Supporting Document
Mandatory Technical Document

Evaluation Activities for Application Software cPP

<Date>

Version 0.39

CCDB-*<Reference from CCDB, in the form 'YYYY-MM-nnn'>*

1. Document Purpose and Overview

This document serves as a template Supporting Document (SD) for use by international Technical Communities (iTCs). SDs are complementary to collaborative Protection Profiles (cPPs) and define the evaluation activities (EAs) required to satisfy the Security Functional Requirements (SFRs) in the cPP.

The intent of this SD template is to provide iTCs with a method for developing SDs that address the CEM work units. International Technical Communities are expected to tailor this template to appropriately address the technology type in question.

Sections of this document contain notes in <*brackets and italics*> for the SD author to take into consideration when developing the SD and should be removed by the iTC prior to finalizing the SD.

Similarly, some sections of this template are populated with examples to illustrate the structure of the section. They are expected to be replaced by the specific EAs deemed necessary by the iTC.

Foreword

This is a supporting document, intended to complement the Common Criteria version 3 and the associated Common Evaluation Methodology for Information Technology Security Evaluation.
Supporting documents 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 supporting document. The usage of the latter class is not only mandatory, but certificates issued as a result of their application are recognized under the CCRA.
This supporting document has been developed by the Application Software iTC and is designed to be used to support the evaluations of products against the cPPs identified in section 1.1.

Document history:
<TBD>

Acknowledgements:
This Supporting Document was developed by the Application Software international Technical Community with representatives from industry, Government agencies, Common Criteria Test Laboratories, and members of academia.

Contents

List of tables

1 Introduction

1.1 Technology Area and Scope of Supporting Document

1 This Supporting Document (SD) is mandatory for evaluations of products that claim conformance to any of the following cPP(s):

a) *Collaborative Protection Profile for Application Software – [Month, Year]*

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

1.2 Structure of the Document

3 Evaluation Activities can be defined for both SFRs and Security Assurance Requirements (SARs). These are defined in separate sections of this SD. The EAs associated with the SFRs are considered to be interpretations of applying the appropriate SAR activity. For instance, activities associated with testing are representative of what is required by ATE_IND.1.

4 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 agreed with the Certification Body for the evaluation.

5 In general, if all EAs (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'.

6 In some cases, the Common Evaluation Methodology (CEM) work units have been interpreted to require the evaluator to perform specific EAs. In these instances, EAs will be specified in Section 2 (*Evaluation Activities for Mandatory* SFRs*),* Section 5 (Evaluation Activities for Objective Requirements*),* and possibly Section 3 (*Evaluation Activities for Optional Requirements) and Section 4 (Evaluation Activities for Selection-Based Requirements)*. In cases where there are no CEM interpretations, the CEM activities are to be used to determine if SARs are satisfied and references to the CEM work units are identified as being the sole EAs to be performed.

7 Finally, there are cases where EAs have rephrased CEM work units to provide clarity on what is required. The EAs are reworded for clarity and interpret the CEM work units such that they will result in more objective and repeatable

actions by the evaluator. In these cases, the EA supplements the CEM work unit. These EAs will be specified in Section 6 (*Evaluation Activities for SARs)*.

1.3 Terminology

1.3.1 Glossary

8 For definitions of standard CC terminology, see [CC] part 1.

Term	Meaning
Assurance	Grounds for confidence that a TOE meets the SFRs [CC1].
Required Supplementary Information	Information that is not necessarily included in the Security Target or operational guidance, and that may not necessarily be public. Examples of such information could be entropy analysis, or description of a cryptographic key management architecture used in (or in support of) the TOE. The requirement for any such supplementary information will be identified in the relevant cPP (see description in Section 6).
Target of Evaluation	A set of software, firmware and/or hardware possibly accompanied by guidance. [CC1]
TOE Security Functionality (TSF)	A set consisting of all hardware, software, and firmware of the TOE that must be relied upon for the correct enforcement of the SFRs. [CC1]
TSF Data	Data for the operation of the TSF upon which the enforcement of the requirements relies.

1.3.2 Acronyms

Acronym	Meaning
AES	Advanced Encryption Standard
CA	Certificate Authority
CBC	Cipher Block Chaining
CCM	Counter with CBC-Message Authentication Code
cPP	Collaborative protection Profile
DSA	Digital Signature Algorithm
ECDSA	Elliptic Curve Digital Signature Algorithm
FIPS	Federal Information Processing Standards
GCM	Galois Counter Mode
HMAC	Keyed-Hash Message Authentication Code
KDF	Key Derivation Function
NIST	National Institute of Standards and Technology
PP	Protection Profile
RBG	Random Bit Generator
RSA	Rivest Shamir Adleman Algorithm
SHA	Secure Hash Algorithm
SFR	Security Functional Requirement
ST	Security Target
TOE	Target of Evaluation

TSF	TOE Security Functionality
TSS	TOE Summary Specification

2 Evaluation Activities for Mandatory SFRs

9 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 work units that are performed in Section 6 (Evaluation Activities for SARs).

10 Regarding design descriptions (designated by the subsections labelled 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 work unit 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 cPP.

11 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 work units ADV_FSP.1-7, AGD_OPE.1-4, and AGD_OPE.1-5.

12 Finally, the subsection labelled Tests is where the iTC has 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 work units 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 Cryptographic Support (FCS)

2.1.1 Random Bit Generation Services (FCS_RBG)

2.1.1.1 FCS_RBG_EXT.2

2.1.1.1.1 TSS

13 [Conditional] 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.

14 [Conditional] If **invoke platform-provided DRBG functionality** is selected, 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 in Test activities.

15 [Conditional] If **implement DRBG functionality** is selected, the evaluator shall ensure that FCS_RBG_EXT.1 is included in the ST.

2.1.1.1.2 Operational Guidance

16 No activities specified.

2.1.1.1.3 Test

17 [Conditional] If **invoke platform-provided DRBG functionality t**he evaluator shall decompile the application binary using an 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.

18 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.

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

20 Test 1: [conditional] For Windows platform, the evaluator shall verify that BCryptGenRandom or CryptGenRandom API is used for classic desktop

applications. The evaluator shall verify that the System.Random API is used for Windows Universal Applications.

21 Test 2: [conditional] For Linux platform, the evaluator shall verify that the application collects random from /dev/random or /dev/urandom.

22 Test 3: [conditional] For macOS, the evaluator shall verify that the application uses /dev/random to acquire random.

23 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).

2.1.2 Storage of Credentials (FCS_STO_EXT)

2.1.2.1 FCS_STO_EXT.1

2.1.2.1.1 TSS

24 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.

25 [Conditional] If **implement functionality to securely store** is selected, the evaluator shall verify that the TSS states how credentials are encrypted according to FCS_COP.1/DataEncryption or conditioned according to FCS_CKM.1.

2.1.2.1.2 Operational Guidance

26 No activities specified.

2.1.2.1.3 Test

27 Test 1: [conditional] If **invoke the functionality provided by the platform to securely store** is selected, the evaluator shall perform the following actions which vary per platform.

28 **For Windows:** 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.

29 **For Linux:** The evaluator shall verify that all keys are stored using Linux *keyrings*.

30 **For Mac OS X:** The evaluator shall verify that all credentials are stored within *Keychain*.

2.2 User Data Protection (FDP)

2.2.1 Network Communications (FDP_NET_EXT)

2.2.1.1 FDP_NET_EXT.1

2.2.1.1.1 TSS

31 The evaluator shall check the TSS and verify that for each connection, inbound and outbound, the protocols and ports used have been listed.

2.2.1.1.2 Operational Guidance

32 No activities specified.

2.2.1.1.3 Test

33 The evaluator shall perform the following test:

34 Test 1: [conditional] If **no network communication** or **outbound connections** is selected then 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.

35 Test 2: [conditional] If **no network communication** or **in-bound connections** is selected then the evaluator shall run the application. After the application initializes, the evaluator shall verify that any ports opened by the application have been captured in the TSS. This includes connection-based protocols (e.g. TCP, DCCP) as well as connectionless protocols (e.g. UDP).

2.3 Security Management (FMT)

2.3.1 Default Configuration (FMT_CFG)

2.3.1.1 FMT_CFG_EXT.1.1

2.3.1.1.1 TSS

36 The evaluator shall check the TSS to determine if the application requires any type of credentials and if the application installs with default credentials. If the TSF doesn't support credentials, the TSS will shall document this.

2.3.1.1.2 Operational Guidance

37 The evaluator shall check the Guidance documentation to check if any default credentials are provided and description of how they are changed at installation or before the application is operational.

2.3.1.1.3 Test

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

39 Test 1: [conditional] If **default credentials are required to be changed** during installation the evaluator shall install the application and verify that the application requires that the default credentials are changed.

Test 2: [conditional] If **default credentials are required to be changed** before application is operational 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.

40 Test 3: The evaluator shall run the application after establishing new credentials and verify that the original default credentials no longer provide access to the application.

2.3.1.2 FMT_CFG_EXT.1.2

2.3.1.2.1 TSS

41 No activities specified

2.3.1.2.2 Operational Guidance

42 No activities specified

2.3.1.2.3 Test

43 The evaluator shall install and run the application. The evaluator shall inspect the filesystem 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.

44 Test 1: [conditional] If the application is being tested on Windows, 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.

45 Test 2: [conditional] If the application is being tested on Linux, 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.

46 Test 3: [conditional] If the application is being tested on macOS, 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.

2.3.2 Specification of Management Functions (FMT_SMF)

2.3.2.1 FMT_SMF.1

2.3.2.1.1 TSS

47 No activities specified

2.3.2.1.2 Operational Guidance

48 The evaluator shall verify that every management function specified in the SFR is described in the operational guidance. If multiple management interfaces are supported, the guidance documentation must describe which interfaces may be used to perform the management functions.

2.3.2.1.3 Test

49 The evaluator shall perform the following test:

50 Test 1: For each option for transmitting sensitive data/PII, the evaluator shall configure the TOE for that option. The evaluator shall then verify that the TOE does or does not transmit sensitive information/PII, as appropriate for the configuration specified. Each function should be tested on each management interface on which the functionality is supported.

Test 2: [conditional] If **other management functions** are specified, the evaluator shall test the application's ability to provide each management function by configuring the application and testing each function specified. 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. Each function should be tested on each management interface on which the functionality is supported.

2.4 Protection of the TSF (FPT)

2.4.1 Anti-Exploitation Capabilities (FPT_AEX_EXT)

2.4.1.1 FPT_AEX_EXT.1.1

2.4.1.1.1 TSS

51 The evaluator shall ensure that the TSS describes the compiler flags used to enable ASLR when the application is compiled. If no compiler flags are

required to be set (the default behaviour satisfies the SFR), this shall be noted in the TSS.

2.4.1.1.2 Operational Guidance

52 No activities specified

2.4.1.1.3 Test

53 The evaluator shall perform either a static or dynamic analysis to determine that no memory mappings are placed at an explicit and consistent address. The method of doing so varies per platform.

54 Test 1: [conditional] If the application is being tested on Windows, the evaluator shall run the same application twice on the same system 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.

55 Test 2: [conditional] If the application is being tested on Linux, the evaluator shall run the same application twice and then compare their memory maps using pmap -x PID to ensure the two different instances share no mapping locations.

56 Test 3: [conditional] If the application is being tested on macOS, the evaluator shall run the same application twice and then compare their memory maps using vmmap PID to ensure the two different instances share no mapping locations

2.4.1.2 FPT_AEX_EXT.1.2

2.4.1.2.1 TSS

57 No activities specified

2.4.1.2.2 Operational Guidance

58 No activities specified

2.4.1.2.3 Test

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

60 Test 1: [conditional] If the application is being tested on Windows, 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.

61 Test 2: [conditional] If the application is being tested on Linux, the evaluator shall perform static analysis on the application to verify that both

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

62 Test 3: [conditional] If the application is being tested on macOS, the evaluator shall perform static analysis on the application to verify that mprotect is never invoked with the PROT_EXEC permission.

2.4.1.3 FPT_AEX_EXT.1.3

2.4.1.3.1 TSS

63 No Activities.

2.4.1.3.2 Operational Guidance

64 No Activities.

2.4.1.3.3 Test

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

66 Test 1 [conditional]: If the application is being tested on Windows which supports Windows Defender Exploit Guard (Windows 10 version 1709 or later), 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).

67 If the application is being tested on Windows which only supports the Enhanced Mitigation Experience Toolkit (EMET) (can be installed on Windows 10 version 1703 and earlier), then the evaluator shall ensure that the application can run successfully with EMET configured with the following minimum mitigations enabled; Memory Protection Check, Randomize memory allocations (Bottom-Up ASLR), Export address filtering (EAF), and Data Execution Prevention (DEP).

68 Test 2 [conditional]: If the application is being tested on Linux, the evaluator shall ensure that the application can successfully run on a system with either SELinux or AppArmor enabled and in enforce mode.

69 Test 3 [conditional]: If the application is being tested on macOS, the evaluator shall ensure that the application can successfully run without disabling any OS security functions.

2.4.1.4 FPT_AEX_EXT.1.4

2.4.1.4.1 TSS

70 No activities specified

2.4.1.4.2 Operational Guidance

71 No activities specified

2.4.1.4.3 Test

72 Test 1: 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.

73 Test 2: The evaluator shall run the application, mimicking normal usage, and note where all files are written. The evaluator shall ensure that there are no executable files stored in the same directories to which the application wrote and no data files in the application's install directory.

2.4.1.5 FPT_AEX_EXT.1.5

2.4.1.5.1 TSS

74 The evaluator shall ensure that the TSS section of the ST describes the compiler flag used to enable stack-based buffer overflow protection in the application. The following flags should be used depending on the compiler:

Compiler	*Flag*
Visual Studio	`/GS`
GCC or Xcode	`-fstack-protector-all` *(preferred)* `-fstack-protector-strong`
clang	`-fsanitize=address`

Windows 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.

2.4.1.5.2 Operational Guidance

75 No Activities.

2.4.1.5.3 Test

76 Test 1: [conditional] If the application is evaluated on Windows platform, evaluator shall run a tool, like BinScope, that can verify the correct usage of /GS

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

78 Test 2: [conditional] If the application is PE type, the evaluator will disassemble each and ensure the following sequence appears:

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

xor rcx, (...)

call (...)
```

79 Test 3: [conditional] If the application contains ELF executables, the evaluator will ensure that each contains references to the symbol **__stack_chk_fail**.

80 The test is considered passing as long as at least one instance of the above mentioned sequence/symbol is found in each compiled binary that makes up the TOE.

81 Tools such as Canary Detector (https://github.com/commoncriteria/canary-detector) may help automate these activities.

2.4.2 Trusted Update (FPT_TUD_EXT.1)

2.4.2.1 FPT_TUD_EXT.1.1

2.4.2.1.1 TSS

82 No activities specified

2.4.2.1.2 Operational Guidance

83 The evaluator must verify that the operational user guidance (AGD_OPE.1) identifies the method to query the current version of the application.

2.4.2.1.3 Test

84 The evaluator shall query the application for the current version of the software and verify that the current version matches that of the documented and installed version.

2.4.2.2 FPT_TUD_EXT.1.2

2.4.2.2.1 TSS

85 The evaluator shall verify that the TSS identifies:

- How the application installation package and updates to it are signed by an authorized source.
- The definition of an authorized source.

2.4.2.2.2 Operational Guidance

86 The evaluator shall verify that the TOE operational guidance documentation covers installation and update procedures for the application.

2.4.2.2.3 Test

87 Test 1: [conditional] If the TOE verifies the digital signature of the installation package and its updates, the evaluator shall obtain or produce illegitimate updates as defined below, and attempt to install them. The evaluator shall verify that the updates are rejected for all illegitimate updates. The evaluator shall perform this test using all of the following forms of illegitimate updates:

1) A modified version (e.g. using a hex editor) of a legitimately signed update

2) An update that has not been signed

3) An update signed with an invalid signature (e.g. by using a different key as expected for creating the signature or by manual modification of a legitimate signature)

2.5 Trusted path/channel (FTP)

2.5.1 Data in Transit (FTP_DIT)

2.5.1.1.1 FTP_DIT_EXT.1

2.5.1.1.2 TSS

88 The evaluator shall verify that the TSS identifies if sensitive data is transmitted by the application. It must also identify types of sensitive data transmitted.

2.5.1.1.3 Operational Guidance

89 No activities specified

2.5.1.1.4 Test

90 Test 1: 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. Based on the selection in the ST, the evaluator

shall verify from the packet capture that the traffic is encrypted with HTTPS, TLS, DTLS or SSH.

91 Test 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 review the packet capture and verify that no sensitive data is transmitted in the clear.

92 Test 3: [conditional] 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.

References

3 Evaluation Activities for Optional Requirements

3.1 Cryptographic Support (FCS)

3.1.1 Cryptographic Key Management (FCS_CKM)

3.1.1.1 FCS_CKM.1.1/Symmetric

3.1.1.1.1 TSS

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

94 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.

3.1.1.1.2 Operational Guidance

95 No activities specified

3.1.1.1.3 Test

96 No activities specified

3.2 Protection of the TSF (FPT)

3.2.1 Use of Supported Services and APIs (FPT_API_EXT.2)

3.2.1.1 FPT_API_EXT.2.1

3.2.1.1.1 TSS

97 The evaluator shall verify that the TSS lists the 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.

3.2.1.1.2 Operational Guidance

98 No activities specified.

3.2.1.1.3 Test

99 No activities specified.

4 Evaluation Activities for Selection-Based Requirements

4.1 Cryptographic Support (FCS)

4.1.1 Random Bit Generation (FCS_RBG_EXT.1)

4.1.1.1 FCS_RBG_EXT.1

100 Documentation shall be produced—and the evaluator shall perform the activities—in accordance with Appendix D of [SWAppcPP].

4.1.1.1.1 TSS

101 The evaluator shall examine the TSS to determine that it specifies the DRBG type, identifies the entropy source(s) seeding the DRBG, and state the assumed or calculated min-entropy supplied either separately by each source or the min-entropy contained in the combined seed value.

4.1.1.1.2 Guidance Documentation

102 The evaluator shall confirm that the guidance documentation contains appropriate instructions for configuring the RNG functionality.

4.1.1.1.3 Tests

103 The evaluator shall perform 15 trials for the RNG implementation. If the RNG is configurable, the evaluator shall perform 15 trials for each configuration.

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

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

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

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

4.1.2 Cryptographic Key Generation Services (FCS_CKM_EXT)

4.1.2.1 FCS_CKM_EXT.1.1

4.1.2.1.1 TSS

107 The evaluator shall inspect the application and its developer documentation to determine if the application needs asymmetric key generation services. If not, the evaluator shall verify the **generate no asymmetric cryptographic keys** selection is present in the ST. Otherwise, the evaluation activities shall be performed as stated in the selection-based requirements.

4.1.2.1.2 Guidance Documentation

108 No activities specified.

4.1.2.1.3 Tests

109 No activities specified.

4.1.2.2 FCS_CKM.1.1

4.1.2.2.1 TSS

110 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.

4.1.2.2.2 Guidance Documentation

111 The evaluator shall verify that the AGD guidance instructs the administrator how to configure the TOE to use the selected key generation scheme(s) and key size(s) for all cryptographic protocols defined in the Security Target.

4.1.2.2.3 Tests

112 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. Generation of long-term cryptographic keys (i.e. keys that are not ephemeral keys/session keys) might be performed automatically (e.g. during initial start-up). Testing of key generation must cover not only administrator invoked key generation but also automated key generation (if supported).

Key Generation for FIPS PUB 186-4 RSA Schemes

113 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*.

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

a) Random Primes:

- Provable primes
- Probable primes

b) 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

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

Key Generation for Elliptic Curve Cryptography (ECC)

FIPS 186-4 ECC Key Generation Test

116 For each supported NIST curve, i.e., P-256, 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 186-4 Public Key Verification (PKV) Test

117 For each supported NIST curve, i.e., P-256, 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)

118 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.

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

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

120 and two ways to generate the cryptographic group generator g:

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

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

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

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

123 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.

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

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

- g^x mod p = y

125 for each FFC parameter set and key pair.

FFC Schemes using "safe-prime" groups

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

4.1.2.3 FCS_CKM.2.1

4.1.2.3.1 TSS

127 The evaluator shall ensure that the supported key establishment schemes correspond to the key generation schemes identified in FCS_CKM.1.1. If the ST specifies more than one scheme, the evaluator shall examine the TSS to verify that it identifies the usage for each scheme (including whether the TOE acts as a sender, a recipient, or both).

128 The intent of this activity is to be able to identify the scheme being used by each service. This would mean, for example, one way to document scheme usage could be:

Scheme	SFR	Service
RSA	FCS_TLSS_EXT.1	Administration
ECDH	FCS_SSHC_EXT.1	Audit Server
Diffie-Hellman (Group 14)	FCS_SSHC_EXT.1	Backup Server
ECDH	FCS_IPSEC_EXT.1	Authentication Server

129 The information provided in the example above does not necessarily have to be included as a table but can be presented in other ways as long as the necessary data is available.

4.1.2.3.2 Guidance Documentation

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

4.1.2.3.3 Tests

Key Establishment Schemes

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

SP800-56A Key Establishment Schemes

132 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

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

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

135 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.

136 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.

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

Validity Test

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

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

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

RSA-based key establishment

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

FFC Schemes using "safe-prime" groups

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

4.1.3 Cryptographic Operation (FCS_COP)

4.1.3.1 FCS_COP.1/DataEncryption

4.1.3.1.1 TSS

143 No activities specified.

4.1.3.1.2 Guidance Documentation

144 No activities specified.

4.1.3.1.3 Tests

AES-CBC Known Answer Tests

145 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.

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

147 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.

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

149 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.

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

151 To test the decrypt functionality of AES-CBC, the evaluator shall supply the two sets of key and ciphertext value pairs described below and obtain the plaintext value that results from AES-CBC decryption of the given ciphertext using the given key and an IV of all zeros. The first set of key/ciphertext pairs shall have 128 128-bit key/ciphertext pairs, and the second set of key/ciphertext pairs shall have 256 256-bit key/ciphertext pairs. Key i in each set shall have the leftmost i bits be ones and the rightmost N-i bits be zeros, for i in [1,N]. The ciphertext value in each pair shall be the value that results in an all-zeros plaintext when decrypted with its corresponding key.

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

153 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

154 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.

155 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

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

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

157 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.

158 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 Test

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

128 bit and 256 bit keys

a) **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.

a) **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.

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

160 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.

161 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.

162 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-CTR Known Answer Tests

163 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.

164 KAT-1 To test the encrypt functionality, the evaluator shall supply a set of 10 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 128-bit all zeros key, and the other five 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 10 ciphertext values as input.

165 KAT-2 To test the encrypt functionality, the evaluator shall supply a set of 10 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 128-bit keys, and the other five 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.

166 KAT-3 To test the encrypt functionality, the evaluator shall supply the two sets of key values described below and obtain the ciphertext values that result from

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

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

AES-CTR Multi-Block Message Test

168 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.

AES-CTR Monte-Carlo Test

169 The evaluator shall test the encrypt functionality using 200 plaintext/key pairs. 100 of these shall use 128 bit keys, and 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:

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

170 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.

171 There is no need to test the decryption engine.

4.1.3.2 FCS_COP.1.1/SigGen

4.1.3.2.1 TSS

172 No activities specified.

4.1.3.2.2 Guidance Documentation

173 No activities specified.

4.1.3.2.3 Tests

ECDSA Algorithm Tests

ECDSA FIPS 186-4 Signature Generation Test

174 For each supported NIST curve (i.e., P-256, 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.

ECDSA FIPS 186-4 Signature Verification Test

175 For each supported NIST curve (i.e., P-256, 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 Tests

Signature Generation Test

176 The evaluator generates or obtains 10 messages for each modulus size/SHA combination supported by the TOE. The TOE generates and returns the corresponding signatures.

177 The evaluator shall verify the correctness of the TOE's signature using a trusted reference implementation of the signature verification algorithm and the associated public keys to verify the signatures.

Signature Verification Test

178 For each modulus size/hash algorithm selected, the evaluator generates a modulus and three associated key pairs, (*d*, *e*). Each private key *d* is used to sign six pseudorandom messages each of 1024 bits using a trusted reference implementation of the signature generation algorithm. Some of the public keys, *e*, messages, or signatures are altered so that signature verification should fail.

For both the set of original messages and the set of altered messages: the modulus, hash algorithm, public key *e* values, messages, and signatures are forwarded to the TOE, which then attempts to verify the signatures and returns the verification results.

179 The evaluator verifies that the TOE confirms correct signatures on the original messages and detects the errors introduced in the altered messages.

4.1.3.3 FCS_COP.1/Hash

4.1.3.3.1 TSS

180 The evaluator shall check that the association of the hash function with other TSF cryptographic functions (for example, the digital signature verification function) is documented in the TSS.

4.1.3.3.2 Guidance Documentation

181 The evaluator checks the AGD documents to determine that any configuration that is required to configure the required hash sizes is present.

4.1.3.3.3 Tests

182 The TSF hashing functions can be implemented in one of two modes. The first mode is the byteoriented mode. In this mode the TSF only hashes messages that are an integral number of bytes in length; i.e., the length (in bits) of the message to be hashed is divisible by 8. The second mode is the bitoriented 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 bitoriented vs. the byteoriented testmacs.

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

Short Messages Test Bitoriented Mode

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

Short Messages Test Byteoriented Mode

185 The evaluators devise an input set consisting of m/8+1 messages, where m is the block length of the hash algorithm. The length of the messages range sequentially from 0 to m/8 bytes, with each message being an integral number of bytes. The message text shall be pseudorandomly generated. The evaluators

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

Selected Long Messages Test Bitoriented Mode

186 The evaluators devise an input set consisting of m messages, where m is the block length of the hash algorithm (e.g. 512 bits for SHA-256). The length of the ith message is m + 99*i, where 1 ≤ i ≤ m. The message text shall be pseudorandomly generated. The evaluators compute the message digest for each of the messages and ensure that the correct result is produced when the messages are provided to the TSF.

Selected Long Messages Test Byteoriented Mode

187 The evaluators devise an input set consisting of m/8 messages, where m is the block length of the hash algorithm (e.g. 512 bits for SHA-256). The length of the ith message is m + 8*99*i, where 1 ≤ i ≤ m/8. The message text shall be pseudorandomly generated. The evaluators compute the message digest for each of the messages and ensure that the correct result is produced when the messages are provided to the TSF.

Pseudorandomly Generated Messages Test

188 This test is for byteoriented 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.

4.1.3.4 FCS_COP.1/KeyedHash

4.1.3.4.1 TSS

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

4.1.3.4.2 Guidance Documentation

190 No activities specified.

4.1.3.4.3 Tests

191 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 message data using a known good implementation.

4.1.4 Cryptographic Protocols (FCS_HTTPS_EXT, FCS_SSHC_EXT, FCS_SSHS_EXT, FCS_TLSC_EXT, FCS_TLSS_EXT))

4.1.4.1 FCS_HTTPS_EXT.1 HTTPS Protocol

4.1.4.1.1 TSS

192 The evaluator shall examine the TSS and determine that enough detail is provided to explain how the implementation complies with RFC 2818.

4.1.4.1.2 Guidance Documentation

193 No activities specified.

4.1.4.1.3 Tests

194 The evaluator shall perform the following tests:

c) Test 1: The evaluator shall attempt to establish each trusted path or channel that utilizes HTTPS, observe the traffic with a packet analyser, verify that the connection succeeds, and verify that the traffic is identified as TLS or HTTPS.

195 Other tests are performed in conjunction with the TLS evaluation activities.

196 If the TOE is an HTTPS client or an HTTPS server utilizing X.509 client authentication, then the certificate validity shall be tested in accordance with testing performed for FIA_X509_EXT.1.

4.1.4.2 TLS Protocol

If the TOE implements the TLS protocol, the evaluator shall test the TOE as per evaluation activities from [TLS Package]

4.1.4.3 FCS_SSH_EXT.1.1

4.1.4.3.1 TSS

197 The evaluator shall ensure that the subsequent SFR selections are consistent with the RFCs claimed in this SFR.

4.1.4.3.2 Guidance Documentation

198 No activities specified.

4.1.4.3.3 Test

199 No activities specified.

4.1.4.4 FCS_SSH_EXT.1.2

4.1.4.4.1 TSS

200 The evaluator shall check to ensure that the session authentication methods listed in the TSS are identical to those listed in this SFR component; and ensure if password-based authentication methods have been selected in the ST then these are also described; and ensure that if keyboard-interactive is selected, it describes the multifactor authentication mechanisms provided by the TOE.

4.1.4.4.2 Guidance Documentation

201 The evaluator shall check the guidance documentation to ensure the configuration options, if any, for session authentication mechanisms provided by the TOE are described.

4.1.4.4.3 Test

202 Test 1: [conditional] If the TOE is acting as SSH Server:

203 a) The evaluator shall use a suitable SSH Client to connect to the TOE, enable debug messages in the SSH Client, and examine the debug messages to determine that only the configured session authentication methods for the TOE were offered by the server.

204 b) [conditional] If the SSH server supports X509 based Client session authentication options:

1. The evaluator shall initiate an SSH session from a client where the username is associated with the X509 certificate. The evaluator shall verify the session is successfully established.

2. Next the evaluator shall use the same X509 certificate as above but include a username not associated with the certificate. The evaluator shall verify that the session does not establish.

3. Finally, the evaluator shall use the correct username (from step a above) but use a different X509 certificate which is not associated with the username. The evaluator shall verify that the session does not establish.

205 Test 2: [conditional] If the TOE is acting as SSH Client, the evaluator shall test for a successful configuration setting of each session authentication method as follows:

206 a) The evaluator shall initiate a SSH session using the authentication method configured and verify that the session is successfully established.

207 b) Next, the evaluator shall use bad authentication data (e.g. incorrectly generated certificate or incorrect password) and ensure that the connection is rejected.

208

209 Steps a-b shall be repeated for each independently configurable authentication method supported by the server.

210 Test 3: [conditional] If the TOE is acting as SSH Client, the evaluator shall verify that the connection fails upon configuration mismatch as follows:

211 a) The evaluator shall configure the Client with an authentication method not supported by the Server.

212 b) The evaluator shall verify that the connection fails.

213 If the Client supports only one authentication method, the evaluator can test this failure of connection by configuring the Server with an authentication method not supported by the Client. In order to facilitate this test, it is acceptable for the evaluator to configure an authentication method that is outside of the selections in the SFR.

4.1.4.5 FCS_SSH_EXT.1.3

4.1.4.5.1 TSS

214 The evaluator shall check that the TSS describes how “large packets” are detected and handled.

4.1.4.5.2 Guidance Documentation

215 No activities specified.

4.1.4.5.3 Test

216 Test 1:

217 The evaluator shall demonstrate that the TOE accepts the maximum allowed packet size.

218 Test2:

219 This test is performed to verify that the TOE drops packets that are larger than size specified in the component.

220 The evaluator shall establish a successful SSH connection with the peer.

221 Next the evaluator shall craft a packet that is one byte larger than the maximum size specified in this component and send it through the established SSH connection to the TOE.

222 Evaluator shall verify that the packet was dropped by the TOE.

4.1.4.6 FCS_SSH_EXT.1.4

4.1.4.6.1 TSS

223 The evaluator will check the description of the implementation of SSH in the TSS to ensure the encryption algorithms supported are specified. The evaluator will check the TSS to ensure that the encryption algorithms specified are identical to those listed for this component.

4.1.4.6.2 Guidance Documentation

224 The evaluator shall check the guidance documentation to ensure that it contains instructions to the administrator on how to ensure that only the allowed mechanisms are used in SSH connections with the TOE.

4.1.4.6.3 Test

225 The evaluator shall perform the following tests.

226 If the TOE can be both a client and a server, these tests must be performed for both roles.

227 Test 1: The evaluator must ensure that only claimed algorithms and cryptographic primitives are used to establish an SSH connection. To verify this, the evaluator shall establish an SSH connection with a remote endpoint. The evaluator shall capture the traffic exchanged between the TOE and the remote endpoint during protocol negotiation (e.g. using a packet capture tool or information provided by the endpoint, respectively). The evaluator shall verify from the captured traffic that the TOE offers only the algorithms defined in the ST for the TOE for SSH connections. The evaluator shall perform one successful negotiation of an SSH connection and verify that the negotiated algorithms were included in the advertised set. If the evaluator detects that not all algorithms defined in the ST for SSH are advertised by the TOE or the TOE advertises additional algorithms not defined in the ST for SSH, the test shall be regarded as failed.

228 The data collected from the connection above shall be used for verification of the advertised hashing and shared secret establishment algorithms in FCS_SSH_EXT.1.5 and FCS_SSH_EXT.1.6 respectively.

229 Test 2: For the connection established in Test 1, the evaluator shall terminate the connection and observe that the TOE terminates the connection.

230 Test 3: The evaluator shall configure the remote endpoint to only allow a mechanism that is not included in the ST selection. The evaluator shall attempt to connect to the TOE and observe that the attempt fails.

1.

4.1.4.7 FCS_SSH_EXT.1.5

4.1.4.7.1 TSS

231 The evaluator will check the description of the implementation of SSH in the TSS to ensure the hashing algorithms supported are specified. The evaluator will check the TSS to ensure that the hashing algorithms specified are identical to those listed for this component.

4.1.4.7.2 Guidance Documentation

232 The evaluator shall check the guidance documentation to ensure that it contains instructions to the administrator on how to ensure that only the allowed mechanisms are used in SSH connections with the TOE.

4.1.4.7.3 Test

233 Test 1: The evaluator shall use the test data collected in FCS_SSH_EXT.1.4, Test 1 to verify that appropriate mechanisms are advertised.

234 Test 2: The evaluator shall configure an SSH peer to allow only a hashing algorithm that is not included in the ST selection. The evaluator shall attempt to establish an SSH connection and observe that the connection is rejected.

4.1.4.8 FCS_SSH_EXT.1.6

4.1.4.8.1 TSS

235 The evaluator will check the description of the implementation of SSH in the TSS to ensure the shared secret establishment algorithms supported are specified. The evaluator will check the TSS to ensure that the shared secret establishment algorithms specified are identical to those listed for this component.

4.1.4.8.2 Guidance Documentation

236 The evaluator shall check the guidance documentation to ensure that it contains instructions to the administrator on how to ensure that only the allowed mechanisms are used in SSH connections with the TOE.

4.1.4.8.3 Test

237 Test 1: The evaluator shall use the test data collected in FCS_SSH_EXT.1.4, Test 1 to verify that appropriate mechanisms are advertised.

238 Test 2: The evaluator shall configure an SSH peer to allow only a key exchange method that is not included in the ST selection. The evaluator shall attempt to establish an SSH connection and observe that the connection is rejected.

4.1.4.9 FCS_SSH_EXT.1.7

4.1.4.9.1 TSS

239 The evaluator will check the description of the implementation of SSH in the TSS to ensure the KDFs supported are specified. The evaluator will check the TSS to ensure that the KDFs specified are identical to those listed for this component.

4.1.4.9.2 Guidance Documentation

240 No activities specified.

4.1.4.9.3 Test

241 This test verifies the ability of the TOE to implement the SSH key derivation function correctly. For each combination of the SHA function and encryption algorithm that is claimed by the SSH implementation on the TOE, the evaluator shall generate a shared secret, K, of length 2048 bits[1]; a hash 'H', and a 'session_id' value. Both 'H' and the 'session_id' values are of the same length as the output of the SHA function under consideration.

242 Using these inputs, the values for the following parameters are required to be calculated as per Section 7.2 of RFC 4253:

1. Initial IV client to server
2. Initial IV server to client
3. Encryption key client to server
4. Encryption key server to client
5. Integrity key client to server
6. Integrity key server to client

243 The lengths of the IVs and encryption keys are determined by the block ciphers supported by the SSH implementation on the TOE. IV lengths are determined by the cipher block size, and encryption key lengths are determined by the

cipher key lengths. To determine correctness, the evaluator shall compare the calculated parameters to those obtained by submitting the same inputs to a known good implementation.

[1]: K is of type 'mpint' (as defined in Section 5 of RFC 4251) and thus has a four byte field at the beginning (left) that indicates the length of the key in bytes.

4.1.4.10 FCS_SSH_EXT.1.8

4.1.4.10.1 TSS

244 The evaluator shall check the TSS to ensure that if the TOE enforces connection rekey or termination limits lower than the maximum values that these lower limits are identified.

245 In cases where hardware limitation will prevent reaching data transfer thresholds, the evaluator shall check the TSS to ensure it contains:
a. An argument describing this hardware-based limitation and
b. Identification of the hardware components that form the basis of such argument. **For example**, if specific Ethernet Controller or Wi-Fi radio chip is the root cause of such limitation, these subsystems shall be identified.

4.1.4.10.2 Guidance Documentation

246 The evaluator shall check the guidance documentation to ensure that if the connection rekey or termination limits are configurable, it contains instructions to the administrator on how to configure the relevant connection rekey or termination limits for the TOE.

4.1.4.10.3 Test

247 The test harness needs to be configured so that its connection rekey or termination limits are greater than the limits supported by the TOE -- it is expected that the test harness should not be initiating the connection rekey or termination.

248 Test 1. Establish an SSH connection. Wait until the identified connection rekey limit is met. Observed that a connection rekey or termination is initiated. This may require traffic to periodically be sent, or connection keep alive to be set, to ensure that the connection is not closed due to an idle timeout.

249 Test 2. Establish an SSH connection. Transmit data from the TOE until the identified connection rekey or termination limit is met. Observe that a connection rekey or termination is initiated.

250 Test 3. Establish an SSH connection. Send data to the TOE until the identified connection rekey limit or termination is met. Observe that a connection rekey or termination is initiated.

4.1.4.11 FCS_SSHS_EXT.1.1

4.1.4.11.1 TSS

251 No activities specified.

4.1.4.11.2 Guidance Documentation

252 The evaluator shall check the guidance documentation to ensure that it contains instructions to the administrator on how to ensure that only the allowed mechanisms are used in SSH connections with the TOE.

4.1.4.11.3 Test

253 The evaluator shall repeat Test 1 and Test 2 from FCS_SSH_EXT.1.4 for each of the authentication mechanisms supported by the TOE.

254 Next the evaluator shall configure the remote peer to only allow an authentication mechanism that is not included in the ST selection. The evaluator shall attempt to connect to the TOE and observe that the attempt fails.

4.1.4.12 FCS_SSHC_EXT.1.1

4.1.4.12.1 TSS

255 No activities specified.

4.1.4.12.2 Guidance Documentation

256 The evaluator shall check the guidance documentation to ensure that it contains instructions to the administrator on how to ensure that only the allowed mechanisms are used in SSH connections with the TOE.

4.1.4.12.3 Test

257 The evaluator shall perform the following tests:

258 Test 1: [conditional] If using a local database by associating each host name with its corresponding public key, the evaluator shall configure the TOE with only a single host name and corresponding public key in the local database. The evaluator shall verify that the TOE can successfully connect to the host identified by the host name.

259 Test 2: [conditional] If using a local database by associating each host name with its corresponding public key, the evaluator shall configure the TOE with only a single host name and non-corresponding public key in the local database. The evaluator shall verify that the TOE fails to connect to a host not identified by the host name.

260 Test 3: [conditional] If using a local database by associating each host name with its corresponding public key, the evaluator shall try to connect to a host not configured in the local database. The evaluator shall verify that the TOE either fails to connect to a host identified by the host name or there is a prompt provided to store the public key in the local database.

261 Test 4: [conditional] If using a list of trusted certification authorities, the evaluator shall configure the TOE with only a single trusted certification authority corresponding to the host. The evaluator shall verify that the TOE can successfully connect to the host identified by the host name.

262 Test 5: [conditional] If using a list of trusted certification authorities, the evaluator shall configure the TOE with only a single trusted certification authority that does not correspond to the host. The evaluator shall verify that the TOE fails to the host identified by the host name.

4.2 Identification and Authentication (FIA)

4.2.1 Certificate Validation (FIA_X509)

4.2.1.1 FIA_X509_EXT.1.1/Rev (X.509 Certificate Validation)

4.2.1.1.1 TSS

263 The evaluator shall ensure the TSS describes where the check of validity of the certificates takes place, and that the TSS identifies any of the rules for extendedKeyUsage fields (in FIA_X509_EXT.1.1/Rev) that are not supported by the TOE (i.e. where the ST is therefore claiming that they are trivially satisfied). It is expected that revocation checking is performed when a certificate is used in an authentication step and when performing trusted updates (if selected). It is not necessary to verify the revocation status of X.509 certificates during power-up self-tests (if the option for using X.509 certificates for self-testing is selected).

4.2.1.1.2 Guidance Documentation

264 No activities specified.

4.2.1.1.3 Test

265 The evaluator shall demonstrate that checking the validity of a certificate is performed when a certificate is used in an authentication step or when performing trusted updates (if FPT_TUD_EXT.1 is selected). It is not sufficient to verify the status of a X.509 certificate only when it is loaded onto the TOE. It is not necessary to verify the revocation status of X.509 certificates during power-up self-tests (if the option for using X.509 certificates for self-testing is selected). The evaluator shall perform the following tests for:

266 Test 1a: The evaluator shall present the TOE with a valid chain of certificates (terminating in a trusted CA certificate) as needed to validate the leaf certificate

to be used in the function, and shall use this chain to demonstrate that the function succeeds. Test 1a shall be designed in a way that the chain can be 'broken' in Test 1b by either being able to remove the trust anchor from the TOEs trust store, or by setting up the trust store in a way that at least one intermediate CA certificate needs to be provided, together with the leaf certificate from outside the TOE, to complete the chain (e.g. by storing only the root CA certificate in the trust store).

267 Test 1b: The evaluator shall then 'break' the chain used in Test 1a by either removing the trust anchor in the TOE's trust store used to terminate the chain, or by removing one of the intermediate CA certificates (provided together with the leaf certificate in Test 1a) to complete the chain. The evaluator shall show that an attempt to validate this broken chain fails.

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

269 Test 3: The evaluator shall test that the TOE can properly handle revoked certificates-–conditional on whether CRL or OCSP is selected; if both are selected, then a test shall be performed for each method. The evaluator shall test revocation of the peer certificate and revocation of the peer intermediate CA certificate i.e. the intermediate CA certificate should be revoked by the root CA. The evaluator shall ensure that a valid certificate is used, and that the validation function succeeds. The evaluator then attempts the test with a certificate that has been revoked (for each method chosen in the selection) to ensure when the certificate is no longer valid that the validation function fails. Revocation checking is only applied to certificates that are not designated as trust anchors. Therefore the revoked certificate(s) used for testing shall not be a trust anchor.

270 Test 4: If OCSP is selected, the evaluator shall configure the OCSP server or use a man-in-the-middle tool to present a certificate that does not have the OCSP signing purpose and verify that validation of the OCSP response fails. If CRL is selected, the evaluator shall configure the CA to sign a CRL with a certificate that does not have the cRLsign key usage bit set, and verify that validation of the CRL fails.

271 Test 5: The evaluator shall modify any byte in the first eight bytes of the certificate and demonstrate that the certificate fails to validate. (The certificate will fail to parse correctly.)

272 Test 6: The evaluator shall modify any byte in the last byte of the certificate and demonstrate that the certificate fails to validate. (The signature on the certificate will not validate.)

273 Test 7: The evaluator shall modify any byte in the public key of the certificate and demonstrate that the certificate fails to validate. (The hash of the certificate will not validate.)

274 Test 8: Create a valid certificate chain from root certificate designated as a trust anchor. This chain must contain at least one Elliptic Curve certificate, that has

a public key information field where the EC parameters uses an explicit format version of the Elliptic Curve parameters in the public key information field. Initiate validation of this chain by the TOE. The evaluator shall confirm the TOE treats the certificate chain as invalid.

4.2.1.2 FIA_X509_EXT.1.2/Rev (X.509 Certificate Validation)

4.2.1.2.1 TSS

275 No activities specified.

4.2.1.2.2 Guidance Documentation

276 No activities specified.

4.2.1.2.3 Tests

277 The evaluator shall perform the following tests for FIA_X509_EXT.1.2/Rev. The tests described must be performed in conjunction with the other certificate services assurance activities, including the functions in FIA_X509_EXT.2.1. The tests for the extendedKeyUsage rules are performed in conjunction with the uses that require those rules. Where the TSS identifies any of the rules for extendedKeyUsage fields (in FIA_X509_EXT.1.1) that are not supported by the TOE (i.e. where the ST is therefore claiming that they are trivially satisfied) then the associated extendedKeyUsage rule testing may be omitted.

278 For each of the following tests the evaluator shall create a chain of at least three certificates: a self-signed root CA certificate, an intermediate CA certificate and a leaf (node) certificate. The properties of the certificates in the chain are adjusted as described in each individual test below (and this modification shall be the only invalid aspect of the relevant certificate chain).

279 Test 1: The evaluator shall ensure that at least one of the CAs in the chain does not contain the basicConstraints extension. The evaluator confirms that the TOE rejects such a certificate at one (or both) of the following points: (i) as part of the validation of the leaf certificate belonging to this chain; (ii) when attempting to add a CA certificate without the basicConstraints extension to the TOE's trust store (i.e. when attempting to install the CA certificate as one which will be retrieved from the TOE itself when validating future certificate chains).

280 Test 2: The evaluator shall ensure that at least one of the CA certificates in the chain has a basicConstraints extension in which the CA flag is set to FALSE. The evaluator confirms that the TOE rejects such a certificate at one (or both) of the following points: (i) as part of the validation of the leaf certificate belonging to this chain; (ii) when attempting to add a CA certificate with the CA flag set to FALSE to the TOE's trust store (i.e. when attempting to install the CA certificate as one which will be retrieved from the TOE itself when validating future certificate chains).

281 The evaluator shall repeat these tests for each distinct use of certificates. Thus, for example, use of certificates for TLS connection is distinct from use of certificates for trusted updates so both of these uses would be tested.

4.2.1.3 FIA_X509.EXT.2 X.509 Certificate Authentication

4.2.1.3.1 TSS

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

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

4.2.1.3.2 Guidance Documentation

284 No activities specified.

4.2.1.3.3 Test

285 The evaluator shall perform the following test for each trusted channel:

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

References

5 Evaluation Activities for Objective Requirements

There are no Objective Requirements present in [SWAppCPP].

References

6 Evaluation Activities for SARs

The sections below specify EAs for the Security Assurance Requirements (SARs) included in the related cPPs. The EAs in Section 2 (*Evaluation Activities for SFRs), Section 3 (Evaluation Activities for Optional Requirements), and Section 4 (Evaluation Activities for Selection-Based Requirements)* are an interpretation of the more general CEM assurance requirements as they apply to the specific technology area of the TOE.

287 In this section, each SAR that is contained in the cPP is listed, and the EAs that are not associated with an SFR are captured here, or a reference is made to the CEM, and the evaluator is expected to perform the CEM work units.

6.1 ASE: Security Target Evaluation

288 When evaluating a Security Target, the evaluator performs the work units as presented in the CEM. In addition, the evaluator ensures the content of the TSS in the ST satisfies the EAs specified in Section 2, 3, 4, and 5 (*Evaluation Activities for SFRs)*.

6.2 ADV: Development

6.2.1 Basic Functional Specification (ADV_FSP.1)

289 The EAs for this assurance component focus on understanding the interfaces (e.g., application programing interfaces, command line interfaces, graphical user interfaces, network interfaces) described in the AGD documentation, and possibly identified in the TOE Summary Specification (TSS) in response to the SFRs. Specific evaluator actions to be performed against this documentation are identified (where relevant) for each SFR in Section 2, 3, 4 and 5. (*Evaluation Activities for SFRs)*, and in EAs for AGD, ATE and AVA SARs in other parts of Section 5.

290 The EAs presented in this section address the CEM work units ADV_FSP.1-1, ADV_FSP.1-2, ADV_FSP.1-3, and ADV_FSP.1-5.

291 The EAs are reworded for clarity and interpret the CEM work units such that they will result in more objective and repeatable actions by the evaluator. The EAs in this SD are intended to ensure the evaluators are consistently performing equivalent actions.

292 The documents to be examined for this assurance component in an evaluation are therefore the Security Target, AGD documentation, and any required supplementary information required by the cPP: no additional "functional specification" documentation is necessary to satisfy the EAs. The interfaces that need to be evaluated are also identified by reference to the EAs listed for each SFR, and are expected to be identified in the context of the Security Target,

AGD documentation, and any required supplementary information defined in the cPP rather than as a separate list specifically for the purposes of CC evaluation. The direct identification of documentation requirements and their assessment as part of the EAs for each SFR also means that the tracing required in ADV_FSP.1.2D (work units ADV_FSP.1-4, ADV_FSP.1-6 and ADV_FSP.1-7 is treated as implicit and no separate mapping information is required for this element.

CEM ADV_FSP.1 Work Units	Evaluation Activities
ADV_FSP.1-1 The evaluator ***shall examine*** the functional specification to determine that it states the purpose of each SFR-supporting and SFR-enforcing TSFI.	*2.* 6.2.1.1 Evaluation Activity: *The evaluator shall examine the interface documentation to ensure it describes the purpose and method of use for each TSFI that is identified as being security relevant.*
ADV_FSP.1-2 The evaluator ***shall examine*** the functional specification to determine that the method of use for each SFR-supporting and SFR-enforcing TSFI is given.	3. 6.2.1.2 Evaluation Activity: *The evaluator shall examine the interface documentation to ensure it describes the purpose and method of use for each TSFI that is identified as being security relevant.*
ADV_FSP.1-3 The evaluator ***shall examine*** the presentation of the TSFI to determine that it identifies all	6.2.1.3 Evaluation Activity: *The evaluator shall check the interface documentation to ensure it*

parameters associated with each SFR-enforcing and SFR supporting TSFI.	*identifies and describes the parameters for each TSFI that is identified as being security relevant.*
ADV_FSP.1-4 The evaluator ***shall examine*** the rationale provided by the developer for the implicit categorization of interfaces as SFR-non-interfering to determine that it is accurate.	Paragraph 561 from the CEM: "In the case where the developer has provided adequate documentation to perform the analysis called for by the rest of the work units for this component without explicitly identifying SFR-enforcing and SFR-supporting interfaces, this work unit should be considered satisfied." Since the rest of the ADV_FSP.1 work units will have been satisfied upon completion of the EAs, it follows that this work unit is satisfied as well.
ADV_FSP.1-5 The evaluator ***shall check*** that the tracing links the SFRs to the corresponding TSFIs.	6.2.1.4 Evaluation Activity: *The evaluator shall examine the interface documentation to develop a mapping of the interfaces to SFRs.*
ADV_FSP.1-6 The evaluator ***shall examine*** the functional specification to determine that it is a complete instantiation of the SFRs.	EAs that are associated with the SFRs in Section 2, and, if applicable, Sections 3 ,4, and 5 are performed to ensure that all the SFRs where the security functionality is externally visible (i.e., at the TSFI) are covered. Therefore, the intent of this work unit is covered.
ADV_FSP.1-7 The evaluator ***shall examine*** the functional specification to determine that it is an accurate instantiation of the SFRs.	EAs that are associated with the SFRs in Section 2, and, if applicable, Sections 3, 4, and 5, are performed to ensure that all the SFRs where the security functionality is externally visible (i.e., at the TSFI) are addressed, and that the description of the interfaces is accurate with respect to the specification captured in the SFRs. Therefore, the intent of this work unit is covered.

Table 1: Mapping of ADV_FSP.1 CEM Work Units to Evaluation Activities

6.3 AGD: Guidance Documents

293 It is not necessary for a TOE to provide separate documentation to meet the individual requirements of AGD_OPE and AGD_PRE. Although the EAs in this section are described under the traditionally separate AGD families, the mapping between the documentation provided by the developer and the AGD_OPE and AGD_PRE requirements may be many-to-many, as long as all requirements are met in documentation that is delivered to administrators and users (as appropriate) as part of the TOE.

6.3.1 Operational User Guidance (AGD_OPE.1)

294 The evaluator performs the CEM work units associated with the AGD_OPE.1 SAR. Specific requirements and EAs on the guidance documentation are identified (where relevant) in the individual EAs for each SFR.

295 In addition, the evaluator performs the EAs specified below.

6.3.1.1 Evaluation Activity

296 The evaluator shall ensure the Operational guidance documentation is distributed to administrators and users (as appropriate) as part of the TOE, so that there is a reasonable guarantee that administrators and users are aware of the existence and role of the documentation in establishing and maintaining the evaluated configuration.

6.3.1.2 Evaluation Activity

297 The evaluator shall ensure that the Operational guidance is provided for every Operational Environment that the product supports as claimed in the Security Target and shall adequately address all platforms claimed for the TOE in the Security Target.

6.3.1.3 Evaluation Activity

298 The evaluator shall ensure that the Operational guidance contains instructions for configuring any cryptographic engine associated with the evaluated configuration of the TOE. It shall provide a warning to the administrator and/or user that use of other cryptographic engines was not evaluated nor tested during the CC evaluation of the TOE.

6.3.1.4 Evaluation Activity

299 The evaluator shall ensure the Operational guidance makes it clear to an administrator which security functionality and interfaces have been assessed and tested by the EAs.

6.3.2 Preparative Procedures (AGD_PRE.1)

300 The evaluator performs the CEM work units associated with the AGD_PRE.1 SAR. Specific requirements and EAs on the preparative documentation are

identified (and where relevant are captured in the Guidance Documentation portions of the EAs) in the individual EAs for each SFR.

301 Preparative procedures are distributed to administrators and users (as appropriate) as part of the TOE, so that there is a reasonable guarantee that administrators and users are aware of the existence and role of the documentation in establishing and maintaining the evaluated configuration.

302 In addition, the evaluator performs the EAs specified below.

6.3.2.1 Evaluation Activity

303 The evaluator shall examine the Preparative procedures to ensure they include a description of how the administrator verifies that the operational environment can fulfil its role to support the security functionality (including the requirements of the Security Objectives for the Operational Environment specified in the Security Target).

304 The documentation should be in an informal style and should be written with sufficient detail and explanation that they can be understood and used by the target audience (which will typically include IT staff who have general IT experience but not necessarily experience with the TOE product itself).

6.3.2.2 Evaluation Activity

305 The evaluator shall examine the Preparative procedures to ensure they are provided for every Operational Environment that the product supports as claimed in the Security Target and shall adequately address all platforms claimed for the TOE in the Security Target.

6.3.2.3 Evaluation Activity

306 The evaluator shall examine the preparative procedures to ensure they include instructions to successfully install the TSF in each Operational Environment.

6.3.2.4 Evaluation Activity

307 The evaluator shall examine the preparative procedures to ensure they include instructions to manage the security of the TSF as a product and as a component of the larger operational environment.

6.4 ALC: Life-cycle Support

6.4.1 Labelling of the TOE (ALC_CMC.1)

308 When evaluating that the TOE has been provided and is labelled with a unique reference, the evaluator performs the work units as presented in the CEM.

6.4.2 TOE CM coverage (ALC_CMS.1)

309 When evaluating the developer's coverage of the TOE in their CM system, the evaluator performs the work units as presented in the CEM.

6.4.3 Systematic Flaw Remediation (ALC_FLR.3)

310 It is not uncommon for software applications needing to be updated due security flaws. Therefore, the response to potential security flaws must be clearly established, and comprehensive. There must be a means of providing information and solutions to users in a timely manner, using automated means. ALC_FLR.3 has been mandated to meet these requirements.

311 The following table indicates, for each work unit in ALC_FLR.3, whether the [CEM] work unit is to be performed as written, or if it has been clarified by an Evaluation Activity. If clarification has been provided, a reference to this clarification is provided in the table.

CEM ALC_FLR.3 Work Units	Evaluation Activities
ALC_FLR.3-1 The evaluator s**hall examine** the flaw remediation procedures documentation to determine that it describes the procedures used to track all reported security flaws in each release of the TOE.	The evaluator shall perform the [CEM] activity as specified.
ALC_FLR.3-2 The evaluator **shall examine** the flaw remediation procedures to determine that the application of these procedures would produce a description of each security flaw in terms of its nature and effects.	The evaluator shall perform the [CEM] activity as specified.
ALC_FLR.3-3 The evaluator **shall examine** the flaw remediation	The evaluator shall perform the [CEM] activity as specified.

procedures to determine that the application of these procedures would identify the status of finding a correction to each security flaw.	
ALC_FLR.3-4 The evaluator **shall check** the flaw remediation procedures to determine that the application of these procedures would identify the corrective action for each security flaw.	The evaluator shall perform the [CEM] activity as specified.
ALC_FLR.3-5 The evaluator **shall examine** the flaw remediation procedures documentation to determine that it describes a means of providing the TOE users with the necessary information on each security flaw.	The evaluator shall perform the [CEM] activity as specified.
ALC_FLR.3-6 The evaluator **shall examine** the flaw remediation procedures to determine that the application of these procedures would result in a means for the developer to receive from TOE user reports of suspected security flaws or requests for corrections to such flaws.	The evaluator shall perform the [CEM] activity as specified.

ALC_FLR.3-7 The evaluator **shall examine** the flaw remediation procedures to determine that the application of these procedures would result in a timely means of providing the registered TOE users who might be affected with reports about, and associated corrections to, each security flaw.	The evaluator shall perform the [CEM] activity as specified. The evaluator must ensure that the vendor has a defined set of timeframes for response to vulnerabilities. The evaluator must ensure that the vendor has rationale for those timeframes.
ALC_FLR.3-8 The evaluator **shall examine** the flaw remediation procedures to determine that the application of these procedures would result in automatic distribution of the reports and associated corrections to the registered TOE users who might be affected.	The evaluator shall perform the [CEM] activity as specified.
ALC_FLR.3-9 The evaluator **shall examine** the flaw remediation procedures to determine that the application of these procedures would help to ensure that every reported flaw is corrected.	The evaluator shall perform the [CEM] activity as specified.
ALC_FLR.3-10 The evaluator **shall examine** the flaw remediation procedures to determine that the application of these	The evaluator shall perform the [CEM] activity as specified.

procedures would help to ensure that the TOE users are issued remediation procedures for each security flaw.	
ALC_FLR.3-11 The evaluator **shall examine** the flaw remediation procedures to determine that the application of these procedures would result in safeguards that the potential correction contains no adverse effects.	The evaluator shall perform the [CEM] activity as specified.
ALC_FLR.3-12 The evaluator **shall examine** the flaw remediation guidance to determine that the application of these procedures would result in a means for the TOE user to provide reports of suspected security flaws or requests for corrections to such flaws.	The evaluator shall perform the [CEM] activity as specified.
ALC_FLR.3-13 The evaluator **shall examine** the flaw remediation guidance to determine that it describes a means of enabling the TOE users to register with the developer.	The evaluator shall perform the [CEM] activity as specified.
ALC_FLR.3-14 The evaluator **shall examine** the flaw remediation guidance to determine that it identifies specific	The evaluator shall perform the [CEM] activity as specified.

points of contact for user reports and enquiries about security issues involving the TOE.	

Table 2: Mapping of ALC_FLR.3 [CEM] Work Units to Evaluation Activities

6.5 ATE: Tests

6.5.1 Independent Testing – Conformance (ATE_IND.1)

312 The focus of the testing is to confirm that the requirements specified in the SFRs are being met. Additionally, testing is performed to confirm the functionality described in the TSS, as well as the dependencies on the Operational guidance documentation is accurate.

313 The evaluator performs the CEM work units associated with the ATE_IND.1 SAR. Specific testing requirements and EAs are captured for each SFR in Sections 2, 3, 4 and 5: *Evaluation Activities for SFRs*.

6.6 AVA: Vulnerability Assessment

6.6.1 Vulnerability Survey (AVA_VAN.1)

314 While vulnerability analysis is inherently a subjective activity, a minimum level of analysis can be defined and some measure of objectivity and repeatability (or at least comparability) can be imposed on the vulnerability analysis process. In order to achieve such objectivity and repeatability it is important that the evaluator follows a set of well-defined activities, and documents their findings so others can follow their arguments and come to the same conclusions as the evaluator. While this does not guarantee that different evaluation facilities will identify exactly the same type of vulnerabilities or come to exactly the same conclusions, the approach defines the minimum level of analysis and the scope of that analysis, and provides Certification Bodies a measure of assurance that the minimum level of analysis is being performed by the evaluation facilities.

315 In order to meet these goals some refinement of the AVA_VAN.1 CEM work units is needed. The following table indicates, for each work unit in AVA_VAN.1, whether the CEM work unit is to be performed as written, or if it has been clarified by an Evaluation Activity. If clarification has been provided, a reference to this clarification is provided in the table.

CEM AVA_VAN.1 Work Units	Evaluation Activities
AVA_VAN.1-1 The evaluator ***shall examine*** the TOE to determine that the test configuration is consistent with the configuration under evaluation as specified in the ST.	The evaluator shall perform the CEM activity as specified. The calibration of test resources specified in paragraph 1418 of the CEM applies to the tools listed in Section A.1.4.

AVA_VAN.1-2 The evaluator ***shall examine*** the TOE to determine that it has been installed properly and is in a known state	The evaluator shall perform the CEM activity as specified.
AVA_VAN.1-3 The evaluator ***shall examine*** sources of information publicly available to identify potential vulnerabilities in the TOE.	Replace CEM work unit with activities outlined in Section A.1.
AVA_VAN.1-4 The evaluator ***shall record*** in the ETR the identified potential vulnerabilities that are candidates for testing and applicable to the TOE in its operational environment.	Replace the CEM work unit with the analysis activities on the list of potential vulnerabilities in Section A.1, and documentation as specified in Section A.3.
AVA_VAN.1-5 The evaluator ***shall devise*** penetration tests, based on the independent search for potential vulnerabilities.	Replace the CEM work unit with the activities specified in Section A.2.
AVA_VAN.1-6 The evaluator ***shall produce*** penetration test documentation for the tests based on the list of potential vulnerabilities in sufficient detail to enable the tests to be repeatable. The test documentation shall include: a) identification of the potential vulnerability the TOE is being tested for; b) instructions to connect and setup all required test equipment as required to conduct the penetration test; c) instructions to establish all penetration test prerequisite initial conditions; d) instructions to stimulate the TSF; e) instructions for observing the behaviour of the TSF; f) descriptions of all expected results and the necessary analysis to be performed on the observed behaviour for comparison against expected results; g) instructions to conclude the test and establish the necessary post-test state for the TOE.	The CEM work unit is captured in Section A.3; there are no substantive differences.
AVA_VAN.1-7 The evaluator ***shall conduct*** penetration testing.	The evaluator shall perform the CEM activity as specified. See Section A.2, paragraph 339 for

	guidance related to attack potential for confirmed flaws.
AVA_VAN.1-8 The evaluator ***shall record*** the actual results of the penetration tests.	The evaluator shall perform the CEM activity as specified.
AVA_VAN.1-9 The evaluator ***shall report*** in the ETR the evaluator penetration testing effort, outlining the testing approach, configuration, depth and results.	Replace the CEM work unit with the reporting called for in Section A.3.
AVA_VAN.1-10 The evaluator ***shall examine*** the results of all penetration testing to determine that the TOE, in its operational environment, is resistant to an attacker possessing a Basic attack potential.	This work unit is not applicable for Type 1 and Type 2 flaws (as defined in Section A.1), as inclusion in this Supporting Document by the iTC makes any confirmed vulnerabilities stemming from these flaws subject to an attacker possessing a Basic attack potential. This work unit is replaced for Type 3 and Type 4 flaws by the activities defined in Section A.2, paragraph 339.
AVA_VAN.1-11 The evaluator ***shall report*** in the ETR all exploitable vulnerabilities and residual vulnerabilities, detailing for each: a) its source (e.g. CEM activity being undertaken when it was conceived, known to the evaluator, read in a publication); b) the SFR(s) not met; c) a description; d) whether it is exploitable in its operational environment or not (i.e. exploitable or residual). e) the amount of time, level of expertise, level of knowledge of the TOE, level of opportunity and the equipment required to perform the identified vulnerabilities, and the corresponding values using the tables 3 and 4 of Annex B.4.	Replace the CEM work unit with the reporting called for in Section A.3.

Table 3: Mapping of AVA_VAN.1 CEM Work Units to Evaluation Activities

316 Because of the level of detail required for the evaluation activities, the bulk of the instructions are contained in Appendix A, while an "outline" of the assurance activity is provided below.

6.6.1.1 Evaluation Activity (Documentation):

317 In addition to the activities specified by the CEM in accordance with Table 2, the evaluator shall perform the following activities.

318 The evaluator shall examine the documentation outlined below provided by the developer to confirm that it contains all required information. This documentation is in addition to the documentation already required to be supplied in response to the EAs listed previously.

319 The developer shall provide documentation identifying the list of software and hardware components that compose the TOE. Hardware components should identify at a minimum the processors used by the TOE. Software components include applications, the operating system and other major components that are independently identifiable and reusable (outside the TOE) such as a web server and protocol or cryptographic libraries. This additional documentation is merely a list of the name and version number of the components, and will be used by the evaluators in formulating hypotheses during their analysis.

6.6.1.2 Evaluation Activity

320 The evaluator formulates hypotheses in accordance with process defined in Appendix A.1. The evaluator documents the flaw hypotheses generated for the TOE in the report in accordance with the guidelines in Appendix A.3. The evaluator shall perform vulnerability analysis in accordance with Appendix A.2. The results of the analysis shall be documented in the report according to Appendix A.3.

7 Required Supplementary Information

321 This Supporting Document refers in various places to the possibility that 'required supplementary information' may need to be supplied as part of the deliverables for an evaluation. This term is intended to describe information that is not necessarily included in the Security Target or operational guidance, and that may not necessarily be public. Examples of such information could be entropy analysis, or description of a cryptographic key management architecture used in (or in support of) the TOE. The requirement for any such supplementary information will be identified in the relevant cPP.

322 The cPPs associated with this SD require an entropy analysis as described in [SWAppCPP, Appendix D].

8 References

Common Criteria[1]

[CC1] Common Criteria for Information Technology Security Evaluation, Part 1: Introduction and General Model, CCMB-2017-04-001, Version 3.1 Revision 5, April 2017.

[CC2] Common Criteria for Information Technology Security Evaluation, Part 2: Security Functional Components, CCMB-2017-04-002, Version 3.1 Revision 5, April 2017.

[CC3] Common Criteria for Information Technology Security Evaluation, Part 3: Security Assurance Components, CCMB-2017-04-003, Version 3.1 Revision 5, April 2017.

[CEM] Common Methodology for Information Technology Security Evaluation, Evaluation Methodology, CCMB-2017-04-004, Version 3.1 Revision 5, April 2017.

[CCADD] CC and CEM Addenda: Exact Conformance, Selection-Based SFRs, Optional SFRs CCDB-2017-05-xxx, Version 0.5, May 2017

Other Documents

[SWAppcPP] collaborative Protection Profile for Application Software, Version TBD

[TLS Package] Functional Package for Transport Layer Security (TLS) v1.1

[SHAVS] The Secure Hash Algorithm Validation System (SHAVS)

[1] For details see http://www.commoncriteriaportal.org/

Appendixes

A. Vulnerability Analysis

A.1 Sources of vulnerability information

323 CEM Work Unit AVA_VAN.1-3 has been supplemented in this Supporting Document to provide a better-defined set of flaws to investigate and procedures to follow based on this particular technology. Terminology used is based on the flaw hypothesis methodology, where the evaluation team hypothesizes flaws and then either proves or disproves those flaws (a flaw is equivalent to a "potential vulnerability" as used in the CEM). Flaws are categorized into four "types" depending on how they are formulated:

1. A list of flaw hypotheses applicable to the technology described by the cPP (in this case, a software application) derived from public sources as documented in Section A.1.1 – this fixed set has been agreed by the iTC. Additionally, this will be supplemented with entries for a set of public sources (as indicated below) that are directly applicable to the TOE or its identified components (as defined by the process in Section A.1.1 below); this is to ensure that the evaluators include in their assessment applicable entries that have been discovered since the cPP was published;
2. A list of flaw hypotheses listed in this document that are derived from lessons learned specific to that technology and other iTC input (that might be derived from other open sources and vulnerability databases, for example) as documented in Section A.1.2;
3. A list of flaw hypotheses derived from information available to the evaluators; this includes the baseline evidence provided by the developer described in this Supporting Document (documentation associated with EAs, documentation described in Section 6.6.1.2, documentation described in Section 6), as well as other information (public and/or based on evaluator experience) as documented in Section A.1.3; and
4. (If applicable) A list of flaw hypotheses that are generated through the use of TC-defined tools (e.g., nmap, fuzz testers) and their application as specified in section A.1.4.

A.1.1 Type 1 Hypotheses—Public-Vulnerability-based

324 The list of public sources of vulnerability information selected by the iTC is given in Section A.4.

325 The evaluators shall perform a search on the sources listed in Section A.4 to determine a list of potential flaw hypotheses that are more recent that the publication date of the cPP, and those that are specific to the TOE and its components as specified by the additional documentation mentioned above. Any duplicates – either in a specific entry, or in the flaw hypothesis that is generated from an entry from the same or a different source – can be noted and removed from consideration by the evaluation team.

326 The search criteria to be used when searching the sources published after the publication date of the cPP shall include:

- Any protocols not listed above supported (through an SFR) by the TOE (these will include at least one of the remote management protocols (IPsec, TLS, SSH))
- The TOE name (including appropriate model information as appropriate)

327 As part of type 1 flaw hypothesis generation for the specific components of the TOE, the evaluator shall also search the component manufacturer's websites to determine if flaw hypotheses can be generated on this basis (for instance, if security patches have been released for the version of the component being evaluated, the subject of those patches may form the basis for a flaw hypothesis).

A.1.2 Type 2 Hypotheses—iTC-Sourced

328 Section A.5 contains the list of flaw hypothesis generated by the iTC for this technology that must be considered by the evaluation team as flaw hypotheses in performing the vulnerability assessment.

329 If the evaluators discover a Type 3 or Type 4 flaw that they believe should be considered as a Type 2 flaw in future versions of this cPP, they should work with their Certification Body to determine the appropriate means of submitting the flaw for consideration by the iTC.

A.1.3 Type 3 Hypotheses—Evaluation-Team-Generated

330 Type 3 flaws are formulated by the evaluator based on information presented by the product (through on-line help, product documentation and user guides, etc.) and product behaviour during the (functional) testing activities. The evaluator is also free to formulate flaws that are based on material that is not part of the baseline evidence (e.g., information gleaned from an Internet mailing list, or reading interface documentation on interfaces not included in the set provided by the developer), although such activities have the potential to vary significantly based upon the product and evaluation facility performing the analysis.

331 If the evaluators discover a Type 3 flaw that they believe should be considered as a Type 2 flaw in future versions of this cPP, they should work with their Certification Body to determine the appropriate means of submitting the flaw for consideration by the iTC.

A.1.4 Type 4 Hypotheses—Tool-Generated

332 If the TOE implements the functionality of a TLS server (i.e. includes FCS_TLSS_EXT.1 from the TLS Package), then the evaluator shall perform the following activities to generate type 4 flaw hypotheses.

333 The evaluator shall perform the following activities to generate type 4 flaw hypotheses:

- Fuzz testing

- Examine effects of sending:
    - mutated packets carrying each ‘Type’ and ‘Code’ value that is undefined in the relevant RFC for each of ICMPv4 (RFC 792) and ICMPv6 (RFC 4443)
    - mutated packets carrying each ‘Transport Layer Protocol’ value that is undefined in the respective RFC for IPv4 (RFC 791) IPv6 (RFC 2460) should also be covered if it is supported and claimed by the TOE.

Since none of these packets will belong to an allowed session, the packets should not be processed by the TOE, and the TOE should not be adversely affected by this traffic. Any results that are unexpected (e.g., core dumps) are candidates for a flaw hypothesis.

- Mutation fuzz testing of the remaining fields in the required protocol headers. This testing requires sending mutations of well-formed packets that have both carefully chosen and random values inserted into each header field in turn (i.e. testing is to include both carefully chosen and random insertion test cases). The original well-formed packets would be accepted as part of a normal existing communication stream and may still be accepted as valid packets when subject to the carefully chosen mutations (the individual packet alone would be valid although its contents may not be valid in the context of preceding and/or following packets), but will often not be valid packets when random values are inserted into fields. The carefully chosen values should include semantically significant values that can be determined from the type of the data that the field represents, such as values indicating positive and negative integers, boundary conditions, invalid binary combinations (e.g. for flag sets with dependencies between bits), and missing start or end values. Randomly chosen values may not result in well-formed packets, but are included nonetheless to see whether they can lead to the device entering an insecure state. Any results that are unexpected (e.g., core dumps) are candidates for a flaw hypothesis.

334 The iTC has not identified a specific tool to be used in accomplishing the above flaw hypothesis generation activity, so any tool used by the evaluation team is acceptable. The evaluation team shall record in the test report the name, version, parameters, and results of all test tools used for this this activity.

335 If the evaluators discover a Type 4 flaw that they believe should be considered as a Type 2 flaw in future versions of this cPP, they should work with their Certification Body to determine the appropriate means of submitting the flaw for consideration by the iTC.

A.2 Process for Evaluator Vulnerability Analysis

336 As flaw hypotheses are generated from the activities described above, the evaluation team will disposition them; that is, attempt to prove, disprove, or determine the non-applicability of the hypotheses. This process is as follows.

337 The evaluator will refine each flaw hypothesis for the TOE and attempt to disprove it using the information provided by the developer or through penetration testing. During this process, the evaluator is free to interact directly with the developer to determine if the flaw exists, including requests to the developer for additional evidence (e.g., detailed design information, consultation with engineering staff); however, the CB should be included in these discussions. Should the developer object to the information being requested as being not compatible with the overall level of the evaluation activity/cPP and cannot provide evidence otherwise that the flaw is disproved, the evaluator prepares an appropriate set of materials as follows:

1. the source documents used in formulating the hypothesis, and why it represents a potential compromise against a specific TOE function;
2. an argument why the flaw hypothesis could not be proven or disproved by the evidence provided so far; and
3. the type of information required to investigate the flaw hypothesis further.

338 The Certification Body (CB) will then either approve or disapprove the request for additional information. If approved, the developer provides the requested evidence to disprove the flaw hypothesis (or, of course, acknowledge the flaw).

339 For each hypothesis, the evaluator will note whether the flaw hypothesis has been successfully disproved, successfully proven to have identified a flaw, or requires further investigation. It is important to have the results documented as outlined in Section A.3 below.

340 If the evaluator finds a flaw, the evaluator must report these flaws to the developer. All reported flaws must be addressed as follows:

341 If the developer confirms that the flaw exists and that it is exploitable at Basic Attack Potential, then a change is made by the developer, and the resulting resolution is agreed by the evaluator and noted as part of the evaluation report.

342 If the developer, the evaluator, and the CB agree that the flaw is exploitable only above Basic Attack Potential and does not require resolution for any other reason, then no change is made and the flaw is noted as a residual vulnerability in the CB-internal report (ETR).

343 If the developer and evaluator agree that the flaw is exploitable only above Basic Attack Potential, but it is deemed critical to fix because of technology-specific or cPP-specific aspects such as typical use cases or operational environments,

then a change is made by the developer, and the resulting resolution is agreed by the evaluator and noted as part of the evaluation report.

344 Disagreements between evaluator and vendor regarding questions of the existence of a flaw, its attack potential, or whether it should be deemed critical to fix are resolved by the CB.

345 Any testing performed by the evaluator shall be documented in the test report as outlined in Section A.3 below.

346 As indicated in Section A.3, Reporting, the public statement with respect to vulnerability analysis that is performed on TOEs conformant to the cPP is constrained to coverage of flaws associated with Types 1 and 2 (defined in Section A.1) flaw hypotheses only. The fact that the iTC generates these candidate hypotheses indicates these must be addressed.

347 For flaws of Types 3 and 4, each CB is responsible for determining what constitutes Basic Attack Potential for the purposes of determining whether a flaw is exploitable in the TOE's environment. The determination criteria shall be documented in the CB-internal report as specified in Section A.3. As this is a per-CB activity, no public claims are made with respect to the resistance of a particular TOE against flaws of Types 3 and 4; rather, the claim is that the activities outlined in this appendix were carried out, and the evaluation team and CB agreed that any residual vulnerabilities are not exploitable by an attacker with Basic Attack Potential.

A.3 Reporting

348 The evaluators shall produce two reports on the testing effort; one that is public-facing (that is, included in the non-proprietary evaluation report, which is a subset of the Evaluation Technical Report (ETR)), and the complete ETR that is delivered to the overseeing CB.

349 The public-facing report contains:

350 * The flaw identifiers returned when the procedures for searching public sources were followed according to instructions in the Supporting Document per Section A.1.1;

351 * A statement that the evaluators have examined the Type 1 flaw hypotheses specified in this Supporting Document in section A.1.1 (i.e. the flaws listed in the previous bullet) and the Type 2 flaw hypotheses specified in this Supporting Document by the iTC in Section A.1.2.

352 * A statement that the evaluation team developed Types 3 and 4 flaw hypotheses in accordance with Sections A.1.3, A.1.4, and A.2, and that no residual vulnerabilities exist that are exploitable by attackers with Basic Attack Potential as defined by the CB in accordance with the guidance in the CEM. It should be noted that this is just a statement about the "fact of" Types 3 and 4 flaw hypotheses being developed, and that no specifics about the number of flaws,

the flaws themselves, or the analysis pertaining to those flaws will be included in the public-facing report.

353 No other information is provided in the public-facing report.

354 The internal CB report contains, in addition to the information in the public-facing report:

- a list of all of the flaw hypotheses generated (cf. AVA_VAN.1-4);
- the evaluator penetration testing effort, outlining the testing approach, configuration, depth and results (cf. AVA_VAN.1-9);
- all documentation used to generate the flaw hypotheses (in identifying the documentation used in coming up with the flaw hypotheses, the evaluation team must characterize the documentation so that a reader can determine whether it is strictly required by this Supporting Document, and the nature of the documentation (design information, developer engineering notebooks, etc.));
- the evaluator shall report all exploitable vulnerabilities and residual vulnerabilities, detailing for each:
    - its source (e.g. CEM activity being undertaken when it was conceived, known to the evaluator, read in a publication);
    - the SFR(s) not met;
    - a description;
    - whether it is exploitable in its operational environment or not (i.e. exploitable or residual).
    - the amount of time, level of expertise, level of knowledge of the TOE, level of opportunity and the equipment required to perform the identified vulnerabilities (cf. AVA_VAN.1-11);
- how each flaw hypothesis was resolved (this includes whether the original flaw hypothesis was confirmed or disproved, and any analysis relating to whether a residual vulnerability is exploitable by an attacker with Basic Attack Potential) (cf. AVA_VAN.1-10); and
- in the case that actual testing was performed in the investigation (either as part of flaw hypothesis generation using tools specified by the iTC in Section A.1.4, or in proving/disproving a particular flaw) the steps followed in setting up the TOE (and any required test equipment); executing the test; post-test procedures; and the actual results (to a level of detail that allow repetition of the test, including the following:
    - identification of the potential vulnerability the TOE is being tested for;

- instructions to connect and setup all required test equipment as required to conduct the penetration test;
- instructions to establish all penetration test prerequisite initial conditions;
- instructions to stimulate the TSF;
- instructions for observing the behaviour of the TSF;
- descriptions of all expected results and the necessary analysis to be performed on the observed behaviour for comparison against expected results;
- instructions to conclude the test and establish the necessary post-test state for the TOE. (cf. AVA_VAN.1-6, AVA_VAN.1-8).

A.4 Public Vulnerability Sources

355 The following sources of public vulnerabilities are sources for the iTC to consider in both formulating the specific list of flaws to be investigated by the evaluators, as well as to reference in directing the evaluators to perform key-word searches during the evaluation of a specific TOE.

a) NIST National Vulnerabilities Database (can be used to access CVE and US-CERT databases identified below):
https://web.nvd.nist.gov/view/vuln/search

b) Common Vulnerabilities and Exposures:
http://cve.mitre.org/cve/
https://www.cvedetails.com/vulnerability-search.php

c) US-CERT:
http://www.kb.cert.org/vuls/html/search

d) Exploit / Vulnerability Search Engine:
www.exploitsearch.net

e) SecurITeam Exploit Search:
www.securiteam.com

f) Tenable Network Security
http://nessus.org/plugins/index.php?view=search

g) Tipping Point Zero Day Initiative
http://www.zerodayinitiative.com/advisories

h) Offensive Security Exploit Database:
https://www.exploit-db.com/

i) Rapid7 Vulnerability Database:
https://www.rapid7.com/db/vulnerabilities

A.5 Additional Flaw Hypotheses

356 No entries are currently defined for this list.

B. Equivalency Considerations

B.1 Introduction

357 The purpose of equivalence in cPP-based evaluations is to find a balance between evaluation rigor and commercial practicability—to ensure that evaluations meet customer expectations while recognizing that there is little to be gained from requiring that every variation in a product or platform be fully tested. If a product is found to be compliant with a cPP on one platform, then all equivalent products on equivalent platforms are also considered to be compliant with the cPP.

358 A Vendor can make a claim of equivalence if the Vendor believes that a particular instance of their Product implements cPP-specified security functionality in a way equivalent to the implementation of the same functionality on another instance of their Product on which the functionality was tested. The Product instances can differ in version number or feature level (model), or the instances may run on different platforms. Equivalency can be used to reduce the testing required across claimed evaluated configurations. It can also be used during Assurance Continuity to reduce testing needed to add more evaluated configurations to a certification.

359 These equivalency guidelines do not replace Assurance Continuity requirements or per scheme equivalency guidelines. Nor may equivalency be used to leverage evaluations with expired certifications.

360 These Equivalency Guidelines represent a shift from complete testing of all product instances to more of a risk-based approach. Rather than require that every combination of product and platform be tested, these guidelines support an approach that recognizes that products are being used in a variety of environments—and often in cloud environments over where the vendor (and sometimes the customer) have little or no control over the underlying hardware. Developers should be responsible for the security functionality of their applications on the platforms they are developed for—whether that is an operating system, a virtual machine, or a software-based execution environment such as a container. But those platforms may themselves run within other environments—virtual machines or operating systems—that completely abstract away the underlying hardware from the application. The developer should not be held accountable for security functionality that is implemented by platform layers that are abstracted away. The implication is that not all security functionality will necessarily be tested for all platform layers down to the hardware for all evaluated configurations—especially for applications developed for software-based execution environments such as containers. For these cases, the balancing of evaluation rigor and commercial practicability tips in favor of practicability.

361 Equivalency has two aspects:

1. ***Product Equivalence:*** Products may be considered equivalent if there are no differences between Product Models and Product Versions with respect to cPP-specified security functionality.
2. ***Platform Equivalence:*** Platforms may be considered equivalent if there are no significant differences in the services they provide to the Product—or in the way the platforms provide those services—with respect to cPP-specified security functionality.

362 The equivalency determination is made in accordance with these guidelines by the Certifier and Scheme using information provided by the Evaluator/Vendor.

B.2 Approach to Equivalency Analysis

363 There are two scenarios for performing equivalency analysis. One is when a product has been certified and the vendor wants to show that a later product should be considered certified due to equivalence with the earlier product. The other is when multiple product variants are going though evaluation together and the vendor would like to reduce the amount of testing that must be done. The basic rules for determining equivalence are the same in both cases. But there is one additional consideration that applies to equivalence with previously certified products. That is, the product with which equivalence is being claimed must have a valid certification in accordance with scheme rules and the Assurance Continuity process must be followed. If a product's certification has expired, then equivalence cannot be claimed with that product.

364 When performing equivalency analysis, the Evaluator/Vendor should first use the factors and guidelines for Product Model equivalence to determine the set of Product Models to be evaluated. In general, Product Models that do not differ in cPP-specified security functionality are considered equivalent for purposes of evaluation against the cPP.

365 If multiple revision levels of Product Models are to be evaluated—or to determine whether a revision of an evaluated product needs re-evaluation—the Evaluator/Vendor and Certifier should use the factors and guidelines for Product Version equivalence to analyze whether Product Versions are equivalent.

366 Having determined the set of Product Models and Versions to be evaluated, the next step is to determine the set of Platforms that the Products must be tested on.

367 Each non-equivalent Product for which compliance is claimed must be fully tested on each non-equivalent platform for which compliance is claimed. For non-equivalent Products on equivalent platforms, only the differences that

affect cPP-specified security functionality must be tested for each product.

368 **"Differences in PP-Specified Security Functionality" Defined**
If cPP-specified security functionality is implemented by the TOE, then differences in the actual implementation between versions or product models break equivalence for that feature. Likewise, if the TOE implements the functionality in one version or model and the functionality is implemented by the platform in another version or model, then equivalence is broken. If the functionality is implemented by the platform in multiple models or versions on equivalent platforms, then the functionality is considered different if the product invokes the platform differently to perform the function.

B.3 Specific Guidance for Determining Product Model Equivalence

369 Product Model equivalence attempts to determine whether different feature levels of the same product across a product line are equivalent for purposes of cPP testing. For example, if a product has a "basic" edition and an "enterprise" edition, is it necessary to test both models? Or does testing one model provide sufficient assurance that both models are compliant?

370 Product models are considered equivalent if there are no differences that affect PP-specified security functionality—as indicated in Table 4.

Factor	Same/Different	Guidance
PP-Specified Functionality	Same	If the differences between Models affect only non-cPP-specified functionality, then the Models are equivalent.
	Different	If cPP-specified security functionality is affected by the differences between Models, then the Models are not equivalent and must be tested separately. It is necessary only to test the functionality affected by the software differences. If only differences are tested, then the differences must be enumerated, and for each difference the Vendor must provide an explanation of why each difference does or does not affect cPP-specified functionality. If the Product Models are separately tested fully, then there is no need to document the differences.

Table 4: Determining Product Model Equivalence

B.4 Specific Guidance for Determining Product Version Equivalence

371 In cases of version equivalence, differences are expressed in terms of changes implemented in revisions of an evaluated Product. In general, versions are

equivalent if the changes have no effect on any security-relevant claims about the TOE or assurance evidence. Non-security-relevant changes to TOE functionality or the addition of non-security-relevant functionality does not affect equivalence.

Factor	Same/Different	Guidance
Product Models	Different	Versions of different Product Models are not equivalent unless the Models are equivalent as defined in previous section.
PP-Specified Functionality	Same	If the differences affect only non-cPP-specified functionality, then the Versions are equivalent.
	Different	If cPP-specified security functionality is affected by the differences, then the Versions are not considered equivalent and must be tested separately. It is necessary only to test the functionality affected by the changes. If only the differences are tested, then for each difference the Vendor must provide an explanation of why the difference does or does not affect cPP-specified functionality. If the Product Versions are separately tested fully, then there is no need to document the differences.

Table 5: Factors for Determining Product Version Equivalence

B.5 Specific Guidance for Determining Platform Equivalence

372 Platform equivalence is used to determine the platforms that equivalent versions of a Product must be tested on. Platform equivalence analysis done for one software application cannot be applied to another software application. Platform equivalence is not general—it is with respect to a particular application.

373 Product Equivalency analysis must already have been done and Products have been determined to be equivalent.

374 The platform can be hardware or virtual hardware, an operating system or similar entity, or a software execution environment such as a container. For purposes of determining equivalence for software applications, we address each type of platform separately. In general, platform equivalence is based on differences in the interfaces between the TOE and Platform that are relevant to the implementation of cPP-specified security functionality.

B.5.1 Platform Equivalence—Hardware/Virtual Hardware Platforms

375 If an Application runs directly on hardware without an operating system—or directly on virtualized hardware without an operating system—then platform equivalence is based on processor architecture and instruction sets. In the case of virtualized hardware, it is the virtualized processor and architecture that are presented to the application that matters—not the physical hardware.

376 Platforms with different processor architectures and instruction sets are not equivalent. This is not likely to be an issue for equivalency analysis for applications since there is likely to be a different version of the application for different hardware environments. Equivalency analysis becomes important when comparing processors with the same architecture. Processors with the same architecture that have instruction sets that are subsets or supersets of each other are not disqualified from being equivalent for purposes of an App evaluation. If the application takes the same code paths when executing cPP-specified security functionality on different processors of the same family, then the processors can be considered equivalent with respect to that application. For example, if an application follows one code path on platforms that support the AES-NI instruction and another on platforms that do not, then those two platforms are not equivalent with respect to that application functionality. But if the application follows the same code path whether or not the platform supports AES-NI, then the platforms are equivalent with respect to that functionality.

377 The platforms are equivalent with respect to the application if the platforms are equivalent with respect to all cPP-specified security functionality.

Factor	Same/Different/None	Guidance
Platform Architectures	Different	Platforms that present different processor architectures and instruction sets to the application are not equivalent.
PP-Specified Functionality	Same	For platforms with the same processor architecture, the platforms are equivalent with respect to the application if execution of all cPP-specified security functionality follows the same code path on both platforms.

Table 6: Factors for Determining Hardware/Virtual Hardware Platform Equivalence

B.5.2 Platform Equivalence—OS Platforms

378 For traditional applications that are built for and run on operating systems, platform equivalence is determined by the interfaces between the application

and the operating system that are relevant to cPP-specified security functionality. Generally, these are the processor interface, device interfaces, and OS APIs. The following factors applied in order:

Factor	Same/Different/None	Guidance
Platform Architectures	Different	Platforms that run on different processor architectures and instruction sets are not equivalent.
Platform Vendors	Different	Platforms from different vendors are not equivalent.
Platform Versions	Different	Platforms from the same vendor with different major version numbers are not equivalent.
Platform Interfaces	Different	Platforms from the same vendor and major version are not equivalent if there are differences in device interfaces and OS APIs that are relevant to the way the platform provides cPP-specified security functionality to the application.
Platform Interfaces	Same	Platforms from the same vendor and major version are equivalent if there are no differences in device interfaces and OS APIs that are relevant to the way the platform provides cPP-specified security functionality to the application, or if the Platform does not provide such functionality to the application.

Table 7: Factors for Determining OS/VS Platform Equivalence

B.5.3 Software-based Execution Environment Platform Equivalence

379 If an Application is built for and runs in a non-OS software-based execution environment, such as a Container or Java Runtime, then the below criteria must be used to determine platform equivalence. The key point is that the underlying hardware (virtual or physical) and OS is not relevant to platform equivalence. This allows applications to be tested and run on software-based execution environments on any hardware—as in cloud deployments

Factor	Same/Different/None	Guidance

Platform Type/Vendor	Different	Software-based execution environments that are substantially different or come from different vendors are not equivalent. For example, a java virtual machine is not the same as a container. A Docker container is not the same as a CoreOS container.
Platform Versions	Different	Execution environments that are otherwise equivalent are not equivalent if they have different major version numbers.
cPP-Specified Security Functionality	Same	All other things being equal, execution environments are equivalent if there is no significant difference in the interfaces through which the environments provide cPP-specified security functionality to applications.

Table 8: Factors for Software-based Execution Environment Platform Equivalence

B.5.4 Level of Specificity for Tested Configurations and Claimed Equivalent Configurations

380 In order to make equivalency determinations, the vendor and evaluator must agree on the equivalency claims. They must then provide the scheme with sufficient information about the TOE instances and platforms that were evaluated, and the TOE instances and platforms that are claimed to be equivalent.

381 The ST must describe all configurations evaluated down to processor manufacturer, model number, and microarchitecture version.

382 The information regarding claimed equivalent configurations depends on the platform that the application was developed for and runs on.

Traditional Applications

383 For applications that run with an operating system as their immediate platform, the claimed configuration must describe the platform down to the specific operating system version. If the platform is a virtualization system, then the claimed configuration must describe the platform down to the specific virtualization system version. The Vendor must describe the differences in the TOE with respect to cPP-specified security functionality and how the TOE functions differently to leverage platform differences in the tested configuration versus the claimed equivalent configuration. Relevant platform differences could include instruction sets, device interfaces, and OS APIs invoked by the TOE to implement cPP-specified security functionality.

Software-Based Execution Environments

384 For applications that run in a software-based execution environment such as a Java virtual machine or a Container, then the claimed configuration must describe the platform down to the specific version of the software execution environment. The Vendor must describe the differences in the TOE with respect to cPP-specified security functionality and how the TOE functions differently to leverage platform differences in the tested configuration versus the claimed equivalent configuration.