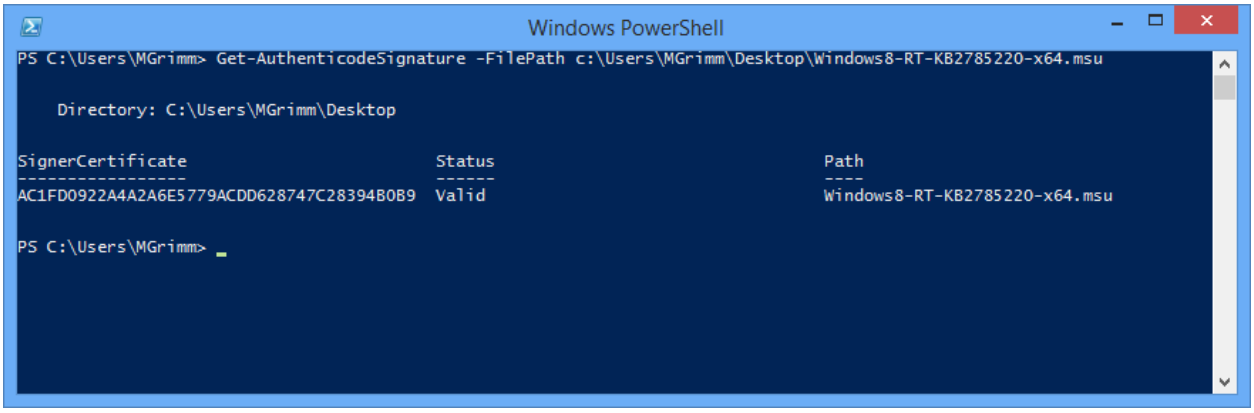
The Windows operating system will check that the certificate is valid and has not been revoked using a standard PKI CRL. Once the Trusted Installer determines that the package is valid, it will update Windows; otherwise the installation will abort and there will be an error message in the event log. 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:

[64] Enforcing the Kernel Mode Code Signing policy is mandatory.



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 Store Applications

Universal Windows Platform (UWP) apps can be downloaded from the Microsoft 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*.[65] The AppX package uses the Open Packaging Conventions (OPC) standard.[66] 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.

[65] Windows Store Applications are typically downloaded from the Windows Store.
[66] 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_BLT_EXT.1**: All Bluetooth profiles are disabled without an explicit authorization by the user.
- **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 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, FPT_TST_EXT.3(WLAN), FPT_TST_EXT.1(VPN)**: 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.

As described in the administrator guidance, an authorized administrator can specify which Wi-Fi networks (SSIDs) a computer may be connected to.

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.
- **FTA_WSE_EXT.1**: An authorized administrator can specify which Wi-Fi networks connect to, as specified in FMT_SMF_EXT.1(WLAN).

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 service 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.
- Using IPsec for remote management of Windows and to connect over a virtual private network (VPN).

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

- PowerShell Remoting: https://docs.microsoft.com/en-US/powershell/scripting/setup/winrmsecurity?view=powershell-6

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

Windows implements IEEE 802.11-2012, IEEE 802.1X and EAP-TLS to provide authenticated wireless networking sessions when requested by the user as described above in **Error! Reference source not found.**.

The specific details for each protocol are described in section Network Protocols.

The Windows implementation of Bluetooth follows the Bluetooth SIG Specification, including OBEX data transfer, RFCOMM, L2CAP, and OPP (object push profile). The OBEX specification, which Windows implements, prevents any transfer of user data until both Bluetooth devices have paired, which requires authorization by the Windows user. When a Windows OS encounters an unpaired device, it does not transfer any data to the unpaired device. When paired to a Bluetooth device will reject connection attempts from other devices that purport to use the same Bluetooth address as the connected device. Windows will attempt to authenticate the device connection using the pre-established link key and if there is a failure of the authentication procedure, or transferring encrypted data, Windows will terminate the device connection and log an entry into the Windows event log.

6.8.1 SFR Summary

- **FTP_ITC_EXT.1(TLS), FTP_ITC_EXT.1(DTLS), FTP_ITC.1(WLAN), FTP_ITC.1(VPN)**: Windows provides several trusted network channels that protect data in transit from disclosure, provide data integrity, and endpoint identification that is used by 802.11-2012, 802.1X, EAP-TLS, TLS, HTTPS, DTLS, and IPsec. 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_BLT_EXT.1, FTP_BLT_EXT.2, FTP_BLT_EXT.3(BR), FTP_BLT_EXT.1(LE)**: The Windows Bluetooth implementation always encrypts data using a key that has at least 128 bits of strength for BR/EDR and LE Bluetooth, Windows may choose to use a key size larger than this minimum as part the Bluetooth pairing.
- **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://www.microsoft.com/en-us/msrc/faqs-report-an-issue?rtc=1), 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://docs.microsoft.com/en-us/security-updates/. If the sender wishes to send secure email, there is a public PGP key for S/MIME at https://www.microsoft.com/en-us/msrc/pgp-key-msrc?rtc=1. 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.

This Security Target is in compliance with the Protection Profile for General Purpose Operating Systems, Version 4.3, September 27, 2022 (GP OS PP), the PP-Module for WLAN Clients, version 1.0, ("WLAN Client Module"), the PP-Module for Virtual Private Network (VPN) Clients, version 2.4, March 31, 2022 ("VPN Client Module") the PP-Module for Bluetooth, version 1.0, April 15, 2021 ("Bluetooth Module"), the Functional Package for Transport Layer Security (TLS), version 2.0, December 19, 2022, ("TLS Module"); the Assurance Package for Flaw Remediation, version 1.0, June 28, 2024, ("ALC_FLR Module");.

For all of the content incorporated from the protection profile or protection profile module, the corresponding rationale in that protection profile, or module, 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, or module, 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.

Table 43 GP OS PP Security Objectives Rationale

Threat or Assumption	Security Objective	Rationale
T.NETWORK_ATTACK	O.PROTECTED_COMMS, O.INTEGRITY, O.MANAGEMENT, O.ACCOUNTABILITY	The threat T.NETWORK_ATTACK is countered by O.PROTECTED_COMMS as this provides for integrity of transmitted data. The threat T.NETWORK_ATTACK is countered by O.INTEGRITY as this provides for integrity of software that is installed onto the system from the network. The threat T.NETWORK_ATTACK is countered by O.MANAGEMENT as this provides for the ability to configure the OS to defend against network attack. The threat T.NETWORK_ATTACK is countered by O.ACCOUNTABILITY as this

		provides a mechanism for the OS to report behavior that may indicate a network attack has occurred.
T.NETWORK_EAVESDROP	O.PROTECTED_COMMS, O.MANAGEMENT	The threat T.NETWORK_EAVESDROP is countered by O.PROTECTED_COMMS as this provides for confidentiality of transmitted data. The threat T.NETWORK_EAVESDROP is countered by O.MANAGEMENT as this provides for the ability to configure the OS to protect the confidentiality of its transmitted data.
T.LOCAL_ATTACK	O.INTEGRITY, O.ACCOUNTABILITY	The objective O.INTEGRITY protects against the use of mechanisms that weaken the TOE with regard to attack by other software on the platform. The objective O.ACCOUNTABILITY protects against local attacks by providing a mechanism to report behavior that may indicate a local attack is occurring or has occurred.
T.LIMITED_PHYSICAL_ACCESS	O.PROTECTED_STORAGE	The objective O.PROTECTED_STORAGE protects against unauthorized attempts to access physical storage used by the TOE.
A.PLATFORM	OE.PLATFORM	The operational environment objective OE.PLATFORM is realized through A.PLATFORM.
A.PROPER_USER	OE.PROPER_USER	The operational environment objective OE.PROPER_USER is realized through A.PROPER_USER.
A.PROPER_ADMIN	OE.PROPER_ADMIN	The operational environment objective OE.PROPER_ADMIN is realized through A.PROPER_ADMIN.

Table 44 VPN Client Module Security Objectives Rationale

Threat or Assumption	Security Objective	Rationale

T.UNAUTHORIZED_ACCESS	O.AUTHENTICATION	The TOE mitigates the threat of unauthorized access by requiring IPsec communications to be properly authenticated.
	O.CRYPTOGRAPHIC_FUNCTIONS	The TOE mitigates the threat of unauthorized access by implementing IPsec using strong cryptographic algorithms.
T.TSF_CONFIGURATION	O.KNOWN_STATE	The TOE mitigates the threat of inadequate configuration by providing a management interface that allows all security-relevant functionality to be configured.
	OE_TRUSTED_CONFIG	This objective mitigates the threat of misconfiguration by ensuring that a malicious actor is not given direct administrative control over the TOE.
T.USER_DATA_REUSE	O.NONDISCLOSURE	The TOE mitigates the threat of data reuse by ensuring that persistently stored data is protected from unauthorized access, non-persistently stored data is appropriately purged, and potentially to ensure that no network traffic is inadvertently transmitted outside of the IPsec tunnel.
T.TSF_FAILURE	O.KNOWN_STATE	The TOE mitigates the threat of TSF failure by enforcing the use of self-tests so that the TOE remains in a known state, and potentially to generate audit records that allow for potential failures to be diagnosed.
A.NO_TOE_BYPASS	OE.NO_TOE_BYPASS	This assumption is satisfied by the environmental objective that ensures network routes do not exist that allow traffic to be transmitted from the TOE system to its intended destination without going through the TOE's IPsec tunnel.
A.PHYSICAL	OE.PHYSICAL	This assumption is satisfied by the environmental objective that ensures the TOE is not deployed

		on a system that is vulnerable to loss of physical custody.
A.TRUSTED_CONFIG	OE.TRUSTED_CONFIG	This assumption is satisfied by the environmental objective that ensures that anyone responsible for administering the TOE can be trusted not to misconfigure it, whether intentionally or not.

Table 45 PP-Module for Bluetooth Security Objectives Rationale

Threat or Assumption	Security Objective	Rationale
T.NETWORK_EAVESDROP	O.PROTECTED_COMMS	The threat T.NETWORK_EAVESDROP is countered by O.PROTECTED_COMMS as this provides the capability to communicate using Bluetooth as a means to maintain the confidentiality of data that are transmitted outside of the TOE.
T.NETWORK_ATTACK	O.PROTECTED_COMMS	The threat T.NETWORK_ATTACK is countered by O.PROTECTED_COMMS as this provides the capability to communicate using Bluetooth as a means to maintain the confidentiality of data that are transmitted outside of the TOE.

Table 46 GP OS PP Tracing Between SFR and TOE Security Objective

Security Objective	Rationale
O.ACCOUNTABILITY	Addressed by: FAU_GEN.1, FTP_ITC_EXT.1 Rationale: FAU_GEN.1 defines the auditable events that must be generated to diagnose the cause of unexpected system behavior. FTP_ITC_EXT.1 provides a mechanism for the TSF to transmit the audit data to a remote system.
O.INTEGRITY	Addressed by: FPT_SBOP_EXT.1, FPT_ASLR_EXT.1, FPT_TUD_EXT.1, FPT_TUD_EXT.2, FCS_COP.1(2), FCS_COP.1(3), FCS_COP.1(4), FPT_ACF_EXT.1, FPT_SRP_EXT.1, FIA_X509_EXT.1, FPT_TST_EXT.1, FTP_ITC_EXT.1, FIA_AFL.1, FIA_UAU.5 Rationale: FPT_SBOP_EXT.1 enforces stack buffer overflow protection that makes it more difficult to exploit running code.

	FPT_ASLR_EXT.1 prevents attackers from exploiting code that executes in static known memory locations. FPT_TUD_EXT.1 and FPT_TUD_EXT.2 enforce integrity of software updates. FCS_COP.1(2), FCS_COP.1(3), and FCS_COP.1(4) provide the cryptographic mechanisms that are used to verify integrity values. FPT_ACF_EXT.1 guarantees the integrity of critical components by preventing unauthorized modifications of them. FPT_SRP_EXT.1 restricts the execution of unauthorized software . FPT_X509_EXT.1 provides X.509 certificates as a way of validating software integrity. FPT_TST_EXT.1 verifies the integrity of stored code. FIA_UAU.5 provides mechanisms that prevent untrusted users from accessing the TSF and FIA_AFL.1 prevents brute-force authentication attempts. FTP_ITC_EXT.1 provides trusted remote communications which makes a remote authenticated session less susceptible to compromise.
O.MANAGEMENT	Addressed by: FMT_MOF_EXT.1, FMT_SMF_EXT.1, FTA_TAB.1, FTP_TRP.1 Rationale: FMT_SMF_EXT.1 defines the TOE's management functions and FMT_MOF_EXT.1 defines the privileges required to invoke them. FTP_TRP.1 provides one or more secure remote interfaces for management of the TSF and FTA_TAB.1 provides actionable warnings against misuse of these interfaces.
O.PROTECTED_STORAGE	Addressed by: FCS_STO_EXT.1, FCS_RBG_EXT.1, FCS_COP.1(1), FDP_ACF_EXT.1 Rationale: FCS_STO_EXT.1 provides a mechanism by which the TOE can designate data as ‘sensitive’ and subsequently require it to be encrypted. FCS_COP.1(1) defines the symmetric algorithm used to encrypt and decrypt sensitive data. FCS_RBG_EXT.1 defines the random bit generator used to create the symmetric keys used to perform this encryption and decryption. FDP_ACF_EXT.1 enforces logical access control on stored data.
O.PROTECTED_COMMS	Addressed by: FCS_TLSC_EXT.1, FCS_TLSC_EXT.2, FCS_TLSC_EXT.3, FCS_TLSC_EXT.4, FCS_DTLS_EXT.1, FCS_RBG_EXT.1, FCS_CKM.1, FCS_CKM.2, FCS_CKM_EXT.4, FCS_COP.1(1), FCS_COP.1(2), FCS_COP.1(3), FCS_COP.1(4), FDP_IFC_EXT.1, FIA_X509_EXT.1, FIA_X509_EXT.2, FTP_ITC_EXT.1 Rationale: FCS_TLSC_EXT.1, FCS_TLSC_EXT.2, FCS_TLSC_EXT.3, and FCS_TLSC_EXT.4 define the ability of the TOE to act as a TLS client as a method of enforcing protected communications. FCS_DTLS_EXT.1 defines the ability of the TOE to act as a DTLS client for the same purpose. FCS_CKM.1, FCS_CKM.2, FCS_COP.1(1), FCS_COP.1(2), FCS_COP.1(3), FCS_COP.1(4), and FCS_RBG_EXT.1 define the cryptographic operations and key lifecycle activity used to support the establishment of protected communications. FIA_X509_EXT.1 defines how the TSF validates x.509 certificates as part of

	establishing protected communications. FIA_X509_EXT.2 defines the trusted communication protocols for which the TOE must perform certificate validation operations. FDP_IFC_EXT.1 defines the extent to which the TSF provides an IPsec VPN as a protected communications method. FTP_ITC_EXT.1 defines the trusted communications channels supported by the TOE.

Table 47 WLAN Client Module Tracing Between SFR and TOE Security Objective

Security Objective	Addressed by	Rationale
O.AUTH_COMM	FCS_TLSC_EXT.1/WLAN	FCS_TLSC_EXT.1/WLAN supports the objective by requiring the TSF to use EAP-TLS to establish a secure connection to a wireless access point, including authentication of the access point.
	FIA_PAE_EXT.1	FIA_PAE_EXT.1 supports the objective by requiring the TSF to act as the supplicant for 802.1X authentication.
	FIA_X509_EXT.1/WLAN	FIA_X509_EXT.1/WLAN supports the objective by defining how the TSF determines the validity of presented X.509 certificates.
	FIA_X509_EXT.2/WLAN	FIA_X509_EXT.2/WLAN supports the objective by requiring the TSF to implement X.509 certificate authentication as the mechanism for authentication EAP-TLS connections.
	FTP_ITC.1/WLAN	FTP_ITC.1/WLAN supports the objective by requiring the TSF to implement trusted protocols that include authentication of the remote endpoints.
	FCS_TLSC_EXT.2/WLAN	FCS_TLSC_EXT.2/WLAN supports the objective by optionally requiring the TSF to support only certain elliptic curves if the TOE implements any EAP-TLS cipher suites that rely on ECDHE as the key establishment method
O.CRYPTOGRAPHIC_FUNCTIONS	FCS_CKM.1/WPA	FCS_CKM.1/WPA supports the objective by requiring the TSF to

		generate symmetric keys used for WPA2 and WPA3 in a specified manner.
	FCS_CKM.2/WLAN	FCS_CKM.2/WLAN supports the objective by requiring the TSF to decrypt group temporal keys used for IEEE 802.11.
	FCS_WPA_EXT.1	FCS_WPA_EXT.1 supports this objective by defining the WPA versions that are supported.
O.TSF_SELF_TEST	FPT_TST_EXT.3/WLAN	FPT_TST_EXT.3/WLAN supports the objective by requiring the TSF to perform self-tests to ensure that it is operating in a known state.
O.SYSTEM_MONITORING	FAU_GEN.1/WLAN	FAU_GEN.1/WLAN supports the objective by requiring the TSF to generate audit records for security-relevant WLAN behavior.
O.TOE_ADMINISTRATION	FIA_X509_EXT.6	FIA_X509_EXT.6 supports the objective by requiring the TSF to securely store certificates in a repository that an administrator can interact with, whether that repository is provided by the WLAN client itself or by a platform storage mechanism defined by the Base-PP portion of the TOE.
	FMT_SMF.1/WLAN	FMT_SMF.1/WLAN supports the objective by requiring the TSF to implement management functionality for security-relevant WLAN behavior.
O.WIRELESS_ACCESS_POINT_CONNECTION	FTA_WSE_EXT.1	FTA_WSE_EXT.1 supports the objective by requiring the TSF to restrict connectivity to allowed wireless networks

Table 48 Tracing Between GP OS PP Security Objective and VPN Client Module SFRs

Security Objective	Rationale	
O.AUTHENTICATION	FIA_X509_EXT.3 (when GPOS PP is Base-PP)	This SFR supports the objective by enforcing the use of X.509

		certificate authentication for IPsec.
	FCS_IPSEC_EXT.1	This SFR supports the objective by requiring the TOE's implementation of IPsec to include requirements for how the remote VPN gateway or peer is authenticated.
	FCS_EAP_EXT.1 (selection-based)	This SFR supports the objective by optionally implementing EAP-TLS or EAP-TTLS as a mechanism for authentication.
O.CRYPTOGRAPHIC_FUNCTIONS	FCS_CKM.1 (refined from GPOS PP)	This SFR supports the objective by requiring that the TOE implement key generation using certain methods.
	FCS_CKM.2 (refined from GPOS PP)	This SFR supports the objective by requiring that the TOE implement key establishment using certain methods.
	FCS_COP.1/1 (refined from GPOS PP)	This SFR supports the objective by requiring that the TOE implement symmetric encryption and decryption using certain methods.
	FTP_ITC.1 (when GPOS PP is Base-PP)	This SFR supports the objective by requiring the TOE to support the use of IPsec as a trusted channel.
	FCS_CKM.1/VPN	This SFR supports the objective by requiring the TOE to generate keys used for IKE using certain methods.
	FCS_IPSEC_EXT.1	This SFR supports the objective by requiring the TOE to implement the IPsec protocol in the specified manner.
	FCS_EAP_EXT.1 (selection-based)	This SFR supports the objective by optionally defining the TOE's implementation of EAP-TLS or EAP-TTLS.
O.KNOWN_STATE	FMT_SMF.1/VPN	This SFR supports the objective by requiring the TOE to implement certain administratively-configurable functions.

	FPT_TST_EXT.1/VPN	This SFR supports the objective by requiring the TOE to execute self-tests that demonstrate that its integrity is maintained.
	FAU_GEN.1/VPN (optional)	This SFR supports the objective by optionally requiring the TOE to generate audit records of its behavior.
	FAU_SEL.1/VPN (optional)	This SFR supports the objective by optionally requiring the TOE to allow for the configuration of what behavior is audited.
O.NONDISCLOSURE	FCS_CKM_EXT.2 (when GPOS PP is Base-PP)	This SFR supports the objective by requiring the TOE to store sensitive data in the OS' key storage
	FDP_RIP.2	This SFR supports the objective by requiring the TOE or its platform to ensure that residual data is purged from the system.
	FDP_VPN_EXT.1 (optional)	This SFR supports the objective by optionally requiring the TOE to prohibit split-tunneling so that network traffic cannot be transmitted outside of an established IPsec tunnel.

Table 49 Tracing Between GP OS PP Security Objective and PP-Module for Bluetooth SFRs[67]

Security Objective	Rationale	
O.ACCOUNTABILITY	FAU_GEN.1(BT)	FAU_GEN.1/BT supports the objective by requiring the TSF to specify the Bluetooth-related auditable events for which it will generate audit records.
O.MANAGEMENT	FMT_MOF_EXT.1(BT)	FMT_MOF_EXT.1/BT supports the objective by restricting the ability to perform Blue-tooth-related management functions to the Administrator.
	FMT_SMF_EXT.1(BT)	FMT_SMF_EXT.1/BT supports the objective by specifying the Bluetooth-related management

[67] This security objective mapping was updated as part of NIAP Technical Decision 685.

		functions that the TSF must perform.
O.PROTECTED_COMMS	FAU_GEN.1(BT)	The PP-Module defines auditable events for Bluetooth that extends the audit functionality defined in each Base-PP.
	FCS_CKM_EXT.8	FCS_CKM_EXT.8 supports the objective by requiring the TSF to specify how ECDH key pairs will be refreshed. This SFR applies to the frequency of key generation activity. This does not conflict with the Base-PP because it involves a key generation mechanism defined in the Base-PP and relates exclusively to Bluetooth functionality so it does not affect any other key generation activities required by the Base-PP.
	FIA_BLT_EXT.1	This SFR applies to the establishment of Bluetooth connectivity, which is behavior not described in or prevented by the Base-PP.
	FIA_BLT_EXT.2	This SFR applies to the establishment of Bluetooth connectivity, which is behavior not described in or prevented by the Base-PP.
	FIA_BLT_EXT.3	This SFR applies to the establishment of Bluetooth connectivity, which is behavior not described in or prevented by the Base-PP.
	FIA_BLT_EXT.4	This SFR applies to the establishment of Bluetooth connectivity, which is behavior not described in or prevented by the Base-PP.
	FIA_BLT_EXT.6	This SFR applies to the establishment of Bluetooth connectivity, which is behavior not described in or prevented by the Base-PP.

	FIA_BLT_EXT.7	This SFR applies to the establishment of Bluetooth connectivity, which is behavior not described in or prevented by the Base-PP.
	FMT_MOF_EXT.1(BT)	This SFR applies to the establishment of Bluetooth connectivity, which is behavior not described in or prevented by the Base-PP.
	FMT_SMF_EXT.1 (when GPOS PP is Base-PP)	This SFR is unchanged from its definition in the Base-PP; the only change required by this PP-Module is how to interpret it in the context of Bluetooth capabilities.
	FMT_SMF_EXT.1(BT)	The ST author is instructed to complete an assignment in the SFR with information related to Bluetooth, and to include additional management functions in this SFR based on the Bluetooth capability defined by the PP-Module.
	FTP_BLT_EXT.1	This SFR applies to encryption of Bluetooth communications. This is a trusted channel that is not discussed in the Base-PP, but it relies on the same cryptographic algorithms specified in the Base-PP to function.
	FTP_BLT_EXT.2	This SFR applies to encryption of Bluetooth communications. This is a trusted channel that is not discussed in the Base-PP, but it relies on the same cryptographic algorithms specified in the Base-PP to function.
	FTP_BLT_EXT.3(BR)	FTP_BLT_EXT.3/BR support the objective by requiring the TSF to implement a minimum encryption key size for Bluetooth BR/EDR. This SFR applies to encryption of Bluetooth communications. This is a trusted channel that is not discussed in the Base-PP, but it

		relies on the same cryptographic algorithms specified in the Base-PP to function.
	FTP_BLT_EXT.3(LE) (selection-based)	FTP_BLT_EXT.3/LE support the objective by requiring the TSF to implement a minimum encryption key size for Bluetooth LE. This SFR applies to encryption of Bluetooth communications. This is a trusted channel that is not discussed in the Base-PP, but it relies on the same cryptographic algorithms specified in the Base-PP to function.

Table 50 WLAN Client Module Consistency Rationale to the GP OS PP

Threat or Assumption	Security Objective	Rationale
T.TSF_FAILURE	O.SELF_TEST	The threat T.TSF_FAILURE is mitigated by O.SELF_TEST as this defines a mechanism for ensuring the reliability of the TSF by detecting potential failure conditions.
T.UNAUTHORIZED_ACCESS	O.AUTH_COMM	The threat T.UNAUTHORIZED_ACCESS is mitigated in part by O.AUTH_COMM by ensuring the authenticity of any remote endpoint that the TSF connects to.
	O.CRYPTOGRAPHIC_FUNCTIONS	The threat T.UNAUTHORIZED_ACCESS is mitigated in part by O.CRYPTOGRAPHIC_FUNCTIONS by ensuring the confidentiality and integrity of data in transit to protect against man-in-the-middle attacks.
	O.TOE_ADMINISTRATION	The threat T.UNAUTHORIZED_ACCESS is mitigated in part by O.TOE_ADMINISTRATION by using the TOE platform's

		authentication mechanism to ensure that only authorized administrators can configure the TOE's behavior.
	O.WIRELESS_ACCESS_POINT_ CONNECTION	The threat T.UNAUTHORIZED_ACCESS is mitigated in part by this objective because it provides a mechanism to restrict the remote entities that the TOE is permitted to communicate with.
T.UNDETECTED_ACTIONS	O.SYSTEM_ MONITORING	The threat T.UNDETECTED_ACTIONS is mitigated by O.SYSTEM_MONITORING by enforcing an auditing Mechanism that can be used to track security-relevant TOE behavior.
A.NO_TOE_BYPASS	OE.NO_TOE_BYPASS	The operational environment objective OE.NO_TOE_BYPASS is realized through A.NO_TOE_BYPASS.
A.TRUSTED_ADMIN	OE.TRUSTED_ADMIN	The Operational Environment objective OE.TRUSTED_ADMIN is realized through A.TRUSTED_ADMIN.

Table 51 WLAN Client Module Security Objectives Consistency Rationale to the GP OS PP

Objective	Rationale
O.AUTH_COMM	This objective is specifically for a communications interface that is defined by the PP-Module, but it is consistent with the general O.PROTECTED_COMMS objective specified in the Base-PP.
O.CRYPTOGRAPHIC_FUNCTIONS	The TOE implements this objective in part by relying on the cryptographic functionality specified in the Base-PP to address the Base-PP's O.PROTECTED_COMMS objective. The PP-Module uses these cryptographic functions for the same purpose as the Base-PP.
O.SELF_TEST	The Base-PP defines a general O.INTEGRITY objective; this PP-Module defines O.SELF_TEST as a specific method of guaranteeing the integrity of the TOE.

O.SYSTEM_MONITORING	The Base-PP defines an O.ACCOUNTABILITY objective for system auditing. The O.SYSTEM_MONITORING objective in this PP-Module serves the same purpose.
O.TOE_ADMINISTRATION	The Base-PP defines an O.MANAGEMENT objective for TOE administration. The O.TOE_ADMINISTRATION objective in this PP-Module serves the same purpose.
O_WIRELESS_ACCESS_POINT_CONNECTION	This objective relates to behavior that applies to a communications interface defined in this PP-Module and therefore does not relate to the Base-PP's functionality.
OE.NO_TOE_BYPASS	This objective relates to the deployment of the TOE in relation to the network resources that it interacts with. It does not enforce any restrictions on the TOE's deployment that are contrary to what the Base-PP requires.
OE.TRUSTED_ADMIN	The Base-PP defines OE.PROPER_USER and OE.PROPER_ADMIN objectives that serve the same purpose as OE.TRUSTED_ADMIN in this PP-Module.

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 52 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 **Error! Reference source not found. Error! Reference source not found.**.

Table 52 Rationale for Operations

PP or EP	PP or EP Requirement	ST Requirement	Operation & Rationale
GP OS	FAU_GEN.1	FAU_GEN.1	A selection and multiple assignments which are allowed by the PP.
GP OS, IPsec	FCS_CKM.1(1)	FCS_CKM.1	Multiple selections which are allowed by the PP and EP.
GP OS, IPsec	FCS_CKM.2(1)	FCS_CKM.2	A selection which is allowed by the PP and EP.
GP OS	FCS_CKM_EXT.4	FCS_CKM_EXT.4	Multiple selections which are allowed by the Technical Decision #239.
GP OS, IPsec	FCS_COP.1/ENCRYPT	FCS_COP.1/ENCRYPT	Multiple selections which are allowed by the PP and EP.
GP OS	FCS_COP.1/HASH	FCS_COP.1/HASH	Multiple selections which are allowed by the PP.
GP OS	FCS_COP.1/SIGN	FCS_COP.1/SIGN	A selection which is allowed by the PP.
GP OS	FCS_COP.1/KEYHMAC	FCS_COP.1/KEYHMAC	An assignment and multiple selections which are allowed by the PP.
GP OS	FCS_RBG_EXT.1	FCS_RBG_EXT.1	Multiple selections which are allowed by the PP.
GP OS	FCS_STO_EXT.1	FCS_STO_EXT.1	Copied from the PP without changes.
GP OS	FDP_ACF_EXT.1	FDP_ACF_EXT.1	Copied from the PP without changes.
GP OS	FDP_IFC_EXT.1	FDP_IFC_EXT.1	A selection which is allowed by the PP.
GP OS	FIA_AFL.1	FIA_AFLT.1	Multiple assignment and multiple selections which are allowed by the PP.

PP or EP	PP or EP Requirement	ST Requirement	Operation & Rationale
GP OS	FIA_UAU.5	FIA_UAU.5	An assignment and a selection which are allowed by the PP.
GP OS	FIA_X509_EXT.1	FIA_X509_EXT.1	A selection which is allowed by the PP.
GP OS	FIA_X509_EXT.2	FIA_X509_EXT.2	A selection which is allowed by the PP.
GP OS	FMT_MOF_EXT.1	FMT_MOF_EXT.1	Copied from the Technical Decision #0104 without changes.
GP OS	FMT_SMF_EXT.1	FMT_SMF_EXT.1	Refinements, selections and assignments which are allowed by the Technical Decision #104.
GP OS	FPT_ACF_EXT.1	FPT_ACF_EXT.1	Two assignment which is allowed by the PP.
GP OS	FPT_ASLR_EXT.1	FPT_ASLR_EXT.1	An assignment which is allowed by the PP.
GP OS	FPT_BLT_EXT.1	FPT_BLT_EXT.1	An assignment which is allowed by the PP.
GP OS	FPT_SBOP_EXT.1	FPT_SBOP_EXT.1	Copied from the PP without changes.
GP OS	FPT_SRP_EXT.1	FPT_SRP_EXT.1	A selection which is allowed by the PP.
GP OS	FPT_TST_EXT.1	FPT_TST_EXT.1	An assignment and multiple selections which are allowed by the PP.
GP OS	FPT_TUD_EXT.1	FPT_TUD_EXT.1	Added a refinement to align on SFR labels.
GP OS	FPT_TUD_EXT.2	FPT_TUD_EXT.2	Added a refinement to align on SFR labels.
GP OS	FTA_TAB.1	FTA_TAB.1	Copied from the PP without changes.
GP OS	FTP_TRP.1	FTP_TRP.1	Multiple selections which are allowed by the PP.
GP OS	FTP_ITC_EXT.1	FTP_ITC_EXT.1	An assignment and a selection which are allowed by the PP.
WLAN	FAU_GEN.1/WLAN	FAU_GEN.1(WLAN)	Two selections which are allowed by the WLAN Client module.
WLAN	FCS_CKM.1/WPA	FCS_CKM.1(WPA)	Two selections which are allowed by the WLAN Client module.
WLAN	FCS_CKM.2/WLAN	FCS_CKM.2(WLAN)	Copied from the WLAN Client module without changes.
WLAN	FCS_TLSC_EXT.1/WLAN	FCS_TLSC_EXT.1(WLAN)	Two selections which are allowed by the WLAN Client module.
WLAN	FCS_TLSC_EXT.2/WLAN	FCS_TLSC_EXT.2(WLAN)	A selection which is allowed by the WLAN Client module.
WLAN	FCS_WPA_EXT.1	FCS_WPA_EXT.1	A selection which is allowed by the WLAN Client module.

PP or EP	PP or EP Requirement	ST Requirement	Operation & Rationale
WLAN	FIA_PAE_EXT.1	FIA_PAE_EXT.1	Copied from the WLAN Client module without changes.
WLAN	FIA_X509_EXT.1/WLAN	FIA_X509_EXT.1(WLAN)	Copied from the WLAN Client module without changes.
WLAN	FIA_X509_EXT.2/WLAN	FIA_X509_EXT.2(WLAN)	A selection which is allowed by the WLAN Client module.
WLAN	FIA_X509_EXT.6	FIA_X509_EXT.6	Two selections which are allowed by the WLAN Client module.
WLAN	FMT_SMF.1/WLAN	FMT_SMF.1(WLAN)	Three selections which are allowed by the WLAN Client module.
WLAN	FPT_TST_EXT.3/WLAN	FPT_TST_EXT.3(WLAN)	Two selections which are allowed by the WLAN Client module.
WLAN	FTA_WSE_EXT.1	FTA_WSE_EXT.1	Copied from the WLAN Client module without changes.
WLAN	FTP_ITC.1/WLAN	FTP_ITC.1(WLAN)	Copied from the WLAN Client module without changes.
IPsec	FAU_GEN.1	FAU_GEN.1 (VPN)	Two selections and a refinement which are allowed by the VPN Client Module.
IPsec	FAU_SEL.1	FAU_SEL.1	A selection and an assignment which are allowed by the VPN Client Module.
IPsec	FCS_CKM.1/VPN	FCS_CKM.1(VPN)	Three selections which are allowed by the VPN Client Module.
IPsec	FCS_CKM_EXT.2	FCS_CKM_EXT.2	A selection which is allowed by the VPN Client Module.
IPsec	FCS_EAP_EXT.1	FCS_EAP_EXT.1	Multiple selections and assignments which are allowed by the VPN Client Module.
IPsec	FCS_IPSEC_EXT.1	FCS_IPSEC_EXT.1	Multiple selections and assignments which are allowed by the VPN Client Module.
IPsec	FDP_IFC_EXT.1	FDP_VPN_EXT.1	Copied from the VPN Client Module without changes.
IPsec	FDP_RDP.2	FDP_RDP.2	Two selections which are allowed by the VPN Client Module.
IPsec	FIA_PSK_EXT.1	FIA_PSK_EXT.1	Two selections which are allowed by the VPN Client Module.
IPsec	FIA_PSK_EXT.2	FIA_PSK_EXT.2	One selection which is allowed by the VPN Client Module.
IPsec	FIA_X509_EXT.3	FIA_X509_EXT.3	Multiple selections which are allowed by the VPN Client Module.
IPsec	FMT_SMF.1/VPN	FMT_SMF.1(VPN)	Two selections which are allowed by the VPN Client Module.
IPsec	FTP_TST_EXT.1/VPN	FTP_TST_EXT.1(VPN)	Three selections which are allowed by the VPN Client Module.

PP or EP	PP or EP Requirement	ST Requirement	Operation & Rationale
IPsec	FTP_ITC.1	FTP_ITC.1(VPN)	Multiple selections which are allowed by the VPN Client Module.
Bluetooth	FAU_GEN.1/BT	FAU_GEN.1(BT)	A selection and assignment which are allowed by the Bluetooth Module.
Bluetooth	FCS_CKM_EXT.8	FCS_CKM_EXT.8	An assignment which is allowed by the Bluetooth Module.
Bluetooth	FIA_BLT_EXT.1	FIA_BLT_EXT.1	Copied from the Bluetooth Module without changes.
Bluetooth	FIA_BLT_EXT.2	FIA_BLT_EXT.2	Copied from the Bluetooth Module without changes.
Bluetooth	FIA_BLT_EXT.3	FIA_BLT_EXT.3	Copied from the Bluetooth Module without changes.
Bluetooth	FIA_BLT_EXT.4	FIA_BLT_EXT.4	Copied from the Bluetooth Module without changes.
Bluetooth	FIA_BLT_EXT.6	FIA_BLT_EXT.6	An assignment which is allowed by the Bluetooth Module.
Bluetooth	FIA_BLT_EXT.7	FIA_BLT_EXT.7	An assignment which is allowed by the Bluetooth Module.
Bluetooth	FMT_MOF_EXT.1/BT	FMT_MOF_EXT.1(BT)	Copied from the Bluetooth Module without changes.
Bluetooth	FMT_SMF_EXT.1/BT	FMT_SMF_EXT.1(BT)	A selection and assignment which are allowed by the Bluetooth Module.
Bluetooth	FTP_BLT_EXT.1	FTP_BLT_EXT.1	Copied from the Bluetooth Module without changes.
Bluetooth	FTP_BLT_EXT.2	FTP_BLT_EXT.2	A selection which is allowed by the Bluetooth Module.
Bluetooth	FTP_BLT_EXT.3/BR	FTP_BLT_EXT.3(BR)	An assignment which is allowed by the Bluetooth Module.
Bluetooth	FTP_BLT_EXT.3/LE	FTP_BLT_EXT.3(LE)	An assignment which is allowed by the Bluetooth Module.
TLS	FCS_TLS_EXT.1	FCS_TLS_EXT.1	A selection which is allowed by the TLS Module.
TLS	FCS_TLSC_EXT.1	FCS_TLSC_EXT.1	Multiple selections and assignments which are allowed by the TLS Module.
TLS	FCS_TLSC_EXT.2	FCS_TLSC_EXT.2	Two selections which are allowed by the TLS Module.
TLS	FCS_TLSC_EXT.3	FCS_TLSC_EXT.3	A selection which is allowed by the TLS Module.
TLS	FCS_TLSC_EXT.4	FCS_TLSC_EXT.4	Two selections which are allowed by the TLS Module.
TLS	FCS_TLSC_EXT.5	FCS_TLSC_EXT.5	A selection which is allowed by the TLS Module.

PP or EP	PP or EP Requirement	ST Requirement	Operation & Rationale
TLS	FCS_TLSC_EXT.6	FCS_TLSC_EXT.6	Copied from the TLS Module without changes.
TLS	FCS_TLSS_EXT.1	FCS_TLSS_EXT.1	A selection which is allowed by the TLS Module.
TLS	FCS_TLSS_EXT.2	FCS_TLSS_EXT.2	Multiple selections and assignments which are allowed by the TLS Module.
TLS	FCS_TLSS_EXT.3	FCS_TLSS_EXT.3	Two selections which are allowed by the TLS Module.
TLS	FCS_TLSS_EXT.5	FCS_TLSS_EXT.5	Two selections which are allowed by the TLS Module.
TLS	FCS_TLSS_EXT.6	FCS_TLSS_EXT.6	A selection which is allowed by the TLS Module.
TLS	FCS_DTLSC_EXT.1	FCS_DTLSC_EXT.1	A selection which is allowed by the TLS Module.
TLS	FCS_DTLSC_EXT.2	FCS_DTLSC_EXT.2	Multiple selections and assignments which are allowed by the TLS Module.
TLS	FCS_DTLSC_EXT.3	FCS_DTLSC_EXT.3	Two selections which are allowed by the TLS Module.
TLS	FCS_DTLSC_EXT.4	FCS_DTLSC_EXT.4	A selection which is allowed by the TLS Module.
TLS	FCS_DTLSC_EXT.5	FCS_DTLSC_EXT.5	Two selections which are allowed by the TLS Module.
TLS	CS_DTLSS_EXT.1	CS_DTLSS_EXT.1	A selection which is allowed by the TLS Module.
TLS	FCS_DTLSS_EXT.2	FCS_DTLSS_EXT.2	Multiple selections and assignments which are allowed by the TLS Module.
TLS	FCS_DTLSS_EXT.3	FCS_DTLSS_EXT.3	Two selections which are allowed by the TLS Module.
TLS	FCS_DTLSS_EXT.5	FCS_DTLSS_EXT.5	Two selections which are allowed by the TLS Module.

8.2 Security Assurance Requirements

The statement of security assurance requirements (SARs) found in section 5.2.1 is in strict conformance with the Protection Profile for General Purpose Operating Systems and the Assurance Package for Flaw Remediation.

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 53**. 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 53 Requirement to Security Function Correspondence

PP or EP	Requirement	Audit	Cryptographic Protection	User Data Protection	I & A	Security Management	TSF Protection	Resource Utilization	TOE Access	Trusted Path / Channel
GP OS	FAU_GEN.1	X								
GP OS	FCS_CKM.1		X							
GP OS	FCS_CKM.2		X							
GP OS	FCS_CKM_EXT.4		X							
GP OS	FCS_COP.1/ENCRYPT		X							
GP OS	FCS_COP.1/HASH		X							
GP OS	FCS_COP.1/SIGN		X							
GP OS	FCS_COP./KEY(MAC		X							
GP OS	FCS_RBG_EXT.1		X							
GP OS	FCS_STO_EXT.1		X							
GP OS	FDP_ACF_EXT.1			X						
GP OS	FDP_IFC_EXT.1			X						
GP OS	FIA_AFL.1				X					
GP OS	FIA_UAU.5				X					
GP OS	FIA_X509_EXT.1				X					
GP OS	FIA_X509_EXT.2				X					
GP OS	FIA_X509_EXT.4				X					
GP OS	FMT_MOF_EXT.1					X				
GP OS	FMT_SMF_EXT.1					X				
GP OS	FPT_ACF_EXT.1						X			
GP OS	FPT_ASLR_EXT.1						X			
GP OS	FPT_BLT_EXT.1						X			
GP OS	FPT_SBOP_EXT.1						X			
GP OS	FPT_SRP_EXT.1						X			

PP or EP	Requirement	Audit	Cryptographic Protection	User Data Protection	I & A	Security Management	TSF Protection	Resource Utilization	TOE Access	Trusted Path / Channel
GP OS	FPT_TST_EXT.1						X			
GP OS	FPT_TUD_EXT.1						X			
GP OS	FPT_TUD_EXT.2						X			
GP OS	FTA_TAB.1								X	
GP OS	FTP_TRP.1									X
GP OS	FTP_ITC_EXT.1									X
WLAN	FAU_GEN.1(WLAN)	X								
WLAN	FCS_CKM.1(WPA)		X							
WLAN	FCS_CKM.2(WLAN)		X							
WLAN	FCS_TLSC_EXT.1(WLAN)		X							
WLAN	FCS_TLSC_EXT.2(WLAN)		X							
WLAN	FCS_WPA_EXT.1		X							
WLAN	FIA_PAE_EXT.1				X					
WLAN	FIA_X509_EXT.1(WLAN)				X					
WLAN	FIA_X509_EXT.2(WLAN)				X					
WLAN	FIA_X509_EXT.6				X					
WLAN	FMT_SMF.1(WLAN)					X				
WLAN	FPT_TST_EXT.3(WLAN)						X			
WLAN	FTA_WSE_EXT.1								X	
WLAN	FTP_ITC.1(WLAN)									X
IPsec	FAU_SEL.1	X								
IPsec	FCS_CKM.1(VPN)		X							
IPsec	FCS_CKM_EXT.2		X							
IPsec	FCS_EAP_EXT.1		X							
IPsec	FCS_IPSEC_EXT.1		X							
IPsec	FDP_VPN_EXT.1			X						
IPsec	FDP_RIP.2			X						
IPsec	FCS_PSK_EXT.1				X					
IPsec	FCS_PSK_EXT.1				X					
IPsec	FIA_X509_EXT.3				X					
IPsec	FMT_SMF.1(VPN)					X				
IPsec	FPT_TST_EXT.1(VPN)						X			
IPsec	FTP_ITC.1(VPN)									X
Bluetooth	FAU_GEN.1(BT)	X								
Bluetooth	FCS_CKM_EXT.8		X							
Bluetooth	FIA_BLT_EXT.1				X					

PP or EP	Requirement	Audit	Cryptographic Protection	User Data Protection	I & A	Security Management	TSF Protection	Resource Utilization	TOE Access	Trusted Path / Channel
Bluetooth	FIA_BLT_EXT.2				X					
Bluetooth	FIA_BLT_EXT.3				X					
Bluetooth	FIA_BLT_EXT.4				X					
Bluetooth	FIA_BLT_EXT.6				X					
Bluetooth	FIA_BLT_EXT.7				X					
Bluetooth	FMT_MOF_EXT.1(BT)					X				
Bluetooth	FMT_SMF_EXT.1(BT)					X				
Bluetooth	FTP_BLT_EXT.1									X
Bluetooth	FTP_BLT_EXT.2									X
Bluetooth	FTP_BLT_EXT.3(BR)									X
Bluetooth	FTP_BLT_EXT.1(LE)									X
TLS	FCS_TLS_EXT.1		X							
TLS	FCS_TLSC_EXT.1		X							
TLS	FCS_TLSC_EXT.2		X							
TLS	FCS_TLSC_EXT.3		X							
TLS	FCS_TLSC_EXT.4		X							
TLS	FCS_TLSC_EXT.5		X							
TLS	FCS_TLSC_EXT.6		X							
TLS	FCS_TLSS_EXT.1		X							
TLS	FCS_TLSS_EXT.2		X							
TLS	FCS_TLSS_EXT.3		X							
TLS	FCS_TLSS_EXT.5		X							
TLS	FCS_TLSS_EXT.6		X							
TLS	FCS_DTLSC_EXT.1		X							
TLS	FCS_DTLSC_EXT.2		X							
TLS	FCS_DTLSC_EXT.3		X							
TLS	FCS_DTLSC_EXT.4		X							
TLS	FCS5DTLSC_EXT.1		X							
TLS	FCS_DTLSC_EXT.4		X							
TLS	FCS_DTLSS_EXT.2		X							
TLS	FCS_DTLSS_EXT.3		X							
TLS	FCS_DTLSS_EXT.5		X							

9 Appendix A: List of Abbreviations

Table 54 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