Supporting Document
Mandatory Technical Document

Collaborative Protection Profile for Application Software
Version: 2.0
2025-06-16
Application Software International Technical Community (AppSW-iTC)

Foreword

This is a Supporting Document (SD), intended to complement the Common Criteria version 3 and the associated Common Evaluation Methodology for Information Technology Security Evaluation.

SDs may be “Guidance Documents”, that highlight specific approaches and application of the standard to areas where no mutual recognition of its application is required, and as such, are not of normative nature, or “Mandatory Technical Documents”, whose application is mandatory for evaluations whose scope is covered by that of the SD. The usage of the latter class is not only mandatory, but certificates issued as a result of their application are recognized under the CCRA.

Technical Editor:
National Information Assurance Partnership (NIAP)

Document history:

Version	Date	Comment
v 1.0	2014-10-20	Initial release
v 1.1	2014-11-05	Addition to TLS cipher suite selections
v 1.2	2016-04-22	Added server-side TLS requirements (selection-based) Multiple clarification based on NIAP TRRT inquiries Refactored FDP_DEC_EXT.1 into separate components
v 1.3	2019-03-01	Incorporated available Technical Decisions Refactored FPT_TUD Added a selection to FTP_DIT Moved SWID Tags requirement Leveraged TLS Package Added equivalency section

v 1.4	2021-10-07	Incorporated applicable Technical Decisions Updated to TLS FP 2.1 Incorporated SSH FP 2.0
v 2.0	2025-06-16	Rebaselined from the NIAP Protection Profile for Application Software, Version 2.0, as the source baseline rather than revising the prior collaborative Protection Profile for Application Software Version 1.4 Converted to collaborative Protection Profile maintained by the Application Software International Technical Community (AppSW-iTC) CC:2022 conversion Updating for TLS FP, SSH FP, and X.509 FP TDs and GitHub Issues CNSA 2.0 updates ALC FLR Updates

General Purpose:
The purpose of this SD is to define evaluation methods for the functional behavior of products.

Acknowledgments:
This SD was developed with support from NIAP Technical Community members, with representatives from industry, government agencies, Common Criteria Test Laboratories, and members of academia.

Table of Contents

Appendix A - References

1 Introduction

1.1 Technology Area and Scope of Supporting Document

The scope of the Collaborative Protection Profile for Application Software is to describe the security functionality of products in terms of [CC] and to define functional and assurance requirements for them.

Although Evaluation Activities are defined mainly for the evaluators to follow, in general they also help developers to prepare for evaluation by identifying specific requirements for their TOE. The specific requirements in Evaluation Activities may in some cases clarify the meaning of Security Functional Requirements (SFR), and may identify particular requirements for the content of Security Targets (ST) (especially the TOE Summary Specification), user guidance documentation, and possibly supplementary information (e.g. for entropy analysis or cryptographic key management architecture).

1.2 Structure of the Document

Evaluation Activities can be defined for both SFRs and Security Assurance Requirements (SAR), which are themselves defined in separate sections of the SD.

If any Evaluation Activity cannot be successfully completed in an evaluation, then the overall verdict for the evaluation is a 'fail'. In rare cases there may be acceptable reasons why an Evaluation Activity may be modified or deemed not applicable for a particular TOE, but this must be approved by the Certification Body for the evaluation.

In general, if all Evaluation Activities (for both SFRs and SARs) are successfully completed in an evaluation then it would be expected that the overall verdict for the evaluation is a ‘pass’. To reach a ‘fail’ verdict when the Evaluation Activities have been successfully completed would require a specific justification from the evaluator as to why the Evaluation Activities were not sufficient for that TOE.

Similarly, at the more granular level of assurance components, if the Evaluation Activities for an assurance component and all of its related SFR Evaluation Activities are successfully completed in an evaluation then it would be expected that the verdict for the assurance component is a ‘pass’. To reach a ‘fail’ verdict for the assurance component when these Evaluation Activities have been successfully completed would require a specific justification from the evaluator as to why the Evaluation Activities were not sufficient for that TOE.

1.3 Terms

The following sections list Common Criteria and technology terms used in this document.

1.3.1 Common Criteria Terms

Assurance	Grounds for confidence that a TOE meets the SFRs [CC].
Base Protection Profile (Base-PP)	Protection Profile used as a basis to build a PP-Configuration.
Collaborative Protection Profile (cPP)	A Protection Profile developed by international technical communities and approved by multiple schemes.

Common Criteria (CC)	Common Criteria for Information Technology Security Evaluation (International Standard ISO/IEC 15408).
Common Criteria Testing Laboratory	Within the context of the Common Criteria Evaluation and Validation Scheme (CCEVS), an IT security evaluation facility accredited by the National Voluntary Laboratory Accreditation Program (NVLAP) and approved by the NIAP Validation Body to conduct Common Criteria-based evaluations.
Common Evaluation Methodology (CEM)	Common Evaluation Methodology for Information Technology Security Evaluation.
Direct Rationale	A type of Protection Profile, PP-Module, or Security Target in which the security problem definition (SPD) elements are mapped directly to the SFRs and possibly to the security objectives for the operational environment. There are no security objectives for the TOE.
Distributed TOE	A TOE composed of multiple components operating as a logical whole.
Extended Package (EP)	A deprecated document form for collecting SFRs that implement a particular protocol, technology, or functionality. See Functional Packages.
Functional Package (FP)	A document that collects SFRs for a particular protocol, technology, or functionality.
Operational Environment (OE)	Hardware and software that are outside the TOE boundary that support the TOE functionality and security policy.
Protection Profile (PP)	An implementation-independent set of security requirements for a category of products.
Protection Profile Configuration (PP-Configuration)	A comprehensive set of security requirements for a product type that consists of at least one Base-PP and at least one PP-Module.
Protection Profile Module (PP-Module)	An implementation-independent statement of security needs for a TOE type complementary to one or more Base-PPs.
Security Assurance Requirement (SAR)	A requirement to assure the security of the TOE.
Security Functional Requirement (SFR)	A requirement for security enforcement by the TOE.
Security Target (ST)	A set of implementation-dependent security requirements for a specific product.
Target of Evaluation (TOE)	The product under evaluation.
TOE Security Functionality (TSF)	The security functionality of the product under evaluation.
TOE Summary Specification (TSS)	A description of how a TOE satisfies the SFRs in an ST.

1.3.2 Technical Terms

Address Space Layout Randomization (ASLR)	An anti-exploitation feature which loads memory mappings into unpredictable locations. ASLR makes it more difficult for an attacker to redirect control to code that they have introduced into the address space of an application process.
Application (app)	Software that runs on a platform and performs tasks on behalf of the user or owner of the platform, as well as its supporting documentation. The terms *TOE* and *application* are interchangeable in this document.
Application Programming Interface (API)	A specification of routines, data structures, object classes, and variables that allows an application to make use of services provided by another software component, such as a library. APIs are often provided for a set of libraries included with the platform.
Credential	Data that establishes the identity of a user, e.g. a cryptographic key or password.
Data Execution Prevention (DEP)	An anti-exploitation feature of modern operating systems executing on modern computer hardware, which enforces a non-execute permission on pages of memory. DEP prevents pages of memory from containing both data and instructions, which makes it more difficult for an attacker to introduce and execute code.
Developer	An entity that writes application software. For the purposes of this document, vendors and developers are the same.
Mobile Code	Software transmitted from a remote system for execution within a limited execution environment on the local system. Typically, there is no persistent installation and execution begins without the user's consent or even notification. Examples of mobile code technologies include JavaScript, Java applets, Adobe Flash, and Microsoft Silverlight.
Operating System (OS)	Software that manages hardware resources and provides services for applications.
Personally Identifiable Information (PII)	Any information about an individual maintained by an agency, including, but not limited to, education, financial transactions, medical history, and criminal or employment history and information which can be used to distinguish or trace an individual's identity, such as their name, social security number, date and place of birth, mother’s maiden name, biometric records, etc., including any other personal information which is linked or linkable to an individual.[OMB]
Platform	The environment in which application software runs. The platform can be an operating system, hardware environment, a software based execution environment, or some combination of these. These types of platforms may also run atop other platforms.
Sensitive Data	Sensitive data may include all user or enterprise data or may be specific application data such as emails, messaging, documents, calendar items, and contacts. Sensitive data must minimally include PII, credentials, and keys. Sensitive data shall be identified in the application’s TSS by the ST author.
Stack Cookie	An anti-exploitation feature that places a value on the stack at the start of a function call, and checks that the value is the same at the end of the function call. This is also referred to as Stack Guard, or Stack Canaries.
Vendor	An entity that sells application software. For purposes of this document, vendors and developers are the same. Vendors are responsible for maintaining and updating application software.

2 Evaluation Activities for SFRs

The EAs presented in this section capture the actions the evaluator performs to address technology specific aspects covering specific SARs (e.g. ASE_TSS.1, ADV_FSP.1, AGD_OPE.1, and ATE_IND.1) – this is in addition to the CEM workunits that are performed in Section 3 Evaluation Activities for SARs.

Regarding design descriptions (designated by the subsections labeled TSS, as well as any required supplementary material that may be treated as proprietary), the evaluator must ensure there is specific information that satisfies the EA. For findings regarding the TSS section, the evaluator's verdicts will be associated with the CEM workunit ASE_TSS.1-1. Evaluator verdicts associated with the supplementary evidence will also be associated with ASE_TSS.1-1, since the requirement to provide such evidence is specified in ASE in the PP.

For ensuring the guidance documentation provides sufficient information for the administrators/users as it pertains to SFRs, the evaluator's verdicts will be associated with CEM workunits ADV_FSP.1-7, AGD_OPE.1-4, and AGD_OPE.1-5.

Finally, the subsection labeled Tests is where the authors have determined that testing of the product in the context of the associated SFR is necessary. While the evaluator is expected to develop tests, there may be instances where it is more practical for the developer to construct tests, or where the developer may have existing tests. Therefore, it is acceptable for the evaluator to witness developer-generated tests in lieu of executing the tests. In this case, the evaluator must ensure the developer's tests are executing both in the manner declared by the developer and as mandated by the EA. The CEM workunits that are associated with the EAs specified in this section are: ATE_IND.1-3, ATE_IND.1-4, ATE_IND.1-5, ATE_IND.1-6, and ATE_IND.1-7.

2.1 TOE SFR Evaluation Activities

2.1.1 Cryptographic Support (FCS)

FCS_CKM_EXT.1 Cryptographic Key Generation Services

FCS_CKM_EXT.1.1
TSS

The evaluator shall examine the TSS to verify that it describes whether the TSF has functions that require the use of asymmetric key generation services, and whether these services are implemented within the TOE boundary or invoked by the TSF from its operational environment.

Conditional: If the ST claims "**generate no asymmetric keys**," the evaluator shall ensure that the TOE does not have any functions that would require asymmetric key generation (for example, because it does not use asymmetric keys for any purpose or because the keys that it does use are generated elsewhere and imported into it as part of initial setup).

Guidance

None.

Tests

None.

FCS_RBG_EXT.1 Random Bit Generation Services

FCS_RBG_EXT.1
TSS

If "**use no DRBG functionality**" is selected, the evaluator shall inspect the application and its developer documentation and verify that the application needs no random bit generation services.

If "**implement DRBG functionality**" is selected, the evaluator shall ensure that FCS_RBG.1 is claimed.

If "**invoke platform-provided DRBG functionality**" is selected, the evaluator performs the following activities. The evaluator shall examine the TSS to confirm that it identifies all functions (as described by the SFRs included in the ST) that obtain random numbers from the platform RBG. The evaluator shall determine that for each of these functions, the TSS states which platform interface (API) is used to obtain the random numbers. The evaluator shall confirm that each of these interfaces corresponds to the acceptable interfaces listed for each platform below.

It should be noted that there is no expectation that the evaluators attempt to confirm that the APIs are being used correctly for the functions identified in the TSS; the activity is to list the used APIs and then do an existence check via decompilation.

Guidance

The evaluator shall verify the guidance documentation contains any information required for configuring the DRBG.

Tests

If "**invoke platform-provided DRBG functionality**" is selected, the following tests shall be performed:

The evaluator shall decompile the application binary using a decompiler suitable for the application (TOE). The evaluator shall search the output of the decompiler to determine that, for each API listed in the TSS, that API appears in the output. If the representation of the API does not correspond directly to the strings in the following list, the evaluator shall provide a mapping from the decompiled text to its corresponding API, with a description of why the API text does not directly correspond to the decompiled text and justification that the decompiled text corresponds to the associated API.

The following are the per-platform list of acceptable APIs:

Platforms: **Android**: *Mobile operating systems based on Google Android*
The evaluator shall verify that the application uses at least one of `javax.crypto.KeyGenerator` *class or the* `java.security.SecureRandom` *class or* `/dev/random` *or* `/dev/urandom`*.*

Platforms: **Microsoft Windows**: *Microsoft Windows operating systems*
The evaluator shall verify that `rand_s`*,* `RtlGenRandom`*,* `BCryptGenRandom`*, or* `CryptGenRandom` *API is used for classic desktop applications. The evaluator shall verify the application uses the* `RNGCryptoServiceProvider` *class or derives a class from* `System.Security.Cryptography.RandomNumberGenerator API` *for Windows Universal Applications. It is only required that the API is called/invoked, there is no requirement that the API be used directly. In future versions of this document,* `CryptGenRandom` *may be removed as an option as it is no longer the preferred API per vendor documentation.*

Platforms: **Apple iOS and iPadOS**: *Apple's mobile operating system for iPhones and iPads*
The evaluator shall verify that the application invokes either `SecRandomCopyBytes`*,* `CCRandomGenerateBytes`*, or* `CCRandomCopyBytes`*, or uses* `/dev/random` *directly to acquire random.*

Platforms: **Linux**: *Linux-based operating systems other than Android*
The evaluator shall verify that the application collects random from `/dev/random` or `/dev/urandom`.

Platforms: **Oracle Solaris**: *Oracle's enterprise operating system*
The evaluator shall verify that the application collects random from `/dev/random`.

Platforms: **Apple macOS**: *Apple's operating system for Mac devices*
The evaluator shall verify that the application invokes either `CCRandomGenerateBytes` or `CCRandomCopyBytes`, or collects random from `/dev/random`.

If invocation of platform-provided functionality is achieved in another way, the evaluator shall ensure the TSS describes how this is carried out, and how it is equivalent to the methods listed here (e.g. higher-level API invokes identical low-level API).

FCS_STO_EXT.1 Storage of Credentials

FCS_STO_EXT.1
TSS

The evaluator shall check the TSS to ensure that it lists all persistent credentials (secret keys, PKI private keys, or passwords) needed to meet the requirements in the ST. For each of these items, the evaluator shall confirm that the TSS lists for what purpose it is used, and how it is stored.

If **not store any credentials** is selected, the evaluator shall verify the TSS describes the behavior of the TOE in sufficient detail to verify that the TSF does not have any behavior that would require any credentials to be stored (e.g., because the TOE does not have any functionality requiring authentication).

If **securely store** is selected, the evaluator shall verify the TSS contains the platform functions utilized and verify those functions are documented by the platform to be non-deprecated functions meeting the specifications in the requirement.

If **invoke the functionality provided by the platform to securely store** is selected, the evaluator shall confirm the TSS describes how the platform storage is invoked for each supported platform. The evaluator shall confirm the invocation of the platform is using non-deprecated functions provided by the platform(s).

Guidance

None.

Tests
For all credentials for which the application implements functionality, the evaluator shall verify credentials are encrypted according to FCS_COP.1/SKC or conditioned according to FCS_PBKDF_EXT.1. For all credentials for which the application invokes platform-provided functionality, the evaluator shall perform the following actions which vary per platform.

Platforms: **Android**: *Mobile operating systems based on Google Android*
The evaluator shall verify that the application uses the Android `KeyStore` or the Android `KeyChain` to store certificates.

Platforms: **Microsoft Windows**: *Microsoft Windows operating systems*
The evaluator shall verify that all certificates are stored in the Windows Certificate Store. The evaluator shall verify that other credentials, like passwords, are stored in the Windows Credential Manager or stored using the Data Protection API (DPAPI). For Windows Universal Applications, the evaluator shall verify that the application is using the ProtectData class and storing credentials in IsolatedStorage.

Platforms: **Apple iOS and iPadOS**: *Apple's mobile operating system for iPhones and iPads*
The evaluator shall verify that all credentials are stored within a `Keychain`.

Platforms: **Linux**: *Linux-based operating systems other than Android*
The evaluator shall verify that all keys are stored using Linux keyrings.

Platforms: **Oracle Solaris**: *Oracle's enterprise operating system*
The evaluator shall verify that all keys are stored using Solaris `Key Management Framework (KMF)`.

Platforms: **Apple macOS**: *Apple's operating system for Mac devices*
The evaluator shall verify that all credentials are stored within `Keychain`.

2.1.2 User Data Protection (FDP)

FDP_DAR_EXT.1 Encryption Of Sensitive Application Data

FDP_DAR_EXT.1
TSS

If any selection other than **not store any sensitive data** is selected the evaluator shall examine the TSS to ensure that it describes the sensitive data processed by the application. The evaluator shall then ensure that the following activities cover all of the sensitive data identified in the TSS.

If **not store any sensitive data** is selected, the evaluator shall inspect the TSS to ensure that it describes how sensitive data cannot be written to non-volatile memory. The evaluator shall also ensure that this is consistent with the file system test below.

If **implement functionality to encrypt sensitive data** is selected the evaluator shall confirm the TSS describes how the application ensures all sensitive data is protected by the file encryption functions.

If **protect sensitive data in accordance with FCS_STO_EXT.1** is selected the evaluator shall confirm the TSS describes which data is protected via this mechanism and the selections within FCS_STO_EXT.1 that are leveraged. If multiple selections are included the evaluator shall ensure the TSS describes which sensitive data is captured by which selection.

If "**leverage platform-provided functionality...**" is selected, the evaluation activities will be performed as stated in the following requirements, which vary on a per-platform basis.

Platforms: **Android**: *Mobile operating systems based on Google Android*
The evaluator shall inspect the TSS and verify that it describes how files containing sensitive data are stored with the `MODE_PRIVATE` flag set.

Platforms: **Microsoft Windows**: *Microsoft Windows operating systems*
The Windows platform currently does not provide data-at-rest encryption services which depend upon invocation by application developers.

Platforms: **Apple iOS and iPadOS**: *Apple's mobile operating system for iPhones and iPads*
The evaluator shall inspect the TSS and ensure that it describes how the application uses the Complete Protection, Protected Unless Open, or Protected Until First User Authentication Data Protection Class for

each data file stored locally.

Platforms: **Linux**: *Linux-based operating systems other than Android*
The Linux platform currently does not provide data-at-rest encryption services which depend upon invocation by application developers.

Platforms: **Oracle Solaris**: *Oracle's enterprise operating system*
The Solaris platform currently does not provide data-at-rest encryption services which depend upon invocation by application developers.

Platforms: **Apple macOS**: *Apple's operating system for Mac devices*
The macOS platform currently does not provide data-at-rest encryption services which depend upon invocation by application developers.

Guidance

The evaluator shall confirm the operational guidance contains any instructions necessary for configuring the storage and protection of any sensitive data.

If **leverage platform-provided functionality to encrypt sensitive data** is selected the evaluator shall confirm the operational guidance contains the list of supported operational environments and any steps necessary to ensure the platform captures any sensitive data that is stored.

Tests

If "**implement functionality to encrypt sensitive data as defined in the PP-Module for File Encryption**" or "**protect sensitive data in accordance with FCS_STO_EXT.1**" is selected, the evaluator shall inventory the file system locations where the application may write data. The evaluator shall run the application and attempt to store sensitive data. The evaluator shall then inspect those areas of the file system to note where data was stored (if any), and verify it has been encrypted.

If "**leverage platform-provided functionality...**" is selected no additional testing is required.

FDP_DEC_EXT.1 Access to Platform Resources

FDP_DEC_EXT.1.1
TSS

None.

Guidance

The evaluator shall perform the platform-specific actions below and inspect user documentation to determine the application's access to hardware resources. The evaluator shall ensure that this is consistent with the selections indicated. The evaluator shall review documentation provided by the application developer and for each resource which it accesses, identify the justification as to why access is required.

Tests

Platforms: **Android**: *Mobile operating systems based on Google Android*
The evaluator shall verify that each `uses-permission` entry in the AndroidManifest.xml file for access to a hardware resource is reflected in the selection.

Platforms: **Microsoft Windows**: *Microsoft Windows operating systems*
For Windows Universal Applications the evaluator shall check the AppxManifest.xml file for a list of required hardware capabilities. The evaluator shall verify that the user is made aware of the required hardware capabilities when the application is first installed. This includes permissions such as

ID_CAP_ISV_CAMERA, ID_CAP_LOCATION, ID_CAP_NETWORKING, ID_CAP_MICROPHONE, ID_CAP_PROXIMITY and so on. A complete list of Windows App permissions can be found at:

- *http://msdn.microsoft.com/en-US/library/windows/apps/jj206936.aspx*

For Windows Desktop Applications the evaluator shall identify in either the application software or its documentation the list of the required hardware resources.

Platforms: **Apple iOS and iPadOS**: *Apple's mobile operating system for iPhones and iPads*

The evaluator shall verify that either the application or the documentation provides a list of the hardware resources it accesses.

Platforms: **Linux**: *Linux-based operating systems other than Android*

The evaluator shall verify that either the application software or its documentation provides a list of the hardware resources it accesses.

Platforms: **Oracle Solaris**: *Oracle's enterprise operating system*

The evaluator shall verify that either the application software or its documentation provides a list of the hardware resources it accesses.

Platforms: **Apple macOS**: *Apple's operating system for Mac devices*

The evaluator shall verify that either the application software or its documentation provides a list of the hardware resources it accesses.

FDP_DEC_EXT.1.2

TSS

None.

Guidance

The evaluator shall perform the platform-specific actions below and inspect user documentation to determine the application's access to sensitive information repositories. The evaluator shall ensure that this is consistent with the selections indicated. The evaluator shall review documentation provided by the application developer and for each sensitive information repository which it accesses, identify the justification as to why access is required.

Tests

Platforms: **Android**: *Mobile operating systems based on Google Android*

The evaluator shall verify that each `uses-permission` entry in the AndroidManifest.xml file for access to a sensitive information repository is reflected in the selection.

Platforms: **Microsoft Windows**: *Microsoft Windows operating systems*

For Windows Universal Applications the evaluator shall check the AppxManifest.xml file for a list of required capabilities. The evaluator shall identify the required information repositories when the application is first installed. This includes permissions such as ID_CAP_CONTACTS,ID_CAP_APPOINTMENTS,ID_CAP_MEDIALIB and so on. A complete list of Windows App permissions can be found at:

- *http://msdn.microsoft.com/en-US/library/windows/apps/jj206936.aspx*

For Windows Desktop Applications the evaluator shall identify in either the application software or its documentation the list of sensitive information repositories it accesses.

Platforms: **Apple iOS and iPadOS**: *Apple's mobile operating system for iPhones and iPads*

The evaluator shall verify that either the application software or its documentation provides a list of the sensitive information repositories it accesses.

Platforms: **Linux**: *Linux-based operating systems other than Android*

The evaluator shall verify that either the application software or its documentation provides a list of sensitive information repositories it accesses.

Platforms: **Oracle Solaris**: *Oracle's enterprise operating system*
The evaluator shall verify that either the application software or its documentation provides a list of sensitive information repositories it accesses.

Platforms: **Apple macOS**: *Apple's operating system for Mac devices*
The evaluator shall verify that either the application software or its documentation provides a list of sensitive information repositories it accesses.

FDP_NET_EXT.1 Network Communications

FDP_NET_EXT.1
TSS

None.

Guidance

The evaluator shall verify the guidance documents contain any instructions necessary to configure the restriction of network communications.

Tests
The evaluator shall perform the following tests:

- Test FDP_NET_EXT.1:1: The evaluator shall run the application. While the application is running, the evaluator shall sniff network traffic ignoring all non-application associated traffic and verify that any network communications witnessed are documented in the TSS or are user-initiated.
- Test FDP_NET_EXT.1:2: The evaluator shall run the application. After the application initializes, the evaluator shall run network port scans to verify that any ports opened by the application have been captured in the ST for the third selection and its assignment. This includes connection-based protocols (e.g. TCP, DCCP) as well as connectionless protocols (e.g. UDP).

Platforms: **Android**: *Mobile operating systems based on Google Android*
*If **"no network communication"** is selected, the evaluator shall ensure that the application's AndroidManifest.xml file does not contain a uses-permission or uses-permission-sdk-23 tag containing android:name="android.permission.INTERNET". In this case, it is not necessary to perform the above Tests 1 and 2, as the platform will not allow the application to perform any network communication.*

2.1.3 Security Management (FMT)

FMT_CFG_EXT.1 Secure by Default Configuration

FMT_CFG_EXT.1.1
TSS

The evaluator shall check that the TSS describes whether the application requires any type of application provided credentials and whether the application is pre-configured with default values for these credentials. If credentials are required, the evaluator shall verify that the TSS details how use of the TOE is restricted until new credentials are set (which includes the replacement of default credentials if any are present).

Guidance

The evaluator shall verify the guidance documentation details regarding any default or null application provided credentials being used and how they would be updated.

Tests

If the application uses any default credentials the evaluator shall run the following tests.

- Test FMT_CFG_EXT.1.1:1: For any application provided credentials the evaluator shall install and run the application without generating or loading new credentials and verify that only the minimal application functionality required to set new credentials is available.
- Test FMT_CFG_EXT.1.1:2: For any application provided credentials the evaluator shall attempt to clear all credentials and verify that only the minimal application functionality required to set new credentials is available.
- Test FMT_CFG_EXT.1.1:3: For any application provided credentials the evaluator shall run the application, establish new credentials and verify that the original default credentials no longer provide access to the application.

FMT_CFG_EXT.1.2

TSS

None.

Guidance

None.

Tests

The evaluator shall install and run the application. The evaluator shall inspect the file system of the platform (to the extent possible) for any files created by the application and ensure that their permissions are adequate to protect them. The method of doing so varies per platform.

> Platforms: **Android**: *Mobile operating systems based on Google Android*
>
> *The evaluator shall run the command `find -L. -perm /002` inside the application's data directories to ensure that all files are not world-writable. The command should not print any files (for this test, directories are not considered to be files).*

> Platforms: **Microsoft Windows**: *Microsoft Windows operating systems*
>
> *The evaluator shall run the SysInternals tools Process Monitor and Access Check (or tools of equivalent capability, like icacls.exe) for Classic Desktop applications to verify that files written to disk during an application's installation have the correct file permissions, such that a standard user cannot modify the application or its data files. For Windows Universal Applications the evaluator shall consider the requirement met because of the AppContainer sandbox.*

> Platforms: **Apple iOS and iPadOS**: *Apple's mobile operating system for iPhones and iPads*
>
> *The evaluator shall determine whether the application leverages the appropriate Data Protection Class for each data file stored locally.*

> Platforms: **Linux**: *Linux-based operating systems other than Android*
>
> *The evaluator shall run the command `find -L. -perm /002` inside the application's data directories to ensure that all files are not world-writable. The command should not print any files.*

> Platforms: **Oracle Solaris**: *Oracle's enterprise operating system*
>
> *The evaluator shall run the command `find. \( -perm -002 \)` inside the application's data directories to ensure that all files are not world-writable. The command should not print any files.*

> Platforms: **Apple macOS**: *Apple's operating system for Mac devices*

The evaluator shall run the command `find. -perm +002` inside the application's data directories to ensure that all files are not world-writable. The command should not print any files.

FMT_MEC_EXT.1 Supported Configuration Mechanism

FMT_MEC_EXT.1
TSS

The evaluator shall review the TSS to identify the application's configuration options (e.g., settings) and determine whether these are stored and set using the mechanisms supported by the platform or implemented by the application in accordance with the PP-Module for File Encryption. At a minimum the TSS shall list settings related to any SFRs and any settings that are mandated in the operational guidance in response to an SFR.

Conditional: If "**implement functionality to encrypt and store configuration options as defined by FDP_PRT_EXT.1 in the PP-Module for File Encryption**" is selected, the evaluator shall ensure that the TSS identifies those options, as well as indicates where the encrypted representation of these options is stored.

Guidance

The evaluator shall verify the guidance documentation contains any information necessary to configure the protection of configuration settings.

Tests
If " **invoke the mechanisms recommended by the platform vendor for storing and setting configuration options**" is selected, the method of testing varies per platform as follows:

Platforms: **Android**: *Mobile operating systems based on Google Android*

The evaluator shall inspect the TSS and verify that it describes what Android API is used (and provides a link to the documentation of the API) when storing configuration data. The evaluator shall run the application and verify that the behavior of the TOE is consistent with where and how the API documentation says the configuration data will be stored.

For SharedPreferences, the evaluator shall examine the XML file to make sure it reflects the changes made to the configuration to verify that the application used SharedPreferences or PreferenceActivity to store the configuration data. For DataStore, the evaluator shall use a protocol buffer analyzer to examine the file to make sure it reflects the changes made to the configuration to verify that the application used DataStore to store the configuration data.

Platforms: **Microsoft Windows**: *Microsoft Windows operating systems*

The evaluator shall determine and verify that Windows Universal Applications use either the Windows.Storage namespace, Windows.UI.ApplicationSettings namespace, or the IsolatedStorageSettings namespace for storing application specific settings. For .NET applications, the evaluator shall determine and verify that the application uses one of the locations listed in https://docs.microsoft.com/en-us/dotnet/framework/configure-apps/ or https://learn.microsoft.com/en-us/previous-versions/dotnet/netframework-4.0/zzdt0e7f(v=vs.100) or https://learn.microsoft.com/en-us/aspnet/core/fundamentals/configuration/ or https://learn.microsoft.com/en-us/aspnet/core/host-and-deploy/iis/web-config for storing application specific (whether application-wide or user-specific) settings.

For Classic Desktop applications, the evaluator shall run the application while monitoring it with the SysInternals tool Process Monitor and make changes to its configuration. The evaluator shall verify that Process Monitor logs show corresponding changes to the Windows Registry or C:\ProgramData

directory.

Platforms: **Apple iOS and iPadOS**: *Apple's mobile operating system for iPhones and iPads*
The evaluator shall verify that the app uses the `user defaults system` or `key-value store` for storing all settings.

Platforms: **Linux**: *Linux-based operating systems other than Android*
The evaluator shall run the application while monitoring it with the utility strace. The evaluator shall make security-related changes to its configuration. The evaluator shall verify that strace logs corresponding changes to configuration files that reside in /etc (for system-specific configuration), in the user's home directory (for user-specific configuration), or /var/lib/ (for configurations controlled by UI and not intended to be directly modified by an administrator).

Platforms: **Oracle Solaris**: *Oracle's enterprise operating system*
The evaluator shall run the application while monitoring it with the utility dtrace. The evaluator shall make security-related changes to its configuration. The evaluator shall verify that dtrace logs corresponding changes to configuration files that reside in /etc (for system-specific configuration) or in the user's home directory (for user-specific configuration).

Platforms: **Apple macOS**: *Apple's operating system for Mac devices*
The evaluator shall verify that the application stores and retrieves settings using the `NSUserDefaults` class.

If " **implement functionality to encrypt and store configuration options as defined by FDP_PRT_EXT.1 in the PP-Module for File Encryption**" is selected, for all configuration options listed in the TSS as being stored and protected using encryption, the evaluator shall examine the contents of the configuration option storage (identified in the TSS) to determine that the options have been encrypted.

FMT_SMF.1 Specification of Management Functions

FMT_SMF.1
TSS

The evaluator shall verify the TSS details how the application's management functions align with the selected management functions.

Guidance

The evaluator shall verify that every management function mandated by the PP 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 shall test the application's ability to provide the management functions by configuring the application 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.

2.1.4 Privacy (FPR)

FPR_ANO_EXT.1 User Consent for Transmission of Personally Identifiable Information

FPR_ANO_EXT.1

TSS

If "**not use PII**" is claimed, the evaluator shall verify the TSS states the application does not utilize any PII.

If "**not transmit PII over a network**" is claimed, the evaluator shall verify that the TSS makes this assertion (e.g., because it does not use network connectivity at all or if the functions for which it uses network connectivity do not involve transmission of PII). If "**require user approval before executing...**" is selected, the evaluator shall inspect the TSS documentation to verify that it identifies the functions where PII may be transmitted over a network.

Guidance

The evaluator shall verify the guidance documentation contains any instructions to configure the transmission of PII and details any prompts that would approve or deny transmission of PII.

Tests
If " **require user approval before executing...**" is selected, the evaluator shall run the application, execute each function that is claimed as being used to transmit PII, and verify that user approval is required before transmission of the PII for each function.

2.1.5 Protection of the TSF (FPT)

FPT_AEX_EXT.1 Anti-Exploitation Capabilities

FPT_AEX_EXT.1.1
TSS

The evaluator shall ensure that the TSS describes the compiler flags used to enable ASLR when the application is compiled. If any explicitly-mapped exceptions are claimed, the evaluator shall check that the TSS identifies these exceptions, describes the static memory mapping that is used, and provides justification for why static memory mapping is appropriate in this case.

Guidance

None.

Tests
The evaluator shall perform either a static or dynamic analysis to determine that no memory mappings are placed at an explicit and consistent address except for any exceptions claimed in the SFR. For these exceptions, the evaluator shall verify that this analysis shows explicit mappings that are consistent with what is claimed in the TSS. The method of doing so varies per platform. For those platforms requiring the same application running on two different systems, the evaluator may alternatively use the same device. After collecting the first instance of mappings, the evaluator must uninstall the application, reboot the device, and reinstall the application to collect the second instance of mappings.

> Platforms: **Android**: *Mobile operating systems based on Google Android*
> *The evaluator shall run the same application on two different Android systems. Both devices do not need to be evaluated, as the second device is acting only as a tool. Connect via ADB and inspect /proc/PID/maps. Ensure the two different instances share no memory mappings made by the application at the same location.*

> Platforms: **Microsoft Windows**: *Microsoft Windows operating systems*
> *The evaluator shall run the same application on two different Windows systems and run a tool that will list all memory mapped addresses for the application. The evaluator shall then verify the two different instances share no mapping locations. The Microsoft SysInternals tool, VMMap, could be used to view*

memory addresses of a running application. The evaluator shall use a tool such as Microsoft's BinScope Binary Analyzer to confirm that the application has ASLR enabled.

Platforms: **Apple iOS and iPadOS**: *Apple's mobile operating system for iPhones and iPads*
The evaluator shall perform a static analysis to search for any mmap calls (or API calls that call mmap), and ensure that no arguments are provided that request a mapping at a fixed address.

Platforms: **Linux**: *Linux-based operating systems other than Android*
The evaluator shall run the same application on two different Linux systems. The evaluator shall then compare their memory maps using `pmap -x PID` to ensure the two different instances share no mapping locations.

Platforms: **Oracle Solaris**: *Oracle's enterprise operating system*
The evaluator shall run the same application on two different Solaris systems. The evaluator shall then compare their memory maps using `pmap -x PID` to ensure the two different instances share no mapping locations.

Platforms: **Apple macOS**: *Apple's operating system for Mac devices*
The evaluator shall run the same application on two different Mac systems. The evaluator shall then compare their memory maps using `vmmap PID` to ensure the two different instances share no mapping locations.

FPT_AEX_EXT.1.2
TSS

None.

Guidance

None.

Tests
The evaluator shall verify that no memory mapping requests are made with write and execute permissions. The method of doing so varies per platform.

Platforms: **Android**: *Mobile operating systems based on Google Android*
The evaluator shall perform static analysis on the application to verify that
- *mmap is never invoked with both the PROT_WRITE and PROT_EXEC permissions, and*
- *mprotect is never invoked.*

Platforms: **Microsoft Windows**: *Microsoft Windows operating systems*
The evaluator shall use a tool such as Microsoft's BinScope Binary Analyzer to confirm that the application passes the NXCheck. The evaluator may also ensure that the `/NXCOMPAT` flag was used during compilation to verify that DEP protections are enabled for the application.

Platforms: **Apple iOS and iPadOS**: *Apple's mobile operating system for iPhones and iPads*
The evaluator shall perform static analysis on the application to verify that mprotect is never invoked with the PROT_EXEC permission.

Platforms: **Linux**: *Linux-based operating systems other than Android*
The evaluator shall perform static analysis on the application to verify that both
- *mmap is never invoked with both the PROT_WRITE and PROT_EXEC permissions, and*
- *mprotect is never invoked with the PROT_EXEC permission.*

Platforms: **Oracle Solaris**: *Oracle's enterprise operating system*

The evaluator shall perform static analysis on the application to verify that both

- *mmap is never invoked with both the PROT_WRITE and PROT_EXEC permissions, and*
- *mprotect is never invoked with the PROT_EXEC permission.*

Platforms: **Apple macOS**: *Apple's operating system for Mac devices*
The evaluator shall perform static analysis on the application to verify that mprotect is never invoked with the PROT_EXEC permission.

FPT_AEX_EXT.1.3
TSS

None.

Guidance

None.

Tests
The evaluator shall configure the platform in the necessary manner and carry out one of the prescribed tests:

Platforms: **Android**: *Mobile operating systems based on Google Android*
Applications running on Android cannot disable Android security features, therefore this requirement is met and no evaluation activity is required.

Platforms: **Microsoft Windows**: *Microsoft Windows operating systems*

If the OS platform supports Windows Defender Exploit Guard, then the evaluator shall ensure that the application can run successfully with Windows Defender Exploit Guard Exploit Protection configured with the following minimum mitigations enabled; Control Flow Guard (CFG), Randomize memory allocations (Bottom-Up ASLR), Export address filtering (EAF), Import address filtering (IAF), and Data Execution Prevention (DEP). The following link describes how to enable Exploit Protection, https://learn.microsoft.com/en-us/microsoft-365/security/defender-endpoint/enable-exploit-protection?view=o365-worldwide.

Platforms: **Apple iOS and iPadOS**: *Apple's mobile operating system for iPhones and iPads*
Applications running on iOS cannot disable security features, therefore this requirement is met and no evaluation activity is required.

Platforms: **Linux**: *Linux-based operating systems other than Android*
The evaluator shall ensure that the application can successfully run on a system with either SELinux or AppArmor enabled and in enforce mode.

Platforms: **Oracle Solaris**: *Oracle's enterprise operating system*
The evaluator shall ensure that the application can run with Solaris Trusted Extensions enabled and enforcing.

Platforms: **Apple macOS**: *Apple's operating system for Mac devices*
The evaluator shall ensure that the application can successfully run on macOS without disabling any security features.

FPT_AEX_EXT.1.4
TSS

None.

Guidance

None.

Tests

The evaluator shall run the application and determine where it writes its files. For files where the user does not choose the destination, the evaluator shall check whether the destination directory contains executable files. This varies per platform:

> Platforms: **Android**: *Mobile operating systems based on Google Android*
> *The evaluator shall run the program, mimicking normal usage, and note where all user-modifiable files are written. The evaluator shall ensure that there are no executable files stored under /data/data/package/ where package is the Java package of the application.*

> Platforms: **Microsoft Windows**: *Microsoft Windows operating systems*
> *For Windows Universal Applications the evaluator shall consider the requirement met because the platform forces applications to write all data within the application working directory (sandbox). For Windows Desktop Applications the evaluator shall run the program, mimicking normal usage, and note where all user-modifiable files are written. The evaluator shall ensure that there are no executable files stored in the same directories to which the application wrote user-modifiable files.*

> Platforms: **Apple iOS and iPadOS**: *Apple's mobile operating system for iPhones and iPads*
> *The evaluator shall consider the requirement met because the platform forces applications to write all data within the application working directory (sandbox).*

> Platforms: **Linux**: *Linux-based operating systems other than Android*
> *The evaluator shall run the program, mimicking normal usage, and note where all user-modifiable files are written. The evaluator shall ensure that there are no executable files stored in the same directories to which the application wrote user-modifiable files.*

> Platforms: **Oracle Solaris**: *Oracle's enterprise operating system*
> *The evaluator shall run the program, mimicking normal usage, and note where all user-modifiable files are written. The evaluator shall ensure that there are no executable files stored in the same directories to which the application wrote user-modifiable files.*

> Platforms: **Apple macOS**: *Apple's operating system for Mac devices*
> *The evaluator shall run the program, mimicking normal usage, and note where all user-modifiable files are written. The evaluator shall ensure that there are no executable files stored in the same directories to which the application wrote user-modifiable files.*

FPT_AEX_EXT.1.5

TSS

(Conditional: The PE or ELF automated tests fail) The evaluator shall ensure that the TSS describes the stack-based buffer overflow compiler flags.

Guidance

None.

Tests

The evaluator will inspect every native executable included in the TOE to ensure that stack-based buffer overflow protection is present.

> Platforms: **Microsoft Windows**: *Microsoft Windows operating systems*

> *Applications that run as Managed Code in the .NET Framework do not require these stack protections. Applications developed in Object Pascal using the Delphi IDE compiled with RangeChecking enabled comply with this element. For other code, the evaluator shall review the TSS and verify that the /GS flag was used during compilation. The evaluator shall run a tool like, BinSkim, that can verify the correct usage of /GS.*

- Test FPT_AEX_EXT.1.5:1:

  For PE, the evaluator will disassemble each and ensure the following sequence appears:

  ```
  mov rcx, QWORD PTR [rsp+(...)]
  xor rcx, (...)
  call (...)
  ```

- Test FPT_AEX_EXT.1.5:2:

  For ELF executables, the evaluator will ensure that each contains references to the symbol **__stack_chk_fail**.

  If these automated tests fail, the evaluator shall perform the above, conditional TSS activity.

Tools such as Canary Detector may help automate these activities.

FPT_API_EXT.1 Use of Supported Services and APIs

FPT_API_EXT.1
TSS

The evaluator shall verify that the TSS lists the platform APIs used in the application. The evaluator shall then compare the list with the supported APIs (available through e.g. developer accounts, platform developer groups) and ensure that all APIs listed in the TSS are supported.

Guidance

None.

Tests
There are no test activities for this component.

FPT_LIB_EXT.1 Use of Third Party Libraries

FPT_LIB_EXT.1
TSS

None.

Guidance

None.

Tests
The evaluator shall install the application and survey its installation directory for dynamic libraries. The evaluator shall verify that libraries found to be packaged with or employed by the application are limited to those in the assignment.

FPT_TUD_EXT.1 Support for Trusted Updates

FPT_TUD_EXT.1.1

TSS

The evaluator shall verify the TSS contains a description of the update mechanism leveraged, how new updates are checked for, how the current version is checked for, and how the updates are signed.

Guidance

The evaluator shall check to ensure the guidance includes a description of how to check for and apply new updates.

Tests

The evaluator shall check for an update using procedures described in either the application documentation or the platform documentation and verify that the application does not issue an error. If it is updated or if it reports that no update is available this requirement is considered to be met.

FPT_TUD_EXT.1.2

TSS

There are no additional TSS evaluation activities for this element.

Guidance

The evaluator shall verify guidance includes a description of how to query the current version of the application.

Tests

The evaluator shall query the application for the current version of the software according to the operational user guidance. The evaluator shall then verify that the current version matches that of the documented and installed version.

FPT_TUD_EXT.1.3

TSS

There are no additional TSS evaluation activities for this element.

Guidance

There are no additional Guidance evaluation activities for this element.

Tests

Conditional: If "not download, modify, replace or update its own binary code" is selected the evaluator shall verify that the application's executable files are not changed by the application with the following tests:

> Platforms: **Apple iOS and iPadOS**: *Apple's mobile operating system for iPhones and iPads*
>
> *The evaluator shall consider the requirement met because the platform forces applications to write all data within the application working directory (sandbox).*

> Platforms: **Microsoft Windows**: *Microsoft Windows operating systems*, **Android**: *Mobile operating systems based on Google Android*, **Linux**: *Linux-based operating systems other than Android*, **Oracle Solaris**: *Oracle's enterprise operating system*, **Apple macOS**: *Apple's operating system for Mac devices*
>
> *The evaluator shall install the application and then locate all of its executable files. The evaluator shall then, for each file, save off either a hash of the file or a copy of the file itself. The evaluator shall then run the application and exercise all features of the application as described in the ST. The evaluator shall then compare each executable file with either the saved hash or the saved copy of the files. The evaluator shall verify that these are identical.*

FPT_TUD_EXT.1.4

TSS

The evaluator shall verify that the TSS identifies how updates to the application are signed by an authorized source. The definition of an authorized source must be contained in the TSS. The evaluator shall also ensure that the TSS (or the operational guidance) describes how candidate updates are obtained.

Guidance

There are no additional Guidance evaluation activities for this element.

Tests
There are no test activities for this element.

FPT_TUD_EXT.1.5
TSS

The evaluator shall verify that the TSS identifies how the application is distributed. If "**as an additional package...**" is selected, the evaluator shall perform the tests in FPT_TUD_EXT.2.

Guidance

None.

Tests
If " **with the platform OS**" is selected, the evaluator shall perform a clean installation or factory reset to confirm that TOE software is included as part of the platform OS.

2.1.6 Trusted Path/Channels (FTP)

FTP_DIT_EXT.1 Protection of Data in Transit

FTP_DIT_EXT.1
TSS

The evaluator shall confirm the TSS describes the data transmitted, and verify it matches the selections of all *data* or *sensitive data*.

The evaluator shall confirm the TSS describes the method by which the data is protected and that is matches the chosen selections, if multiple selections are included the evaluator shall verify the TSS describes which data is sent over which trusted channels and the totality of the data type selection is covered by all chosen selections.

For platform-provided functionality, the evaluator shall verify the TSS contains the calls to the platform that the TOE is leveraging to invoke the functionality. The evaluator shall verify calls are documented by the platform vendor and non-deprecated.

For platform-provided HTTPS, IPsec, TLS, or DTLS as a client the evaluator shall verify that the TSS lists any specific calls the product uses that specifies or allows the end users to specify cipher suites, support for mutual authentication, support for session renegotiation, hash algorithms for the signature_algorithms extension in the Client Hello with the supported_signature_algorithms value, and the supported groups in the Supported Groups Extension in Client Hello. The evaluator shall verify any calls the product specifies align with the options provided in this PP and the Functional Package for Transport Layer Security (TLS), version 2.1.

For platform-provided HTTPS, IPsec, TLS, or DTLS as a server the evaluator shall verify that the TSS lists any specific calls the product uses that specifies or allows the end users to specify cipher suites, which protocols are denied connection requests, key establishment algorithms, support for mutual authentication, response to an invalid client certificate, and support for session renegotiation. The evaluator shall verify any calls the product specifies align with the options provided in this PP and the Functional Package for Transport Layer Security (TLS), version 2.1.

For platform-provided HTTPS the evaluator shall verify that the TSS lists any specific calls the product uses that specifies or allows the end users to specify the response to an invalid certificate.

For platform-provided HTTPS as a server the evaluator shall verify that the TSS lists any specific calls the product uses that specifies or allows the end users to specify cipher suites, which protocols are denied connection requests, key establishment algorithms, support for mutual authentication, response to an invalid client certificate, and support for session renegotiation. The evaluator shall verify any calls the product specifies align with the options provided in this PP and the Functional Package for Transport Layer Security (TLS), version 2.1.

For platform-provided SSH the evaluator shall verify that the TSS lists any specific calls the product uses that specifies or allows the end users to specify the applicable RFCs, the authentication methods, the limit for dropping large packets in an SSH transport connection, the SSH transport accepted algorithms, the SSH public key for public-key based authentication, The diffie-hellman-group used for key exchange, and the parameters of session rekey or termination. The evaluator shall verify any calls the product specifies align with the options provided in this PP and the Functional Package for Secure Shell (SSH), version 2.0.

Guidance

The evaluator shall confirm the guidance documentation contains any information necessary for enabling and configuring the trusted channels that have been selected.

Tests
The evaluator shall perform the following tests.

- Test FTP_DIT_EXT.1:1: If "**not transmit any data**" is selected, the evaluator shall exercise each of the TOE's identified functions, while observing the network traffic from the device and verify that no TSF initiated connections were observed during the attempts.
- Test FTP_DIT_EXT.1:2: The evaluator shall exercise the application (attempting to transmit data; for example by connecting to remote systems or websites) while capturing packets from the application. The evaluator shall verify from the packet capture that the traffic is encrypted with HTTPS, TLS, DTLS, SSH, or IPsec in accordance with the selection in the ST.
- Test FTP_DIT_EXT.1:3: The evaluator shall exercise the application (attempting to transmit data; for example by connecting to remote systems or websites) while capturing packets from the application. The evaluator shall review the packet capture and verify that no sensitive data is transmitted in the clear.
- Test FTP_DIT_EXT.1:4: The evaluator shall inspect the TSS to determine if user credentials are transmitted. If credentials are transmitted the evaluator shall set the credential to a known value. The evaluator shall capture packets from the application while causing credentials to be transmitted as described in the TSS. The evaluator shall perform a string search of the captured network packets and verify that the plaintext credential previously set by the evaluator is not found.

> Platforms: **Android**: *Mobile operating systems based on Google Android*
> *If **"not transmit any data"** is selected, the evaluator shall ensure that the application's AndroidManifest.xml file does not contain a uses-permission or uses-permission-sdk-23 tag containing android:name="android.permission.INTERNET". In this case, it is not necessary to perform the above Tests 1, 2, 3, or 4 as the platform will not allow the application to perform any network communication.*

> Platforms: **Apple iOS and iPadOS**: *Apple's mobile operating system for iPhones and iPads*
> *If **"encrypt all transmitted data"** is selected, the evaluator shall ensure that the application's Info.plist file does not contain the NSAllowsArbitraryLoads or NSExceptionAllowsInsecureHTTPLoads keys, as these keys disable iOS's Application Transport Security feature.*

2.2 Evaluation Activities for Optional SFRs

The PP-Module does not define any optional requirements.

2.3 Evaluation Activities for Selection-Based SFRs

2.3.1 Cryptographic Support (FCS)

FCS_CKM.1/AK Cryptographic Asymmetric Key Generation

FCS_CKM.1.1/AK
TSS

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

If the ST selects "**invoke platform-provided functionality**," then the evaluator shall examine the TSS to verify that it describes how the key generation functionality is invoked and that the invocation matches the algorithm and size selections for each supported platform. The evaluator shall confirm the invocation of the platform is using non-deprecated functions provided by the platform(s).

Guidance

The evaluator shall verify that the operational guidance instructs the administrator how to configure the TOE to use the selected key generation scheme(s) and key size(s) for all uses defined in this PP if any configuration is required.

Tests

If the ST selects "**implement functionality**," then the following test activities shall be carried out.

Evaluation Activity Note: The following tests may require the developer to provide access to a developer environment that provides the evaluator with tools that are not typically available to end-users of the application

Key Generation for FIPS PUB 186-5 RSA Schemes

The evaluator shall verify the implementation of RSA Key Generation by the TOE using the Key Generation test. This test verifies the ability of the TSF to correctly produce values for the key components including the public verification exponent e, the private prime factors p and q, the public modulus n and the calculation of the private signature exponent d. Key Pair generation specifies 5 ways (or methods) to generate the primes p and q. These include:

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

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

If possible, the Random Probable primes method should also be verified against a known good implementation as described above. Otherwise, the evaluator shall 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 shall require the implementation under test (IUT) to generate 10 private/public key pairs. The private key shall be generated using an approved random bit generator (RBG). To determine correctness, the evaluator shall submit the generated key pairs to the public key verification (PKV) function of a known good implementation.

FIPS 186-5 Public Key Verification (PKV) Test- For each supported NIST curve, i.e., P-384 and P-521, the evaluator shall generate 10 private/public key pairs using the key generation function of a known good implementation and modify five of the public key values so that they are incorrect, leaving five values unchanged (i.e., correct). The evaluator shall obtain in response a set of 10 PASS/FAIL values.

Key Generation for Finite-Field Cryptography (FFC)

The evaluator shall verify the implementation of the Parameters Generation and the Key Generation for FFC by the TOE using the Parameter Generation and Key Generation test. This test verifies the ability of the TSF to correctly produce values for the field prime p, the cryptographic prime q (dividing p-1), the cryptographic group generator g, and the calculation of the private key x and public key y. The Parameter generation specifies two 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 $1 \leq x \leq q-1$
- len(q) + 64 bit output of RBG, followed by a mod q-1 operation where $1 \leq x \leq q-1$.

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

comparing values generated by the TSF with those generated from a known good implementation. Verification must also confirm

- $g \neq 0,1$
- q divides p-1
- $g^{q} \bmod p = 1$
- $g^{x} \bmod p = y$

for each FFC parameter set and key pair.

Testing for FFC Schemes using safe-prime groups is done as part of testing in FCS_CKM.2.1

Key Generation for LMS/XMSS

For each supported LMS/LMSOTS pair, the evaluator will provide 1, 2, 3, 4, 5 seeds for H = 25, 20, 15, 10, 5 respectively where H = the height of the LMS tree. For each seed, the TOE will generate the corresponding public key which is to be verified by the evaluator using a known good implementation.

Key Generation for ML-DSA

The evaluator shall 10x input to the internal KeyGen function a 32-byte random seed. Verify the returned public-private key pair is correct using a known good implementation. Here internal KeyGen refers to the TOE's implementation of the function ML-DSA.KeyGen_internal(-) as described in FIPS.204.

Key Generation for ML-KEM

The evaluator shall 10x input to the internal KeyGen function a pair of 32-byte random string. Verify the returned encapsulation and decapsulation key pair is correct using a known good implementation. Here internal KeyGen refers to the TOE's implementation of the function ML-KEM.KeyGen_internal(-,-) as described in FIPS.203.

FCS_CKM.1/SK Cryptographic Symmetric Key Generation

FCS_CKM.1/SK
TSS

The evaluator shall review the TSS to determine that it describes how the functionality described by FCS_RBG_EXT.1 is invoked.

If the application is relying on random bit generation from the host platform, the evaluator shall verify the TSS includes the name/manufacturer of the external RBG and describes the function call and parameters used when calling the external DRBG function. If different external RBGs are used for different platforms, the evaluator shall verify the TSS identifies each RBG for each platform. Also, the evaluator shall verify the TSS includes a short description of the vendor's assumption for the amount of entropy seeding the external DRBG. The evaluator uses the description of the RBG functionality in FCS_RBG_EXT or documentation available for the operational environment to determine that the key size being requested is identical to the key size and mode to be used for the encryption/decryption of the user data.

Guidance

The evaluator shall verify the guidance documentation contains any information necessary to configure key sizes.

Tests
None.

FCS_CKM.2 Cryptographic Key Establishment

FCS_CKM.2.1

TSS

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

If the ST selects "**invoke platform-provided functionality**," then the evaluator shall examine the TSS to verify that it describes how the key establishment functionality is invoked and that the invocation matches the algorithm selection for each supported platform. The evaluator shall confirm the invocation of the platform is using non-deprecated functions provided by the platform(s).

Guidance

The evaluator shall verify that the operational guidance instructs the administrator how to configure the TOE to use the selected key establishment scheme(s) if configuration is required.

Tests

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

SP800-56A Key Establishment Schemes

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

Function Test

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

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

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

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

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

Validity Test

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

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

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

SP800-56B Key Establishment Schemes

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

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

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

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

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

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

FFC Schemes using “safe-prime” groups

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_DIT_EXT.1 that uses safe-prime groups. This test must be performed for each safe-prime group that each protocol uses.

ML-KEM Key Establishment Schemes

To test encapsulation the evaluator shall 10x input to the internal Encaps function a random 32-byte string and an encapsulation key. Verify the returned cipher text and shared secret is correct using a known good implementation. Here internal refers to the TOE’s implementation of the function ML-KEM.Encaps_internal(-,-) as described in FIPS.203.

To test decapsulation the evaluator shall 10x input to the internal Decaps function a cipher text and decapsulation key. Verify the returned shared secret is correct using a known good implementation. The tests should include a mix of valid and invalid/garbled cipher texts. Here internal refers to the TOE’s implementation of the function ML-KEM.Decaps_internal(-,-) as described in FIPS.203.

FCS_COP.1/Hash Cryptographic Operation - Hashing

FCS_COP.1/Hash
TSS

The evaluator shall 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.

Guidance

The evaluator shall verify the guidance documentation contains any information required for configuring the algorithm or size.

Tests

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 hashes only 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 shall 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 FCS_COP.1/Hash:1: Short Messages Test - Bit-oriented Mode. The evaluators devise an input set consisting of m+1 messages, where m is the block length of the hash algorithm. The length of the messages

range sequentially from 0 to m bits. The message text shall be pseudorandomly generated. The evaluators compute the message digest for each of the messages and ensure that the correct result is produced when the messages are provided to the TSF.

- Test FCS_COP.1/Hash:2: Short Messages Test - Byte-oriented Mode. The evaluators devise an input set consisting of m/8+1 messages, where m is the block length of the hash algorithm. The length of the messages range sequentially from 0 to m/8 bytes, with each message being an integral number of bytes. The message text shall be pseudorandomly generated. The evaluators compute the message digest for each of the messages and ensure that the correct result is produced when the messages are provided to the TSF.
- Test FCS_COP.1/Hash:3: Selected Long Messages Test - Bit-oriented Mode. The evaluators devise an input set consisting of m messages, where m is the block length of the hash algorithm. The length of the ith message is 512 + 99*i, where $1 \leq i \leq m$. The message text shall be pseudorandomly generated. The evaluators compute the message digest for each of the messages and ensure that the correct result is produced when the messages are provided to the TSF.
- Test FCS_COP.1/Hash:4: Selected Long Messages Test - Byte-oriented Mode. The evaluators devise an input set consisting of m/8 messages, where m is the block length of the hash algorithm. The length of the ith message is 512 + 8*99*i, where $1 \leq i \leq m/8$. The message text shall be pseudorandomly generated. The evaluators compute the message digest for each of the messages and ensure that the correct result is produced when the messages are provided to the TSF.
- Test FCS_COP.1/Hash:5: Pseudorandomly Generated Messages Test. This test is for byte-oriented implementations only. The evaluators randomly generate a seed that is n bits long, where n is the length of the message digest produced by the hash function to be tested. The evaluators then formulate a set of 100 messages and associated digests by following the algorithm provided in Figure 1 of [SHAVS]. The evaluators then ensure that the correct result is produced when the messages are provided to the TSF.

FCS_COP.1/KeyedHash Cryptographic Operation - Keyed-Hash Message Authentication

FCS_COP.1/KeyedHash
The evaluator shall perform the following activities based on the selections in the ST:
TSS

None.

Guidance

The evaluator shall verify the guidance documentation contains any information required for configuring the algorithm or size.

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

FCS_COP.1/SigGen Cryptographic Operation - Signing Generation

FCS_COP.1/SigGen
The evaluator shall perform the following activities based on the selections in the ST.
TSS
There are no additional TSS evaluation activities for this component.
Guidance
The evaluator shall verify the guidance documentation contains any information required for configuring the algorithm or size.
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 Test

- Test FCS_COP.1/SigGen:1: 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 shall generate 10 1024-bit long messages and obtain for each message a public key and the resulting signature values R and S. To determine correctness, the evaluator shall use the signature verification function of a known good implementation.

RSA Signature Algorithm Test

- Test FCS_COP.1/SigGen:2: Signature Generation Test. The evaluator shall verify the implementation of RSA Signature Generation by the TOE using the Signature Generation Test. To conduct this test the evaluator must generate or obtain 10 messages from a trusted reference implementation for each modulus size/SHA combination supported by the TSF. The evaluator shall have the TOE use their private key and modulus value to sign these messages. The evaluator shall verify the correctness of the TSF's signature using a known good implementation and the associated public keys to verify the signatures.

ML-DSA Test

- Test FCS_COP.1/SigGen:3: The evaluator shall 10x input to the internal Sign function a 32-byte random string, private key, and a randomly generated message. Check and confirm the value of the returned signature using a known good implementation. Here internal Sign refers to the TOE's implementation of the function ML-DSA.Sign_internal(-,-,-) as described in NIST FIPS PUB 204.

FCS_COP.1/SigVer Cryptographic Operation - Signing Verification

FCS_COP.1/SigVer
The evaluator shall perform the following activities based on the selections in the ST.
TSS

None.

Guidance

The evaluator shall verify the guidance documentation contains any information required for configuring the algorithm or size.

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 Test

- Test FCS_COP.1/SigVer:1: 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 shall generate a set of 10 1024-bit message, public key and signature tuples and modify one of the values (message, public key or signature) in five of the 10 tuples. The evaluator shall obtain in response a set of 10 PASS/FAIL values.

RSA Signature Algorithm Test

- Test FCS_COP.1/SigVer:2: Signature Verification Test. The evaluator shall perform the Signature Verification test to verify the ability of the TOE to recognize another party's valid and invalid signatures. The evaluator shall inject errors into the test vectors produced during the Signature Verification Test by introducing errors in some of the public keys, e, messages, IR format, and/or signatures. The TOE attempts to verify the signatures and returns success or failure.

LMS/XMSS Signature Algorithm Test

- Test FCS_COP.1/SigVer:3: For each supported LMS/LMSOTS pair, the evaluator generates a private/public key pair. With the private key, the evaluator generates 4 messages of length 1024 bits. The messages and public key are provided to the TOE. The signature for each message is generated with the following error types "none", “modify message”, “modify signature”, “modify header”. For "none" the message is unmodified and the signature is correct. For "modify message" the signature is for a modified message where a single bit is flipped. For "modify signature", one bit of the signature is flipped. For "modify header" the signature uses a different LMS/LMSOTS pair. Each error type is represented. For each message, signature pair the TOE returns "true" or "false" depending on whether the signature verifies or not.

ML-DSA Test

- Test FCS_COP.1/SigVer:4: The evaluator shall 10x input to the internal SigVer function, a public key, message and signature. Verify the signature. Tests should involve a mix of good and bad signatures generated using different messages, keys, etc. Here internal SigVer refers to the TOE’s implementation of the function ML-DSA.Verify_internal(-,-,-) as described in FIPS.204.

FCS_COP.1/SKC Cryptographic Operation - Encryption/Decryption

FCS_COP.1/SKC
TSS

Conditional: If AES-GCM is selected, the evaluator shall verify the tag length is described in the TSS and that a tag length of at least 128 is used unless the following "Appendix C: Requirements and Guidelines for Using Short Tags" is being followed from NIST SP 800-38D.

Guidance

The evaluator checks the guidance documents to determine that any configuration that is required to be done to configure the functionality for the required modes and key size is present.

Tests

The evaluator shall perform all of the following tests for each algorithm implemented by the TSF 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 shall compare the resulting values to those obtained by submitting the same inputs to a known good implementation.

- KAT-1. To test the encrypt functionality of AES-CBC, the evaluator shall supply a set of 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 256-bit all- zeros key. To test the decrypt functionality of AES-CBC, the evaluator shall perform the same test as for encrypt, using 10 ciphertext values as input and AES-CBC decryption.

- KAT-2. To test the encrypt functionality of AES-CBC, the evaluator shall supply a set of 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. The keys shall be 256-bit keys. To test the decrypt functionality of AES-CBC, the evaluator shall perform the same test as for encrypt, using an all-zero ciphertext value as input and AES-CBC decryption.

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

- KAT-4. To test the encrypt functionality of AES-CBC, the evaluator shall supply the set of 128 plaintext values described below and obtain the 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 shall have the leftmost i bits be ones and the rightmost 128-i bits be zeros, for i in [1,128].

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

AES-CBC Multi-Block Message Test

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

AES-CBC Monte Carlo Tests

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

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

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

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

AES-GCM Monte Carlo Tests

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

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

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

AES-XTS Tests

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

512 bit (for AES-256) keys

Three data unit (i.e., plaintext) lengths. One of the data unit lengths shall be a non-zero integer multiple of 128 bits, if supported. One of the data unit lengths shall not 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, the evaluator shall obtain the ciphertext that results from XTS-AES encrypt.

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

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

AES-CCM Tests

It is not recommended that evaluators use values obtained from static sources such as http://csrc.nist.gov/groups/STM/cavp/documents/mac/ccmtestvectors.zip or use values not generated expressly to exercise the AES-CCM implementation.

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

- Keys: All supported and selected key sizes (e.g., 256 bits).
- Associated Data: Two or three values for associated data length: The minimum (≥ 0 bytes) and maximum (≤ 32 bytes) supported associated data lengths, and 2^16 (65536) bytes, if supported.

- Payload: Two values for payload length: The minimum (≥ 0 bytes) and maximum (≤ 32 bytes) supported payload lengths.
- Nonces: All supported nonce lengths (7, 8, 9, 10, 11, 12, 13) in bytes.
- Tag: All supported tag lengths (4, 6, 8, 10, 12, 14, 16) in bytes.

The testing for CCM consists of five tests. To determine correctness in each of the below tests, the evaluator shall compare the ciphertext with the result of encryption of the same inputs with a known good implementation.

Variable Associated Data Test

For each supported key size and associated data length, and any supported payload length, nonce length, and tag length, the evaluator shall supply one key value, one nonce value, and 10 pairs of associated data and payload values, and obtain the resulting ciphertext.

Variable Payload Test

For each supported key size and payload length, and any supported associated data length, nonce length, and tag length, the evaluator shall supply one key value, one nonce value, and 10 pairs of associated data and payload values, and obtain the resulting ciphertext.

Variable Nonce Test

For each supported key size and nonce length, and any supported associated data length, payload length, and tag length, the evaluator shall supply one key value, one nonce value, and 10 pairs of associated data and payload values, and obtain the resulting ciphertext.

Variable Tag Test

For each supported key size and tag length, and any supported associated data length, payload length, and nonce length, the evaluator shall supply one key value, one nonce value, and 10 pairs of associated data and payload values, and obtain the resulting ciphertext.

Decryption-Verification Process Test

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 sets of input plus ciphertext, and obtain the decrypted payload. Ten of the 15 input sets supplied should fail verification and five should pass.

AES-CTR Tests

Test 1: Known Answer Tests (KATs)

There are four Known Answer Tests (KATs) described below. For all KATs, the plaintext, IV, and ciphertext values shall be 128-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 shall compare the resulting values to those obtained by submitting the same inputs to a known good implementation.

- KAT-1. To test the encrypt functionality, the evaluator shall supply a set of 5 plaintext values and obtain the ciphertext value that results from 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 256-bit all zeros key. To test the decrypt functionality, the evaluator shall perform the same test as for encrypt, using 5 ciphertext values as input.

- KAT-2. To test the encrypt functionality, the evaluator shall supply a set of 5 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. Five of the key values shall be 256-bit keys. To test the decrypt functionality, the evaluator shall perform the same test as for encrypt, using an all zero ciphertext value as input.

- KAT-3. To test the encrypt functionality, the evaluator shall supply the key values described below and obtain the ciphertext values that result from AES encryption of an all zeros plaintext using the given key values an an IV of all zeros. The set of keys shall have 256 256-bit keys. Key_i shall have the leftmost i bits be ones and the rightmost N-i bits be zeros, for i in [1, N]. To test the decrypt functionality, the evaluator shall supply the 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 first set of key/ciphertext pairs shall have 256 256-bit pairs. Key_i shall have the leftmost i bits be ones and the rightmost N-i bits be zeros for i in [1, N]. The ciphertext value in each pair shall be the value that results in an all zeros plaintext when decrypted with its corresponding key.

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

Test 2: Multi-Block Message Test

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

Test 3: Monte-Carlo Test

For AES-CTR mode, perform the Monte Carlo Test for ECB Mode on the encryption engine of the counter mode implementation. There is no need to test the decryption engine.

The evaluator shall test the encrypt functionality using 100 plaintext/key pairs. 100 of these shall use 256-bit keys. The plaintext values shall be 128-bit blocks. For each pair, 1000 iterations shall be run as follows:

```
For AES-ECB mode
# Input: PT, Key
         for i = 1 to 1000:
             CT[i] = AES-ECB-Encrypt(Key, PT)
             PT = CT[i]
```

The ciphertext computed in the 1000th iteration 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.

FCS_HTTPS_EXT.1 HTTPS Protocol

FCS_HTTPS_EXT.1
TSS

The evaluator shall examine the TSS to verify that it describes the TSF's HTTPS implementation as a client, server, or both, and that if the TSF implements an HTTPS server, whether this supports mutual authentication. Additionally, the evaluator shall examine the TSS to verify that it includes enough detail is provided to explain how the implementation complies with RFC 2818.

Guidance

The evaluator shall review the operational guidance to ensure that it includes any information necessary for configuring HTTPS in alignment with RFC 2818, or whether this is the default behavior of the TOE.

Tests

- Test FCS_HTTPS_EXT.1:1: Conditional: If the TOE implements HTTPS as a client, for each HTTPS client interface, the evaluator shall attempt to establish an HTTPS connection from the TOE to an external server, observe the traffic with a packet analyzer, and verify that the connection succeeds and that the traffic is identified as TLS or HTTPS.
- Test FCS_HTTPS_EXT.1:2: Conditional: If the TOE implements HTTPS as a server, for each HTTPS server interface, the evaluator shall attempt to establish an HTTPS connection to the TOE using a client, observe the traffic with a packet analyzer, and verify that the connection succeeds and that the traffic is identified as TLS or HTTPS.

FCS_HTTPS_EXT.2 HTTPS Support for Authentication

FCS_HTTPS_EXT.2.1
TSS
The evaluator shall verify that the TSS describes the TOE's behavior when an invalid peer certificate is presented to it during establishment of an HTTPS connection.
Guidance
If the TOE's response to being presented with an invalid certificate is configurable, the evaluator shall verify that the operational guidance includes instructions for configuring this behavior and what the configurable options are. If this includes configuration to a permissive setting where an invalid certificate may be accepted without administrator intervention, the evaluator shall verify that the operational guidance includes sufficient warning of the potential security risks of applying this setting.
Tests

The evaluator shall attempt to establish an HTTPS connection using the TOE, present an invalid peer certificate, and verify that the TSF behaves in the manner specified in the TSF in response. If this behavior is configurable, the evaluator shall iterate this test as necessary to exercise each configuration option and verify that the TSF behaves in the configured manner.

Other tests are performed in conjunction with the Functional Package for Transport Layer Security (TLS), version 2.1 and the Functional Package for X.509.

FCS_PBKDF_EXT.1 Password Conditioning

FCS_PBKDF_EXT.1
TSS

Support for PBKDF: The evaluator shall examine the password hierarchy described in the TSS to ensure that the formation of all password based derived keys is described and that the key sizes match that described by the ST author. The evaluator shall check that the TSS describes the method by which the password/passphrase is first encoded and then fed to the SHA algorithm. The settings for the algorithm (padding, blocking, etc.) shall be described, and the evaluator shall verify that these are supported by the selections in this component as well as the selections concerning the hash function itself. The evaluator shall verify that the TSS contains a description of how

the output of the hash function is used to form the submask that will be input into the function. For the NIST SP 800-132-based conditioning of the password/passphrase, the required evaluation activities will be performed when doing the evaluation activities for the appropriate requirements (FCS_COP.1.1/KeyedHash). No explicit testing of the formation of the submask from the input password is required. FCS_PBKDF_EXT.1: The evaluator shall verify the TSS describes the salt size and verify that the salt size aligns with NIST SP 800-132 with a minimum random length of 128 bits.

Guidance

The evaluator shall confirm the guidance documentation contains any information necessary for configuring the password conditioning if any configuration is supported.

Tests
None.

FCS_RBG.1 Random Bit Generation (RBG)

FCS_RBG.1.1
TSS
The evaluator shall verify that the TSS identifies the DRBGs used by the TOE.
Guidance
If the DRBG functionality is configurable, the evaluator shall verify that the operational guidance includes instructions on how to configure this behavior.
Tests

The evaluator shall perform the following tests:

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

If the DRBG has prediction resistance enabled, each trial consists of (1) instantiate DRBG, (2) generate the first block of random bits (3) generate a second block of random bits (4) uninstantiate. The evaluator verifies that the second block of random bits is the expected value. The evaluator shall generate eight input values for each trial. The first is a count (0 – 14). The next three are entropy input, nonce, and personalization string for the instantiate operation. The next two are additional input and entropy input for the first call to generate. The final two are additional input and entropy input for the second call to generate. These values are randomly generated. "generate one block of random bits" means to generate random bits with number of returned bits equal to the Output Block Length (as defined in NIST SP 800-90A).

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

The following 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 shall use personalization strings of two different lengths. If the implementation does not use a personalization string, no value needs to be supplied.
- **Additional input:** The additional input bit lengths have the same defaults and restrictions as the personalization string lengths.

FCS_RBG.1.2
Documentation will be produced - and the evaluator shall perform the activities - in accordance with Appendix D - Entropy Documentation and Assessment appendix and the Clarification to the Entropy Documentation and Assessment Annex.
TSS
There are no additional TSS evaluation activities for this element.

Guidance
There are no additional Guidance evaluation activities for this element.

Tests
There are no test activities for this element.

FCS_RBG.1.3
TSS
The evaluator shall verify that the TSS identifies how the DRBG state is updated, and the situations under which this may occur.
Guidance
If the ST claims that the DRBG state can be updated on demand, the evaluator shall verify that the operational guidance has instructions for how to perform this operation.
Tests
There are no test activities for this element.

FCS_RBG.2 Random Bit Generation (External Seeding)

FCS_RBG.2
The evaluator shall examine the entropy documentation required by FCS_RBG.1.2 to verify that it identifies, for each DRBG function implemented by the TOE, the TSF external interface used to seed the TOE's DRBG. The evaluator shall verify that this includes the amount of sampled data and the min-entropy rate of the sampled data such that it can be determined that sufficient entropy can be made available for the highest strength keys that the TSF can generate (e.g., 256 bits). If the seed data cannot be assumed to have full entropy (e.g., the min-entropy of the sampled bits is less than 1), the evaluator shall ensure that the entropy documentation describes the method by which the TOE estimates the amount of entropy that has been accumulated to ensure that sufficient data is collected and any conditioning that the TSF applies to the output data to create a seed of sufficient size with full entropy.
TSS
There are no additional TSS evaluation activities for this component.
Guidance
There are no additional Guidance evaluation activities for this component.
Tests
There are no test activities for this component.

FCS_RBG.3 Random Bit Generation (Internal Seeding - Single Source)

FCS_RBG.3
The evaluator shall examine the entropy documentation required by FCS_RBG.1.2 to verify that it identifies, for each DRBG function implemented by the TOE, the TSF noise source used to seed the TOE's DRBG. The evaluator shall verify that this includes the amount of sampled data and the min-entropy rate of the sampled data such that it

can be determined that sufficient entropy can be made available for the highest strength keys that the TSF can generate (e.g., 256 bits). If the seed data cannot be assumed to have full entropy (e.g., the min-entropy of the sampled bits is less than 1), the evaluator shall ensure that the entropy documentation describes the method by which the TOE estimates the amount of entropy that has been accumulated to ensure that sufficient data is collected and any conditioning that the TSF applies to the output data to create a seed of sufficient size with full entropy.
TSS
There are no additional TSS evaluation activities for this component.
Guidance
There are no additional Guidance evaluation activities for this component.
Tests
There are no test activities for this component.

FCS_RBG.4 Random Bit Generation (Internal Seeding - Multiple Sources)

FCS_RBG.4
The evaluator shall examine the entropy documentation required by FCS_RBG.1.2 to verify that it identifies, for each DRBG function implemented by the TOE, each TSF noise source used to seed the TOE's DRBG. The evaluator shall verify that this includes the amount of sampled data and the min-entropy rate of the sampled data from each data source.
TSS
There are no additional TSS evaluation activities for this component.
Guidance
There are no additional Guidance evaluation activities for this component.
Tests
There are no test activities for this component.

FCS_RBG.5 Random Bit Generation (Combining Noise Sources)

FCS_RBG.5
Using the entropy sources specified in FCS_RBG.4, the evaluator shall examine the entropy documentation required by FCS_RBG.1.2 to verify that it describes the method by which the various entropy sources are combined into a single seed. This should include an estimation of the rate at which each noise source outputs data and whether this is dependent on any system-specific factors so that each source's relative contribution to the overall entropy is understood. The evaluator shall verify that the resulting combination of sampled data and the min-entropy rate of the sampled data is described in sufficient detail to determine that sufficient entropy can be made available for the highest strength keys that the TSF can generate (e.g., 256 bits). If the seed data cannot be assumed to have full entropy (e.g., the min-entropy of the sampled bits is less than 1), the evaluator shall ensure that the entropy documentation describes the method by which the TOE estimates the amount of entropy that has been accumulated to ensure that sufficient data is collected and any conditioning that the TSF applies to the output data to create a seed of sufficient size with full entropy.
TSS
There are no additional TSS evaluation activities for this component.
Guidance
There are no additional Guidance evaluation activities for this component.
Tests
There are no test activities for this component.

FCS_SNI_EXT.1 Cryptographic Operation (Salt, Nonce, and Initialization Vector Generation)

FCS_SNI_EXT.1
TSS

If salts are used, the evaluator shall ensure the TSS describes how salts are generated. The evaluator shall confirm that the salt is generating using an RBG described in FCS_RBG_EXT.1.

If nonces are used the evaluator shall ensure the TSS describes how nonces are created verify they are a minimum of 64 bits in size.

If initialization vectors (IV) are used the evaluator shall ensure the TSS describes how IVs and tweaks are handled based on the AES mode. The evaluator shall confirm that the IVs and tweaks meet the stated requirements for each AES mode.

If using a GCM IV, the evaluator shall confirm the TSS describes the GCM IV construction and that it matches one of two allowed construction methods given in Section 8.2 of SP800-38D.

Guidance

None.

Tests
None.

2.3.2 Protection of the TSF (FPT)

FPT_FLS.1 Failure with Preservation of Secure State

FPT_FLS.1
TSS
The evaluator shall verify that the TSF describes how the TOE enters an error state in the event of a DRBG self-test failure.
Guidance
The evaluator shall verify that the guidance documentation describes the error state that results from a DRBG self-test failure and the actions that a user or administrator should take in response to attempt to resolve the error state.
Tests
There are no test activities for this component.

FPT_TST.1 TSF Self-Testing

FPT_TST.1
TSS

The evaluator shall examine the TSS to ensure that it details the self-tests that are run by the TSF along with how they are run. This description should include an outline of what the tests are actually doing. The evaluator shall ensure that the TSS makes an argument that the tests are sufficient to demonstrate that the DRBG is operating correctly.

Note that this information may also be placed in the entropy documentation specified by .

Guidance

If a self-test can be executed at the request of an authorized user, the evaluator shall verify that the operational guidance provides instructions on how to execute that self-test.

Tests

For each self-test, the evaluator shall verify that evidence is produced that the self-test is executed when specified by FPT_TST.1.1.

If a self-test can be executed at the request of an authorized user, the evaluator shall verify that following the steps documented in the operational guidance to perform the self-test will result in execution of the self-test.

FPT_TUD_EXT.2 Integrity for Installation and Update

FPT_TUD_EXT.2.1
TSS

The evaluator shall verify that the TSS describes how the application is distributed and verify that description aligns with the selections in the ST.

Guidance

None.

Tests
If a container image is claimed, the evaluator shall verify that application updates are distributed as container images. If the format of the platform-supported package manager is claimed, the evaluator shall verify that application updates are distributed in the format supported by the platform. This varies per platform:

> Platforms: **Android**: *Mobile operating systems based on Google Android*
> *The evaluator shall ensure that the application is packaged in the Android application package (APK) format.*

> Platforms: **Microsoft Windows**: *Microsoft Windows operating systems*
> *The evaluator shall ensure that the application is packaged in the standard Windows Installer (.MSI) format, the Windows Application Software (.EXE) format signed using the Microsoft Authenticode process, or the Windows Universal Application package (.APPX) format. See https://msdn.microsoft.com/en-us/library/ms537364(v=vs.85).aspx for details regarding Authenticode signing.*

> Platforms: **Apple iOS and iPadOS**: *Apple's mobile operating system for iPhones and iPads*
> *The evaluator shall ensure that the application is packaged in the IPA format.*

> Platforms: **Linux**: *Linux-based operating systems other than Android*
> *The evaluator shall ensure that the application is packaged in the format of the package management infrastructure of the chosen distribution. For example, applications running on Red Hat and Red Hat derivatives shall be packaged in `RPM` format. Applications running on Debian and Debian derivatives shall be packaged in `DEB` format.*

> Platforms: **Oracle Solaris**: *Oracle's enterprise operating system*
> *The evaluator shall ensure that the application is packaged in the PKG format.*

> Platforms: **Apple macOS**: *Apple's operating system for Mac devices*
> *The evaluator shall ensure that the application is packaged in the DMG format, the PKG format, or the MPKG format.*

FPT_TUD_EXT.2.2
TSS

None.

Guidance

The evaluator shall verify the guidance documentation details how uninstallation of the application is performed.

Tests

> Platforms: **Android**: *Mobile operating systems based on Google Android*
> *The evaluator shall consider the requirement met because the platform forces applications to write all data within the application working directory (sandbox).*

> Platforms: **Apple iOS and iPadOS**: *Apple's mobile operating system for iPhones and iPads*
> *The evaluator shall consider the requirement met because the platform forces applications to write all data within the application working directory (sandbox).*

For all other platforms, the evaluator shall record the path of every file on the entire file system prior to installation of the application, and then install and run the application. Afterward, the evaluator shall uninstall the application, and compare the resulting file system to the initial record to verify that no files, other than configuration, output, and audit or log files, have been added to the file system.

FPT_TUD_EXT.2.3
TSS

The evaluator shall verify that the TSS identifies how the application installation package is signed by an authorized source. The definition of an authorized source must be contained in the TSS.

Guidance

None.

Tests
Conditional: if "the application can verify them using" is selected the evaluator shall perform the following tests:

- Test FPT_TUD_EXT.2.3:1: The evaluator shall ensure that the update has a digital signature belonging to the vendor prior to its installation. The evaluator shall 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 shall ensure that the modified update fails to install.
- Test FPT_TUD_EXT.2.3:2: The evaluator shall ensure that the update has a digital signature belonging to the vendor. The evaluator shall then attempt to install the update (or permit installation to continue). The evaluator shall ensure that the update successfully installs.

2.4 Evaluation Activities for Objective SFRs

2.4.1 Protection of the TSF (FPT)

FPT_API_EXT.2 Use of Supported Services and APIs

FPT_API_EXT.2
TSS

The evaluator shall verify that the TSS lists any IANA MIME media types (as described by http://www.iana.org/assignments/media-types) for all formats the application processes and that it maps those formats to parsing services provided by the platform.

The API shall be verified in FPT_API_EXT.1.

Guidance

None.

Tests

None.

FPT_IDV_EXT.1 Software Identification and Versions

FPT_IDV_EXT.1
TSS

None.

Guidance

None.

Tests
The evaluator shall install the application and check for a .swidtag file. The evaluator shall open the file and verify that is contains at least a SoftwareIdentity element and an Entity element.

2.5 Evaluation Activities for Implementation-dependent SFRs

The PP-Module does not define any implementation-dependent requirements.

3 Evaluation Activities for SARs

3.1 Class ADV: Development

ADV_FSP.1 Basic Functional Specification (ADV_FSP.1)

ADV_FSP.1
There are no specific evaluation activities associated with these SARs, except ensuring the information is provided. The functional specification documentation is provided to support the evaluation activities described in Section , and other activities described for AGD, ATE, and AVA SARs. The requirements on the content of the functional specification information is implicitly assessed by virtue of the other evaluation activities being performed; if the evaluator is unable to perform an activity because there is insufficient interface information, then an adequate functional specification has not been provided.

3.2 Class AGD: Guidance Documentation

AGD_OPE.1 Operational User Guidance (AGD_OPE.1)

AGD_OPE.1

Some of the contents of the operational guidance will be verified by the evaluation activities in Section and evaluation of the TOE according to the . The following additional information is also required.

If cryptographic functions are provided by the TOE, the operational guidance shall contain instructions for configuring the cryptographic engine associated with the evaluated configuration of the TOE. It shall provide a warning to the administrator that use of other cryptographic engines was not evaluated nor tested during the CC evaluation of the TOE.

The documentation must describe the process for verifying updates to the TOE by verifying a digital signature – this may be done by the TOE or the underlying platform.

The evaluator shall verify that this process includes the following steps:

- Instructions for obtaining the update itself. This should include instructions for making the update accessible to the TOE (e.g., placement in a specific directory).
- Instructions for initiating the update process, as well as discerning whether the process was successful or unsuccessful. This includes generation of the digital signature. The TOE will likely contain security functionality that does not fall in the scope of evaluation under this PP. The operational guidance shall make it clear to an administrator which security functionality is covered by the evaluation activities.

AGD_PRE.1 Preparative Procedures (AGD_PRE.1)

AGD_PRE.1
As indicated in the introduction above, there are significant expectations with respect to the documentation—especially when configuring the operational environment to support TOE functional requirements. The evaluator shall check to ensure that the guidance provided for the TOE adequately addresses all platforms claimed for the TOE in the ST.

3.3 Class ALC: Life-cycle Support

ALC_CMC.1 Labeling of the TOE (ALC_CMC.1)

ALC_CMC.1
The evaluator shall check the ST to ensure that it contains an identifier (such as a product name/version number) that specifically identifies the version that meets the requirements of the ST. Further, the evaluator shall check the operational guidance and TOE samples received for testing to ensure that the version number is consistent with that in the ST. If the vendor maintains a website advertising the TOE, the evaluator shall examine the information on the website to ensure that the information in the ST is sufficient to distinguish the product.

ALC_CMS.1 TOE CM Coverage (ALC_CMS.1)

ALC_CMS.1
The "evaluation evidence required by the SARs" in this PP is limited to the information in the ST coupled with the guidance provided to administrators and users under the AGD requirements. By ensuring that the TOE is specifically identified and that this identification is consistent in the ST and in the AGD guidance (as done in the evaluation activity for ALC_CMC.1), the evaluator implicitly confirms the information required by this component. Life-cycle support is targeted aspects of the developer’s life-cycle and instructions to providers of applications for the developer’s devices, rather than an in-depth examination of the TSF manufacturer’s development and configuration management process. This is not meant to diminish the critical role that a developer’s practices play in contributing to the overall trustworthiness of a product; rather, it’s a reflection on the information to be made available for evaluation.

The evaluator shall ensure that the developer has identified (in guidance documentation for application developers concerning the targeted platform) one or more development environments appropriate for use in developing applications for the developer’s platform. For each of these development environments, the developer shall provide

information on how to configure the environment to ensure that buffer overflow protection mechanisms in the environment(s) are invoked (e.g., compiler flags). The evaluator shall ensure that this documentation also includes an indication of whether such protections are on by default, or have to be specifically enabled. The evaluator shall ensure that the TSF is uniquely identified (with respect to other products from the TSF vendor), and that documentation provided by the developer in association with the requirements in the ST is associated with the TSF using this unique identification.

ALC_FLR.1 Basic Flaw Remediation (ALC_FLR.1)

ALC_FLR.1
The evaluator shall inspect the TSS and verify it identifies how to access the flaw remediation procedures.

ALC_FLR.2 Flaw Reporting Procedures (ALC_FLR.2)

ALC_FLR.2
The evaluator shall inspect the TSS and verify it identifies how to access the flaw remediation procedures.

The evaluator shall inspect the guidance document and verify it describes how to access the flaw remediation guidance.

ALC_FLR.3 Systematic Flaw Remediation (ALC_FLR.3)

ALC_FLR.3
The evaluator shall inspect the TSS and verify it identifies how to access the flaw remediation procedures.

The evaluator shall inspect the guidance document and verify it describes how to access the flaw remediation guidance.

ALC_TSU_EXT.1 Timely Security Updates

ALC_TSU_EXT.1

The evaluator shall 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 shall verify that this description addresses the entire application. The evaluator shall also verify that, in addition to the TOE developer’s process, any third-party processes are also addressed in the description. The evaluator shall also verify that each mechanism for deployment of security updates is described.

The evaluator shall 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 TOE patching this vulnerability, to include any third-party or carrier delays in deployment. The evaluator shall verify that this time is expressed in a number or range of days.

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

3.4 Class ATE: Tests

ATE_IND.1 Independent Testing – Conformance (ATE_IND.1)

ATE_IND.1

The evaluator shall prepare a test plan and report documenting the testing aspects of the system, including any application crashes during testing. The evaluator shall determine the root cause of any application crashes and include that information in the report. The test plan covers all of the testing actions contained in the and the body of this PP's evaluation activities.

While it is not necessary to have one test case per test listed in an evaluation activity, the evaluator must document in the test plan that each applicable testing requirement in the ST is covered. The test plan identifies the platforms to be tested, and for those platforms not included in the test plan but included in the ST, the test plan provides a justification for not testing the platforms. This justification must address the differences between the tested platforms and the untested platforms, and make an argument that the differences do not affect the testing to be performed. It is not sufficient to merely assert that the differences have no effect; rationale must be provided. If all platforms claimed in the ST are tested, then no rationale is necessary. The test plan describes the composition of each platform to be tested, and any setup that is necessary beyond what is contained in the AGD documentation. It should be noted that the evaluator is expected to follow the AGD documentation for installation and setup of each platform either as part of a test or as a standard pre-test condition. This may include special test drivers or tools. For each driver or tool, an argument (not just an assertion) should be provided that the driver or tool will not adversely affect the performance of the functionality by the TOE and its platform.

This also includes the configuration of the cryptographic engine to be used. The cryptographic algorithms implemented by this engine are those specified by this PP and used by the cryptographic protocols being evaluated (e.g., SSH). The test plan identifies high-level test objectives as well as the test procedures to be followed to achieve those objectives. These procedures include expected results.

The test report (which could just be an annotated version of the test plan) details the activities that took place when the test procedures were executed, and includes the actual results of the tests. This shall be a cumulative account, so if there was a test run that resulted in a failure; a fix installed; and then a successful re-run of the test, the report would show a "fail" and "pass" result (and the supporting details), and not just the "pass" result.

3.5 Class AVA: Vulnerability Assessment

AVA_VAN.1 Vulnerability Survey (AVA_VAN.1)

AVA_VAN.1

The evaluator shall generate a report to document their findings with respect to this requirement. This report could physically be part of the overall test report mentioned in ATE_IND, or a separate document. The evaluator performs a search of public information to find vulnerabilities that have been found in similar applications with a particular focus on network protocols the application uses and document formats it parses.

The evaluator documents the sources consulted and the vulnerabilities found in the report.

For each vulnerability found, the evaluator either provides a rationale with respect to its non-applicability, or the evaluator formulates a test (using the guidelines provided in ATE_IND) to confirm the vulnerability, if suitable. Suitability is determined by assessing the attack vector needed to take advantage of the vulnerability. If exploiting the vulnerability requires expert skills and an electron microscope, for instance, then a test would not be suitable and an appropriate justification would be formulated.

Platforms: **Microsoft Windows**: *Microsoft Windows operating systems*, **Linux**: *Linux-based operating systems other than Android*, **Apple macOS**: *Apple's operating system for Mac devices*, **Oracle Solaris**: *Oracle's enterprise operating system*

The evaluator shall also run a virus scanner with the most current virus definitions against the application files and verify that no files are flagged as malicious.

4 Required Supplementary Information

This Supporting Document has no required supplementary information beyond the ST, operational guidance, and testing.

Appendix A - References

Identifier	Title
[CC]	Common Criteria for Information Technology Security Evaluation - • Part 1: Introduction and General Model, CCMB-2017-04-001, Version 3.1 Revision 5, April 2017. • Part 2: Security Functional Components, CCMB-2017-04-002, Version 3.1 Revision 5, April 2017. • Part 3: Security Assurance Components, CCMB-2017-04-003, Version 3.1 Revision 5, April 2017.
[OMB]	Reporting Incidents Involving Personally Identifiable Information and Incorporating the Cost for Security in Agency Information Technology Investments, OMB M-06-19, July 12, 2006.