Supporting Document Mandatory Technical Document

Evaluation Activities for collaborative Protection Profile for Hardcopy Devices [DRAFT]

Version 1.0e-DRAFT, 06 February 2024

Foreword

This is a Supporting Document (SD), intended to complement the Common Criteria (CC) 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 SD has been developed by the Hardcopy Devices (HCD-iTC) and is designed to be used to support the evaluations of TOEs against the cPP identified in Section 1.1, “Technology Area and Scope of Supporting Document”.

Technical Editor

Hardcopy Device International Technical Community (HCD-iTC)

Revision History

Table 1. Revision history

Version	Date	Description
0.91	8 October 2021	Public Review Draft 1
0.98	24 February 2022	Public Review Draft 2
0.99	29 July 2022	Final Public Review Draft
1.0	31 October 2022	Initial release for use
1.0e-DRAFT	06 February 2024	DRAFT release

General Purpose

See Section 1.1, “Technology Area and Scope of Supporting Document”.

Field of special use

This Supporting Document applies to the evaluation of TOEs claiming conformance with the collaborative Protection Profile for Hardcopy Devices [HCDcPP].

Acknowledgements

This Supporting Document (SD) was developed by the Hardcopy Devices international Technical Community (iTC) also known as HCD-iTC with representatives from industry, Government agencies, Common Criteria Test Laboratories, and members of academia.

Table of Contents

Foreword . . . 1
Technical Editor . . . 1
Revision History . . . 1
General Purpose . . . 1
Field of special use . . . 1
Acknowledgements . . . 2
1. Introduction . . . 16
1.1. Technology Area and Scope of Supporting Document . . . 16
1.2. Structure of the Document . . . 16
2. Evaluation Activities for SFRs . . . 17
2.1. Security Audit (FAU) . . . 17
2.1.1. FAU_GEN.1 Audit data generation . . . 17
2.1.1.1. TSS . . . 17
2.1.1.2. Guidance Documentation . . . 17
2.1.1.3. Tests . . . 17
2.1.2. FAU_GEN.2 User identity association . . . 18
2.1.3. FAU_SAR.1 Audit review . . . 18
2.1.3.1. TSS . . . 18
2.1.3.2. Guidance Documentation . . . 18
2.1.3.3. Tests . . . 18
2.1.4. FAU_SAR.2 Restricted audit review . . . 18
2.1.4.1. Tests . . . 18
2.1.5. FAU_STG.1 Protected audit trail storage . . . 18
2.1.5.1. TSS . . . 18
2.1.5.2. Guidance Documentation . . . 18
2.1.5.3. Tests . . . 19
2.1.6. FAU_STG.4 Prevention of audit data loss . . . 19
2.1.6.1. TSS . . . 19
2.1.6.2. Guidance Documentation . . . 19
2.1.6.3. Tests . . . 19
2.1.7. FAU_STG_EXT.1 External Audit Trail Storage . . . 19
2.1.7.1. TSS . . . 19
2.1.7.2. Guidance Documentation . . . 19
2.1.7.3. Tests . . . 20
2.2. Cryptographic Support (FCS) . . . 20
2.2.1. FCS_CKM.1/SKG Cryptographic key generation (Symmetric Keys) . . . 20
2.2.1.1. TSS . . . 20
2.2.1.2. KMD . . . 20

2.2.1.3. Guidance Documentation . . . 21
2.2.1.4. Tests . . . 21
2.2.2. FCS_CKM.2 Cryptographic Key Establishment . . . 21
2.2.2.1. TSS . . . 21
2.2.2.2. Guidance Documentation . . . 21
2.2.2.3. Tests . . . 22
2.2.3. FCS_CKM_EXT.4 Cryptographic Key Material Destruction . . . 23
2.2.3.1. TSS . . . 23
2.2.3.2. KMD . . . 23
2.2.4. FCS_CKM.4 Cryptographic key destruction . . . 24
2.2.4.1. TSS . . . 24
2.2.4.2. KMD . . . 24
2.2.4.3. Guidance Documentation . . . 24
2.2.4.4. Tests . . . 25
2.2.5. FCS_COP.1/DataEncryption Cryptographic Operation (Data Encryption/Decryption) . . . 26
2.2.5.1. TSS . . . 26
2.2.5.2. Guidance Documentation . . . 26
2.2.5.3. Tests . . . 26
2.2.6. FCS_COP.1/SigGen Cryptographic Operation (Signature Generation and Verification) . . . 46
2.2.6.1. TSS . . . 46
2.2.6.2. Guidance Documentation . . . 46
2.2.6.3. Tests . . . 46
2.2.7. FCS_COP.1/Hash Cryptographic Operation (Hash Algorithm) . . . 48
2.2.7.1. TSS . . . 48
2.2.7.2. Guidance Documentation . . . 48
2.2.7.3. Tests . . . 48
2.2.8. FCS_RBG_EXT.1 Cryptographic Operation (Random Bit Generation) . . . 49
2.2.8.1. TSS . . . 50
2.2.8.2. Guidance Documentation . . . 50
2.2.8.3. Tests . . . 50
2.3. User Data Protection (FDP) . . . 51
2.3.1. FDP_ACC.1 Subset access control . . . 51
2.3.2. FDP_ACF.1 Security attribute based access control . . . 51
2.3.2.1. TSS . . . 51
2.3.2.2. Guidance Documentation . . . 51
2.3.2.3. Tests . . . 52
2.4. Identification and Authentication (FIA) . . . 52
2.4.1. FIA_AFL.1 Authentication failure handling . . . 52
2.4.1.1. TSS . . . 52
2.4.1.2. Guidance Documentation . . . 52
2.4.1.3. Tests . . . 52

2.4.2. FIA_ATD.1 User attribute definition . . . 52
2.4.2.1. TSS . . . 53
2.4.3. FIA_PMG_EXT.1 Password Management . . . 53
2.4.3.1. Guidance Documentation . . . 53
2.4.3.2. Tests . . . 53
2.4.4. FIA_UAU.1 Timing of authentication . . . 53
2.4.4.1. TSS . . . 53
2.4.4.2. Guidance Documentation . . . 54
2.4.4.3. Tests . . . 54
2.4.5. FIA_UAU.7 Protected authentication feedback . . . 54
2.4.5.1. TSS . . . 54
2.4.5.2. Tests . . . 54
2.4.6. FIA_UID.1 Timing of identification . . . 54
2.4.7. FIA_USB.1 User-subject binding . . . 54
2.4.7.1. TSS . . . 54
2.4.7.2. Tests . . . 55
2.5. Security Management (FMT) . . . 55
2.5.1. FMT_MOF.1 Management of security functions behavior . . . 55
2.5.1.1. TSS . . . 55
2.5.1.2. Guidance Documentation . . . 55
2.5.1.3. Tests . . . 55
2.5.2. FMT_MSA.1 Management of security attributes . . . 55
2.5.2.1. TSS . . . 55
2.5.2.2. Guidance Documentation . . . 55
2.5.2.3. Tests . . . 56
2.5.3. FMT_MSA.3 Static attribute initialization . . . 56
2.5.3.1. TSS . . . 56
2.5.3.2. Tests . . . 56
2.5.4. FMT_MTD.1 Management of TSF data . . . 56
2.5.4.1. Guidance Documentation . . . 56
2.5.4.2. Tests . . . 56
2.5.5. FMT_SMF.1 Specification of Management Functions . . . 57
2.5.5.1. TSS . . . 57
2.5.5.2. Guidance Documentation . . . 57
2.5.6. FMT_SMR.1 Security roles . . . 57
2.5.6.1. TSS . . . 57
2.5.6.2. Tests . . . 57
2.6. Protection of the TSF (FPT) . . . 57
2.6.1. FPT_SBT_EXT.1 Secure Boot . . . 57
2.6.1.1. TSS . . . 57
2.6.1.2. Guidance . . . 58

2.6.1.3. Tests . . . 58
2.6.2. FPT_SKP_EXT.1 Protection of TSF Data . . . 59
2.6.2.1. TSS . . . 59
2.6.3. FPT_STM.1 Reliable time stamps . . . 59
2.6.3.1. TSS . . . 59
2.6.3.2. Guidance Documentation . . . 59
2.6.3.3. Tests . . . 59
2.6.4. FPT_TST_EXT.1 TSF testing . . . 59
2.6.4.1. TSS . . . 59
2.6.4.2. Guidance Documentation . . . 59
2.6.4.3. Tests . . . 60
2.6.5. FPT_TUD_EXT.1 Trusted Update . . . 60
2.6.5.1. TSS . . . 60
2.6.5.2. Guidance Documentation . . . 60
2.6.5.3. Tests . . . 60
2.7. TOE Access (FTA) . . . 61
2.7.1. FTA_SSL.3 TSF-initiated termination . . . 61
2.7.1.1. TSS . . . 61
2.7.1.2. Guidance Documentation . . . 61
2.7.1.3. Tests . . . 61
2.8. Trusted Channels (FTP) . . . 61
2.8.1. FTP_ITC.1 Inter-TSF trusted channel . . . 61
2.8.1.1. TSS . . . 61
2.8.1.2. Guidance Documentation . . . 61
2.8.1.3. Tests . . . 61
2.8.2. FTP_TRP.1/Admin Trusted path (for Administrators) . . . 62
2.8.2.1. TSS . . . 62
2.8.2.2. Guidance Documentation . . . 62
2.8.2.3. Tests . . . 62
3. Evaluation Activities for Conditionally Mandatory Requirements . . . 63
3.1. Confidential Data on Nonvolatile Storage Devices . . . 63
3.1.1. FPT_KYP_EXT.1 Protection of Key and Key Material . . . 63
3.1.1.1. TSS . . . 63
3.1.1.2. KMD . . . 63
3.1.2. FCS_KYC_EXT.1 Key Chaining . . . 63
3.1.2.1. TSS . . . 63
3.1.2.2. KMD . . . 63
3.1.3. FDP_DSK_EXT.1 Protection of Data on Disk . . . 64
3.1.3.1. TSS . . . 64
3.1.3.2. Guidance Documentation . . . 64
3.1.3.3. KMD . . . 65

3.1.3.4. Tests . . . 65
3.2. PSTN Fax-Network Separation . . . 66
3.2.1. FDP_FXS_EXT.1 Fax separation . . . 66
3.2.1.1. TSS . . . 66
3.2.1.2. Guidance Documentation . . . 66
3.2.1.3. Tests . . . 66
3.3. Asymmetric Key Generation . . . 67
3.3.1. FCS_CKM.1/AKG Cryptographic Key Generation (for asymmetric keys) . . . 67
3.3.1.1. TSS . . . 67
3.3.1.2. Guidance Documentation . . . 67
3.3.1.3. Tests . . . 67
3.4. Network Communications . . . 71
3.4.1. FTP_TRP.1/NonAdmin Trusted path (for Non-administrators) . . . 71
3.4.1.1. TSS . . . 71
3.4.1.2. Guidance Documentation . . . 71
3.4.1.3. Tests . . . 71
4. Evaluation Activities for Optional Requirements . . . 72
4.1. USER.DOC Unavailability . . . 72
4.1.1. FDP_UDU_EXT Document Unavailability . . . 72
4.1.1.1. TSS . . . 72
4.1.1.2. Guidance Documentation . . . 72
4.1.1.3. Tests . . . 72
4.2. Data Wiping . . . 72
4.2.1. FPT_WIPE_EXT Data Wiping . . . 72
4.2.1.1. TSS . . . 72
4.2.1.2. Guidance Documentation . . . 73
4.2.1.3. Tests . . . 73
4.3. Protected Communications . . . 76
4.3.1. FCS_TLSC_EXT.2 TLS Client support for mutual authentication . . . 76
4.3.1.1. TSS . . . 76
4.3.1.1.1. FCS_TLSC_EXT.2.1 . . . 76
4.3.1.2. Guidance Documentation . . . 76
4.3.1.2.1. FCS_TLSC_EXT.2.1 . . . 76
4.3.1.3. Tests . . . 76
4.3.1.3.1. FCS_TLSC_EXT.2.1 . . . 76
4.3.2. FCS_TLSS_EXT.2 TLS Server support for mutual authentication . . . 76
4.3.2.1. TSS . . . 76
4.3.2.1.1. FCS_TLSS_EXT.2.1 and FCS_TLSS_EXT.2.2 . . . 76
4.3.2.1.2. FCS_TLSS_EXT.2.3 . . . 77
4.3.2.2. Guidance Documentation . . . 77
4.3.2.2.1. FCS_TLSS_EXT.2.1 and FCS_TLSS_EXT.2.2 . . . 77

4.3.2.2.2. FCS_TLSS_EXT.2.3 . . . 77
4.3.2.3. Tests . . . 77
4.3.2.3.1. FCS_TLSS_EXT.2.1 and FCS_TLSS_EXT.2.2 . . . 77
4.3.2.3.2. FCS_TLSS_EXT.2.3 . . . 79
4.3.3. FCS_DTLSC_EXT.2 DTLS Client support for mutual authentication . . . 79
4.3.3.1. TSS . . . 79
4.3.3.1.1. FCS_DTLSC_EXT.2.1 . . . 79
4.3.3.1.2. FCS_DTLSC_EXT.2.2 . . . 79
4.3.3.1.3. FCS_DTLSC_EXT.2.3 . . . 79
4.3.3.2. Guidance Documentation . . . 79
4.3.3.2.1. FCS_DTLSC_EXT.2.1 . . . 79
4.3.3.3. Tests . . . 79
4.3.3.3.1. FCS_DTLSC_EXT.2.1 . . . 79
4.3.3.3.2. FCS_DTLSC_EXT.2.2 . . . 80
4.3.3.3.3. FCS_DTLSC_EXT.2.3 . . . 80
4.3.4. FCS_DTLSS_EXT.2 DTLS Server support for mutual authentication . . . 80
4.3.4.1. TSS . . . 80
4.3.4.1.1. FCS_DTLSS_EXT.2.1 and FCS_DTLSS_EXT.2.2 . . . 80
4.3.4.1.2. FCS_DTLSS_EXT.2.3 . . . 80
4.3.4.2. Guidance Documentation . . . 80
4.3.4.2.1. FCS_DTLSS_EXT.2.1 and FCS_DTLSS_EXT.2.2 . . . 80
4.3.4.2.2. FCS_DTLSS_EXT.2.3 . . . 81
4.3.4.3. Tests . . . 81
4.3.4.3.1. FCS_DTLSS_EXT.2.1 and FCS_DTLSS_EXT.2.2 . . . 81
4.3.4.3.2. FCS_DTLSS_EXT.2.3 . . . 82
5. Evaluation Activities for Selection-Based Requirements . . . 83
5.1. Confidential Data on Nonvolatile Storage Devices . . . 83
5.1.1. FCS_COP.1/StorageEncryption Cryptographic operation (Data Encryption/Decryption) . . . 83
5.1.1.1. TSS . . . 83
5.1.1.2. Guidance Documentation . . . 83
5.1.1.3. Tests . . . 83
5.1.2. FCS_COP.1/KeyWrap Cryptographic operation (Key Wrapping) . . . 87
5.1.2.1. TSS . . . 87
5.1.2.2. Guidance Documentation . . . 87
5.1.2.3. KMD . . . 88
5.1.2.4. Tests . . . 88
5.1.3. FCS_COP.1/KeyEnc Cryptographic operation (Key Encryption) . . . 91
5.1.3.1. TSS . . . 91
5.1.3.2. KMD . . . 92
5.1.3.3. Tests . . . 92
5.1.4. FCS_COP.1/KeyTransport Cryptographic operation (Key Transport) . . . 92

5.1.4.1. TSS . . . 92
5.1.4.2. Guidance Documentation . . . 92
5.1.4.3. KMD . . . 92
5.1.4.4. Tests . . . 92
5.1.5. FCS_SMC_EXT.1 Submask Combining . . . 94
5.1.5.1. TSS . . . 94
5.1.5.2. KMD . . . 94
5.1.5.3. Tests . . . 94
5.2. Protected Communications . . . 94
5.2.1. FCS_IPSEC_EXT.1 IPsec selected . . . 94
5.2.1.1. TSS . . . 94
5.2.1.1.1. FCS_IPSEC_EXT.1.1 . . . 94
5.2.1.1.2. FCS_IPSEC_EXT.1.3 . . . 95
5.2.1.1.3. FCS_IPSEC_EXT.1.4 . . . 95
5.2.1.1.4. FCS_IPSEC_EXT.1.5 . . . 95
5.2.1.1.5. FCS_IPSEC_EXT.1.6 . . . 95
5.2.1.1.6. FCS_IPSEC_EXT.1.7 . . . 95
5.2.1.1.7. FCS_IPSEC_EXT.1.8 . . . 95
5.2.1.1.8. FCS_IPSEC_EXT.1.9 . . . 95
5.2.1.1.9. FCS_IPSEC_EXT.1.10 . . . 96
5.2.1.1.10. FCS_IPSEC_EXT.1.11 . . . 96
5.2.1.1.11. FCS_IPSEC_EXT.1.12 . . . 96
5.2.1.1.12. FCS_IPSEC_EXT.1.13 . . . 96
5.2.1.1.13. FCS_IPSEC_EXT.1.14 . . . 96
5.2.1.2. Guidance Documentation . . . 97
5.2.1.2.1. FCS_IPSEC_EXT.1.1 . . . 97
5.2.1.2.2. FCS_IPSEC_EXT.1.3 . . . 97
5.2.1.2.3. FCS_IPSEC_EXT.1.4 . . . 97
5.2.1.2.4. FCS_IPSEC_EXT.1.5 . . . 97
5.2.1.2.5. FCS_IPSEC_EXT.1.6 . . . 97
5.2.1.2.6. FCS_IPSEC_EXT.1.7 . . . 97
5.2.1.2.7. FCS_IPSEC_EXT.1.8 . . . 97
5.2.1.2.8. FCS_IPSEC_EXT.1.11 . . . 98
5.2.1.2.9. FCS_IPSEC_EXT.1.13 . . . 98
5.2.1.2.10. FCS_IPSEC_EXT.1.14 . . . 98
5.2.1.3. Tests . . . 98
5.2.1.3.1. FCS_IPSEC_EXT.1.1 . . . 98
5.2.1.3.2. FCS_IPSEC_EXT.1.2 . . . 99
5.2.1.3.3. FCS_IPSEC_EXT.1.3 . . . 99
5.2.1.3.4. FCS_IPSEC_EXT.1.4 . . . 99
5.2.1.3.5. FCS_IPSEC_EXT.1.5 . . . 99

5.2.1.3.6. FCS_IPSEC_EXT.1.6 . . . 100
5.2.1.3.7. FCS_IPSEC_EXT.1.7 . . . 100
5.2.1.3.8. FCS_IPSEC_EXT.1.8 . . . 100
5.2.1.3.9. FCS_IPSEC_EXT.1.10 . . . 101
5.2.1.3.10. FCS_IPSEC_EXT.1.11 . . . 101
5.2.1.3.11. FCS_IPSEC_EXT.1.12 . . . 101
5.2.1.3.12. FCS_IPSEC_EXT.1.13 . . . 102
5.2.1.3.13. FCS_IPSEC_EXT.1.14 . . . 102
5.2.2. FCS_TLSC_EXT.1 TLS Client Protocol without mutual authentication . . . 103
5.2.2.1. TSS . . . 103
5.2.2.1.1. FCS_TLSC_EXT.1.1 . . . 103
5.2.2.1.2. FCS_TLSC_EXT.1.2 . . . 103
5.2.2.1.3. FCS_TLSC_EXT.1.4 . . . 103
5.2.2.2. Guidance Documentation . . . 104
5.2.2.2.1. FCS_TLSC_EXT.1.1 . . . 104
5.2.2.2.2. FCS_TLSC_EXT.1.2 . . . 104
5.2.2.2.3. FCS_TLSC_EXT.1.4 . . . 104
5.2.2.3. Tests . . . 104
5.2.2.3.1. FCS_TLSC_EXT.1.1 . . . 104
5.2.2.3.2. FCS_TLSC_EXT.1.2 . . . 105
5.2.2.3.3. FCS_TLSC_EXT.1.3 . . . 107
5.2.2.3.4. FCS_TLSC_EXT.1.4 . . . 107
5.2.3. FCS_TLSS_EXT.1 TLS Server Protocol without mutual authentication . . . 108
5.2.3.1. TSS . . . 108
5.2.3.1.1. FCS_TLSS_EXT.1.1 . . . 108
5.2.3.1.2. FCS_TLSS_EXT.1.2 . . . 108
5.2.3.1.3. FCS_TLSS_EXT.1.3 . . . 108
5.2.3.1.4. FCS_TLSS_EXT.1.4 . . . 108
5.2.3.2. Guidance Documentation . . . 108
5.2.3.2.1. FCS_TLSS_EXT.1.1 . . . 108
5.2.3.2.2. FCS_TLSS_EXT.1.2 . . . 109
5.2.3.2.3. FCS_TLSS_EXT.1.3 . . . 109
5.2.3.2.4. FCS_TLSS_EXT.1.4 . . . 109
5.2.3.3. Tests . . . 109
5.2.3.3.1. FCS_TLSS_EXT.1.1 . . . 109
5.2.3.3.2. FCS_TLSS_EXT.1.2 . . . 110
5.2.3.3.3. FCS_TLSS_EXT.1.3 . . . 110
5.2.3.3.4. FCS_TLSS_EXT.1.4 . . . 111
5.2.4. FCS_DTLSC_EXT.1 DTLS Client Protocol without mutual authentication . . . 112
5.2.4.1. TSS . . . 112
5.2.4.1.1. FCS_DTLSC_EXT.1.1 . . . 112

5.2.4.1.2. FCS_DTLSC_EXT.1.2 . . . 112
5.2.4.1.3. FCS_DTLSC_EXT.1.4 . . . 113
5.2.4.2. Guidance Documentation . . . 113
5.2.4.2.1. FCS_DTLSC_EXT.1.1 . . . 113
5.2.4.2.2. FCS_DTLSC_EXT.1.2 . . . 113
5.2.4.2.3. FCS_DTLSC_EXT.1.4 . . . 113
5.2.4.3. Tests . . . 113
5.2.4.3.1. FCS_DTLSC_EXT.1.1 . . . 113
5.2.4.3.2. FCS_DTLSC_EXT.1.2 . . . 114
5.2.4.3.3. FCS_DTLSC_EXT.1.3 . . . 116
5.2.4.3.4. FCS_DTLSC_EXT.1.4 . . . 117
5.2.5. FCS_DTLSS_EXT.1 DTLS Server Protocol without mutual authentication . . . 117
5.2.5.1. TSS . . . 117
5.2.5.1.1. FCS_DTLSS_EXT.1.1 . . . 117
5.2.5.1.2. FCS_DTLSS_EXT.1.3 . . . 117
5.2.5.1.3. FCS_DTLSS_EXT.1.4 . . . 117
5.2.5.1.4. FCS_DTLSS_EXT.1.5 . . . 117
5.2.5.1.5. FCS_DTLSS_EXT.1.6 . . . 117
5.2.5.1.6. FCS_DTLSS_EXT.1.7 . . . 117
5.2.5.2. Guidance Documentation . . . 118
5.2.5.2.1. FCS_DTLSS_EXT.1.1 . . . 118
5.2.5.2.2. FCS_DTLSS_EXT.1.4 . . . 118
5.2.5.3. Tests . . . 118
5.2.5.3.1. FCS_DTLSS_EXT.1.1 . . . 118
5.2.5.3.2. FCS_DTLSS_EXT.1.3 . . . 119
5.2.5.3.3. FCS_DTLSS_EXT.1.4 . . . 119
5.2.5.3.4. FCS_DTLSS_EXT.1.5 . . . 120
5.2.5.3.5. FCS_DTLSS_EXT.1.6 . . . 120
5.2.5.3.6. FCS_DTLSS_EXT.1.7 . . . 120
5.2.6. FCS_SSHC_EXT.1 SSH Client . . . 122
5.2.6.1. TSS . . . 122
5.2.6.1.1. FCS_SSHC_EXT.1.2 . . . 122
5.2.6.1.2. FCS_SSHC_EXT.1.3 . . . 122
5.2.6.1.3. FCS_SSHC_EXT.1.4 . . . 122
5.2.6.1.4. FCS_SSHC_EXT.1.5 . . . 122
5.2.6.1.5. FCS_SSHC_EXT.1.6 . . . 123
5.2.6.1.6. FCS_SSHC_EXT.1.7 . . . 123
5.2.6.1.7. FCS_SSHC_EXT.1.8 . . . 123
5.2.6.2. Guidance Documentation . . . 123
5.2.6.2.1. FCS_SSHC_EXT.1.2 . . . 123
5.2.6.2.2. FCS_SSHC_EXT.1.4 . . . 123

5.2.6.2.3. FCS_SSHC_EXT.1.5 . . . 123
5.2.6.2.4. FCS_SSHC_EXT.1.6 . . . 123
5.2.6.2.5. FCS_SSHC_EXT.1.7 . . . 124
5.2.6.2.6. FCS_SSHC_EXT.1.8 . . . 124
5.2.6.3. Tests . . . 124
5.2.6.3.1. FCS_SSHC_EXT.1.2 . . . 124
5.2.6.3.2. FCS_SSHC_EXT.1.3 . . . 124
5.2.6.3.3. FCS_SSHC_EXT.1.4 . . . 124
5.2.6.3.4. FCS_SSHC_EXT.1.5 . . . 125
5.2.6.3.5. FCS_SSHC_EXT.1.6 . . . 125
5.2.6.3.6. FCS_SSHC_EXT.1.7 . . . 125
5.2.6.3.7. FCS_SSHC_EXT.1.8 . . . 125
5.2.6.3.8. FCS_SSHC_EXT.1.9 . . . 126
5.2.7. FCS_SSHS_EXT.1 SSH Server . . . 127
5.2.7.1. TSS . . . 127
5.2.7.1.1. FCS_SSHS_EXT.1.2 . . . 127
5.2.7.1.2. FCS_SSHS_EXT.1.3 . . . 127
5.2.7.1.3. FCS_SSHS_EXT.1.4 . . . 127
5.2.7.1.4. FCS_SSHS_EXT.1.5 . . . 127
5.2.7.1.5. FCS_SSHS_EXT.1.6 . . . 127
5.2.7.1.6. FCS_SSHS_EXT.1.7 . . . 127
5.2.7.1.7. FCS_SSHS_EXT.1.8 . . . 128
5.2.7.2. Guidance Documentation . . . 128
5.2.7.2.1. FCS_SSHS_EXT.1.4 . . . 128
5.2.7.2.2. FCS_SSHS_EXT.1.5 . . . 128
5.2.7.2.3. FCS_SSHS_EXT.1.6 . . . 128
5.2.7.2.4. FCS_SSHS_EXT.1.7 . . . 128
5.2.7.2.5. FCS_SSHS_EXT.1.8 . . . 128
5.2.7.3. Tests . . . 128
5.2.7.3.1. FCS_SSHS_EXT.1.2 . . . 128
5.2.7.3.2. FCS_SSHS_EXT.1.3 . . . 129
5.2.7.3.3. FCS_SSHS_EXT.1.4 . . . 129
5.2.7.3.4. FCS_SSHS_EXT.1.5 . . . 129
5.2.7.3.5. FCS_SSHS_EXT.1.6 . . . 129
5.2.7.3.6. FCS_SSHS_EXT.1.7 . . . 130
5.2.7.3.7. FCS_SSHS_EXT.1.8 . . . 130
5.2.8. FCS_HTTPS_EXT.1 HTTPS selected . . . 131
5.2.8.1. TSS . . . 131
5.2.8.2. Guidance Documentation . . . 131
5.2.8.3. Tests . . . 131
5.2.9. FCS_COP.1/KeyedHash Cryptographic Operation (Keyed Hash Algorithm) . . . 131

5.2.9.1. TSS . . . 131
5.2.9.2. Guidance Documentation . . . 131
5.2.9.3. Tests . . . 132
5.2.10. FIA_PSK_EXT.1 Pre-Shared Key Composition . . . 132
5.2.10.1. TSS . . . 132
5.2.10.2. Guidance Documentation . . . 132
5.2.10.3. Tests . . . 132
5.3. Passphrase-based Key Entry . . . 133
5.3.1. FCS_PCC_EXT.1 Cryptographic Password Construct and Conditioning . . . 133
5.3.1.1. TSS . . . 133
5.3.1.2. KMD . . . 133
5.3.1.3. Tests . . . 133
5.3.2. FCS_KDF_EXT.1 Cryptographic Key Derivation . . . 133
5.3.2.1. TSS . . . 133
5.3.2.2. KMD . . . 134
5.3.2.3. Tests . . . 134
5.3.3. FCS_COP.1/CMAC Cryptographic Operation (for cipher-based message authentication) 136
5.3.3.1. TSS . . . 136
5.3.3.2. Guidance Documentation . . . 136
5.3.3.3. KMD . . . 136
5.3.3.4. Tests . . . 136
5.3.4. FCS_SNI_EXT.1 Cryptographic Operation (Salt, Nonce, and Initialization Vector Generation) . . . 136
5.3.4.1. TSS . . . 136
5.4. Identification and Authentication (FIA) . . . 137
5.4.1. Authentication using X.509 certificates (FIA_X509_EXT) . . . 137
5.4.1.1. FIA_X509_EXT.1 X.509 Certificate Validation . . . 137
5.4.1.1.1. TSS . . . 137
5.4.1.1.2. Guidance Documentation . . . 137
5.4.1.1.3. Tests . . . 137
5.4.1.2. FIA_X509_EXT.2 X.509 Certificate Authentication . . . 140
5.4.1.2.1. TSS . . . 140
5.4.1.2.2. Guidance Documentation . . . 140
5.4.1.2.3. Tests . . . 140
5.4.1.3. FIA_X509_EXT.3 X509 Certificate Requests . . . 140
5.4.1.3.1. TSS . . . 140
5.4.1.3.2. Guidance Documentation . . . 141
5.4.1.3.3. Tests . . . 141
6. Evaluation Activities for SARs . . . 142
6.1. Class ASE: Security Target . . . 142
6.2. Class ADV: Development . . . 142

6.2.1. Basic Functional Specification (ADV_FSP.1) . . . 142
6.2.2. ADV_FSP.1-1 Evaluation Activity . . . 144
6.2.3. ADV_FSP.1-2 Evaluation Activity . . . 144
6.2.4. ADV_FSP.1-3 Evaluation Activity . . . 144
6.2.5. ADV_FSP.1-5 Evaluation Activity . . . 145
6.3. Class AGD: Guidance Documentation . . . 145
6.3.1. Operational User Guidance (AGD_OPE.1) . . . 145
6.3.1.1. Evaluation Activity . . . 145
6.3.1.2. Evaluation Activity . . . 145
6.3.1.3. Evaluation Activity . . . 146
6.3.1.4. Evaluation Activity . . . 146
6.3.1.5. Evaluation Activity . . . 146
6.3.2. Preparative Procedures (AGD_PRE.1) . . . 146
6.3.2.1. Evaluation Activity . . . 146
6.3.2.2. Evaluation Activity . . . 147
6.3.2.3. Evaluation Activity . . . 147
6.3.2.4. Evaluation Activity . . . 147
6.3.2.5. Evaluation Activity . . . 147
6.4. Class ALC: Life-cycle Support . . . 147
6.4.1. Labelling of the TOE (ALC_CMC.1) . . . 147
6.4.2. TOE CM coverage (ALC_CMS.1) . . . 147
6.5. Class ATE: Tests . . . 147
6.5.1. Independent Testing - Conformance (ATE_IND.1) . . . 148
6.6. Class AVA: Vulnerability Assessment . . . 148
6.6.1. Vulnerability Survey (AVA_VAN.1) . . . 148
6.6.1.1. Evaluation Activity (Documentation) . . . 152
7. Required Supplementary Information . . . 153
8. Evaluation Activity . . . 154
9. References . . . 155
Appendix A: Vulnerability Analysis . . . 156
A.1. Sources of Vulnerability Information . . . 156
A.1.1. Type 1 Hypotheses - Public-Vulnerability-based . . . 156
A.1.2. Type 2 Hypotheses - iTC-sourced . . . 157
A.1.3. Type 3 Hypotheses - Evaluation-Team-Generated . . . 157
A.1.4. Type 4 Hypotheses - Tool-Generated . . . 157
A.2. Process for Evaluator Vulnerability Analysis . . . 157
A.3. Reporting . . . 159
Appendix B: Equivalency Considerations . . . 161
B.1. Introduction . . . 161
B.2. Evaluator guidance for determining equivalence . . . 161
B.2.1. Strategy . . . 161

B.2.2. Guidance for hardcopy devices . . . 162
B.2.3. Test presentation/Truth in advertising . . . 167
Appendix C: Glossary . . . 168
Appendix D: Acronyms . . . 170

Chapter 1. Introduction

1.1. Technology Area and Scope of Supporting Document

This Supporting Document (SD) defines the Evaluation Activities (EAs) associated with the collaborative Protection Profile for Hardcopy Devices [HCDcPP].

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

- collaborative Protection Profile for Hardcopy Devices, 1.0e-DRAFT, 17 January 2024

Although EAs are defined mainly for the evaluator to follow, the definitions in this SD aim to provide a common understanding for developers, evaluators, certifiers/certification bodies, and users as to what aspects of the TOE are tested in an evaluation against the associated cPPs, and to what depth the testing is carried out. This common understanding in turn contributes to the goal of ensuring that evaluations against the cPP achieve comparable, transparent and repeatable results. In general, the definition of EAs will also help developers to prepare for evaluation by identifying specific requirements for their TOE. The specific requirements in EAs may in some cases clarify the meaning of SFRs, and may identify particular requirements for the content of Security Targets (STs) (especially the TOE Summary Specification (TSS)), Administrator Guidance Documentation (AGD), and possibly required supplementary information (e.g., for entropy analysis or cryptographic key management architecture - see Chapter 7, *Required Supplementary Information*).

1.2. Structure of the Document

EAs can be defined for both SFRs and SARs. These are defined in separate sections of this SD.

If any EA cannot be successfully completed in an evaluation then the overall verdict for the evaluation is a 'fail'.

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'. To reach a 'fail' verdict when the EAs have been successfully completed would require a specific justification from the evaluator as to why the EAs were not sufficient for that TOE.

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

Chapter 2. Evaluation Activities for SFRs

The EAs presented in this section capture the actions the evaluator performs to address technology specific aspects covering specific SARs (e.g.., ASE_TSS.1, ADV_FSP.1, AGD_OPE.1, and ATE_IND.1) - this is in addition to the CEM work units that are performed in Section 6 (Evaluation Activities for SARs).

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.

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.

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. 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. Security Audit (FAU)

2.1.1. FAU_GEN.1 Audit data generation

2.1.1.1. TSS

The evaluator shall check the TOE Summary Specification (TSS) to ensure that auditable events and its recorded information are consistent with the definition of the SFR.

2.1.1.2. Guidance Documentation

The evaluator shall check the guidance documents to ensure that auditable events and its recorded information are consistent with the definition of the SFRs.

2.1.1.3. Tests

The evaluator shall also perform the following tests:

The evaluator shall check to ensure that the audit record of each of the auditable events described in Table 3 of [HCDcPP] is appropriately generated.

The evaluator shall check a representative sample of methods for generating auditable events, if there are multiple methods.

The evaluator shall check that FIA_UAU.1 events have been generated for each mechanism, if there are several different I&A mechanisms.

2.1.2. FAU_GEN.2 User identity association

The EAs for FAU_GEN.1 address this SFR.

2.1.3. FAU_SAR.1 Audit review

2.1.3.1. TSS

The evaluator shall check to ensure that the TSS contains a description that audit records can be viewed only by an Administrator and functions to view audit records.

The evaluator shall check to ensure that the TSS contains a description of the methods of using interfaces that retrieve audit records (e.g., methods for user identification and authentication, authorization, and retrieving audit records).

2.1.3.2. Guidance Documentation

The evaluator shall check to ensure that the operational guidance appropriately describes the ways of viewing audit records and forms of viewing.

2.1.3.3. Tests

The evaluator shall also perform the following tests:

1. The evaluator shall check to ensure that the forms of audit records are provided as specified in the operational guidance by retrieving audit records in accordance with the operational guidance.
2. The evaluator shall check to ensure that no users other than an Administrator can retrieve audit records.
3. The evaluator shall check to ensure that all audit records are retrieved by the operation of retrieving audit records.

2.1.4. FAU_SAR.2 Restricted audit review

2.1.4.1. Tests

The evaluator shall include test 2 related to this function in the set of tests performed in FAU_SAR.1.

2.1.5. FAU_STG.1 Protected audit trail storage

2.1.5.1. TSS

The evaluator shall check to ensure that the TSS contains a description of the means of preventing audit records from unauthorized access (modification, deletion).

2.1.5.2. Guidance Documentation

The evaluator shall check to ensure that the TSS and operational guidance contain descriptions of the interfaces to access to audit records, and if the descriptions of the means of preventing audit

records from unauthorized access (modification, deletion) are consistent.

2.1.5.3. Tests

The evaluator shall also perform the following test:

1. The evaluator shall test that an authorized user can access the audit records.
2. The evaluator shall test that a user without authorization for the audit data cannot access the audit records.

2.1.6. FAU_STG.4 Prevention of audit data loss

2.1.6.1. TSS

The evaluator shall check to ensure that the TSS contains a description of the processing performed when the capacity of audit records becomes full, which is consistent with the definition of the SFR.

2.1.6.2. Guidance Documentation

The evaluator shall check to ensure that the operational guidance contains a description of the processing performed (such as informing the authorized users) when the capacity of audit records becomes full.

2.1.6.3. Tests

The evaluator shall also perform the following tests:

1. The evaluator generates auditable events after the capacity of audit records becomes full by generating auditable events in accordance with the operational guidance.
2. The evaluator shall check to ensure that audit records are processed in accordance with the definition of the SFR.

2.1.7. FAU_STG_EXT.1 External Audit Trail Storage

2.1.7.1. TSS

The evaluator shall examine the TSS to ensure it describes the means by which the audit data are transferred to the external audit server, and how the trusted channel is provided.

The evaluator shall examine the TSS to ensure it describes the amount of audit data that are stored locally; what happens when the local audit data store is full; and how these records are protected against unauthorized access.

2.1.7.2. Guidance Documentation

The evaluator shall also examine the operational guidance to determine that it describes the relationship between the local audit data and the audit data that are sent to the audit log server. For example, when an audit event is generated, is it simultaneously sent to the external server and the local store, or is the local store used as a buffer and “cleared” periodically by sending the data to the

audit server.

The evaluator shall also examine the operational guidance to ensure it describes how to establish the trusted channel to the audit server, as well as describe any requirements on the audit server (particular audit server protocol, version of the protocol required, etc.), as well as configuration of the TOE needed to communicate with the audit server.

2.1.7.3. Tests

Testing of the trusted channel mechanism will be performed as specified in the associated assurance activities for the particular trusted channel mechanism. The evaluator shall perform the following test for this requirement:

Test 1: The evaluator shall establish a session between the TOE and the audit server according to the configuration guidance provided. The evaluator shall then examine the traffic that passes between the audit server and the TOE during several activities of the evaluator's choice designed to generate audit data to be transferred to the audit server. The evaluator shall observe that these data are not able to be viewed in the clear during this transfer, and that they are successfully received by the audit server. The evaluator shall record the particular software (name, version) used on the audit server during testing. The evaluator shall verify that the TOE is capable of transferring audit data to an external audit server automatically without administrator intervention.

2.2. Cryptographic Support (FCS)

2.2.1. FCS_CKM.1/SKG Cryptographic key generation (Symmetric Keys)

2.2.1.1. TSS

The evaluator shall review the TSS to determine that it describes how the functionality described by FCS_RBG_EXT.1 is invoked and how the TOE obtains a symmetric key through direct generation from a random bit generator as specified in FCS_RBG_EXT.1 or by combining one or more keys and other data.

2.2.1.2. KMD

If the TOE is relying on random number generation from a third-party source, the KMD needs to describe the function call and parameters used when calling the third-party DRBG function. Also, the KMD needs to include a short description of the vendor's assumption for the amount of entropy seeding the third-party DRBG. The evaluator uses the description of the RBG functionality in FCS_RBG_EXT or the KMD 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 (FCS_COP.1/StorageEncryption).

If the TOE uses the generated key in a key chain/hierarchy, then the evaluator shall examine the KMD to confirm that it describes how the key is used as part of the key chain/hierarchy.

The KMD is described in Appendix F of [HCDcPP].

2.2.1.3. Guidance Documentation

If the TOE provides interfaces to configure the cryptographic key generation functionality to the authorized role, then the evaluator shall examine the operational user guidance to determine that it describes advice regarding effective use of selected cryptographic key length and relative cryptographic algorithms.

2.2.1.4. Tests

The evaluator shall include test cases of FCS_CKM.1/SKG to the test subset. Note that FCS_CKM.1/SKG may be not mapped to the specific interface(s) after evaluator's analysis during ADV_FSP.1.

The evaluator shall produce test documentation for test cases of FCS_CKM.1/SKG. If there is no explicit external interface(s) mapped to FCS_CKM.1/SKG, the evaluator shall employ an alternative test approach (refer to CEM, section 15.2.2.)

For each selected key size, the evaluator shall configure the symmetric key generation capability. The evaluator shall use the description of the RBG interface to verify that the TOE requests and receives an amount of RBG output greater than or equal to the requested key size.

2.2.2. FCS_CKM.2 Cryptographic Key Establishment

2.2.2.1. TSS

The evaluator shall ensure that the supported key establishment schemes correspond to the key generation schemes identified in FCS_CKM.1.1/AKG. If the ST specifies more than one scheme, the evaluator shall examine the TSS to verify that it identifies the usage for each scheme. It is sufficient to provide the scheme, SFR, and service in the TSS.

If Diffie-Hellman group 14 is selected from FCS_CKM.2.1, the TSS shall claim the TOE meets RFC 3526 Section 3.

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

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.

2.2.2.2. Guidance Documentation

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

2.2.2.3. Tests

Key Establishment Schemes

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

SP800-56A Key Establishment Schemes

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

Function Test

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

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

If the TOE does not use a KDF defined in SP800-56A, the evaluator shall obtain only the public keys and the hashed value of the shared secret.

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

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

Validity Test

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

pairs, MACTag, and any inputs used in the KDF, such as the other info and TOE id fields.

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

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

RSA-based key establishment

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_TRP.1/Admin, FTP_TRP.1/NonAdmin and FTP_ITC.1 that uses RSAES-PKCS1-v1_5.

Diffie-Hellman Group 14

The evaluator shall verify the correctness of the TSF's implementation of Diffie-Hellman group 14 by using a known good implementation for each protocol selected in FTP_TRP.1/Admin, FTP_TRP.1/NonAdmin and FTP_ITC.1 that uses Diffie-Hellman group 14. FFC Schemes using "safe-prime" groups

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

2.2.3. FCS_CKM_EXT.4 Cryptographic Key Material Destruction

2.2.3.1. TSS

The evaluator shall verify the TSS provides a high level description of what it means for keys and key material to be no longer needed and when they should be expected to be destroyed.

2.2.3.2. KMD

The evaluator shall verify the Key Management Description (KMD) includes a description of the areas where keys and key material reside and when the keys and key material are no longer needed.

The evaluator shall verify the KMD includes a key lifecycle, that includes a description where key material reside, how the key material is used, how it is determined that keys and key material are no longer needed, and how the material is destroyed once it is not needed and that the documentation in the KMD follows FCS_CKM.4 for the destruction.

2.2.4. FCS_CKM.4 Cryptographic key destruction

2.2.4.1. TSS

The evaluator shall verify the TSS provides a high level description of how keys and key material are destroyed.

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

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

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

2.2.4.2. KMD

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

The evaluator shall check to ensure the KMD lists each type of key that is stored in nonvolatile memory, and identifies the memory type (volatile or nonvolatile) where key material is stored.

The KMD identifies and describes the interface(s) that is used to service commands to read/write memory. The evaluator examines the interface description for each different media type to ensure that the interface supports the selection(s) made by the ST Author.

2.2.4.3. Guidance Documentation

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

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

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

when not actively engaged in other tasks.

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

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

2.2.4.4. Tests

For these tests the evaluator shall utilize appropriate development environment (e.g., a Virtual Machine) and development tools (debuggers, simulators, etc.) to test that keys are cleared, including all copies of the key that may have been created internally by the TOE during normal cryptographic processing with that key.

Test 1 [conditional]: Applied to each key held as in volatile memory and subject to destruction by overwrite by the TOE (whether or not the value is subsequently encrypted for storage in volatile or nonvolatile memory). In the case where the only selection made for the key destruction method was removal of power, destruction of reference to the key directly followed by a request for garbage collection, or memory management, then this test is unnecessary. The evaluator shall:

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

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

Test 2: Applied to each key held in nonvolatile memory and subject to destruction by the TOE, except for replacing a key using the selection [a new value of a key of the same size]. The evaluator shall use special tools (as needed), provided by the TOE developer if necessary, to ensure the tests function as intended.

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

succeeds if step 3 fails.

Test 3: Applied to each key held in nonvolatile memory and subject to destruction by overwrite by the TOE. The evaluator shall use special tools (as needed), provided by the TOE developer if necessary, to view the key storage location:

1. Record the value of the key in the TOE subject to clearing.
2. Cause the TOE to perform a normal cryptographic processing with the key from Step #1.
3. Cause the TOE to clear the key.
4. Search the nonvolatile memory the key was stored in for instances of the known key value from Step #1. If a copy is found, then the test fails.

Test 4: Applied to each key held as nonvolatile memory and subject to destruction by overwrite by the TOE. The evaluator shall use special tools (as needed), provided by the TOE developer if necessary, to view the key storage location:

1. Record the storage location of the key in the TOE subject to clearing.
2. Cause the TOE to perform a normal cryptographic processing with the key from Step #1.
3. Cause the TOE to clear the key.
4. Search the storage location in Step #1 of nonvolatile memory to ensure the appropriate pattern is utilized.

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

2.2.5. FCS_COP.1/DataEncryption Cryptographic Operation (Data Encryption/Decryption)

2.2.5.1. TSS

The evaluator shall examine the TSS to ensure it identifies the key size(s) and mode(s) supported by the TOE for data encryption/decryption.

2.2.5.2. Guidance Documentation

The evaluator shall verify that the AGD guidance instructs the administrator how to configure the TOE to use the selected mode(s) and key size(s) defined in the Security Target supported by the TOE for data encryption/decryption.

2.2.5.3. Tests

Note: The tests detailed below are not required to be performed for cryptographic functions implemented in the Root of Trust for Secure Boot (FPT_SBT_EXT.1).

AES-CBC Known Answer Tests

There are four Known Answer Tests (KATs), described below. In all KATs, the plaintext, ciphertext, and IV values shall be 128-bit blocks. The results from each test may either be obtained by the

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

KAT-1 (GFSBox): To test the encrypt functionality of AES-CBC, the evaluator shall supply a set of five different plaintext values for each selected key size 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.

To test the decrypt functionality of AES-CBC, the evaluator shall supply a set of five different ciphertext values for each selected key size and obtain the plaintext value that results from AES-CBC decryption of the given ciphertext using a key value of all zeros and an IV of all zeros.

KAT-2 (KeySBox): To test the encrypt functionality of AES-CBC, the evaluator shall supply a set of five different key values for each selected key size 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.

To test the decrypt functionality of AES-CBC, the evaluator shall supply a set of five different key values for each selected key size and obtain the plaintext that results from AES-CBC decryption of an all-zeros ciphertext using the given key and an IV of all zeros.

KAT-3 (Variable Key): To test the encrypt functionality of AES-CBC, the evaluator shall supply a set of keys for each selected key size (as described below) and obtain the ciphertext value that results from AES encryption of an all-zeros plaintext using each key and an IV of all zeros.

Key i in each set shall have the leftmost i bits set to ones and the remaining bits to zeros, for values of i from 1 to the key size. The keys and corresponding ciphertext are listed in AESAVS, Appendix E.

To test the decrypt functionality of AES-CBC, the evaluator shall use the same keys as above to decrypt the ciphertext results from above. Each decryption should result in an all-zeros plaintext.

KAT-4 (Variable Text): To test the encrypt functionality of AES-CBC, for each selected key size, the evaluator shall supply a set of 128-bit plaintext values (as described below) and obtain the ciphertext values that result from AES-CBC encryption of each plaintext value using a key of each size and IV consisting of all zeros.

Plaintext value i shall have the leftmost i bits set to ones and the remaining bits set to zeros, for values of i from 1 to 128. The plaintext values are listed in AESAVS, Appendix D.

To test the decrypt functionality of AES-CBC, for each selected key size, use the plaintext values from above as ciphertext input, and AES-CBC decrypt each ciphertext value using key of each size consisting of all zeros and an IV of all zeros.

AES-CBC Multi-Block Message Test

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

The evaluator shall also test the decrypt functionality for each mode by decrypting an *i*-block

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

AES-CBC Monte Carlo Tests

The evaluator shall test the encrypt functionality for each selected key size using 100 3-tuples of pseudo-random values for plaintext, IVs, and keys.

The evaluator shall supply a single 3-tuple of pseudo-random values for each selected key size. This 3-tuple of plaintext, IV, and key is provided as input to the below algorithm to generate the remaining 99 3-tuples, and to run each 3-tuple through 1000 iterations of AES-CBC encryption.

```
# Input: PT, IV, Key
Key[0] = Key
IV[0] = IV
PT[0] = PT
for i = 0 to 99 {
    Output Key[i], IV[i], PT[0]
    for j = 0 to 999 {
        if (j == 0) {
            CT[j] = AES-CBC-Encrypt(Key[i], IV[i], PT[j])
            PT[j+1] = IV[i]
        } else {
            CT[j] = AES-CBC-Encrypt(Key[i], PT[j])
            PT[j+1] = CT[j-1]
        }
    }
    Output CT[j]
    If (KeySize == 128) Key[i+1] = Key[i] xor CT[j]
    If (KeySize == 192) Key[i+1] = Key[i] xor (last 64 bits of CT[j-1] || CT[j])
    If (KeySize == 256) Key[i+1] = Key[i] xor ((CT[j-1] | CT[j])
    IV[i+1] = CT[j]
    PT[0] = CT[j-1]
}
```

The ciphertext computed in the 1000th iteration (CT[999]) is the result for each of the 100 3-tuples for each selected key size. This result shall be compared to the result of running 1000 iterations with the same values using a known good implementation.

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

AES-GCM Test

These tests are intended to be equivalent to those described in the NIST document, “The Galois/Counter Mode (GCM) and GMAC Validation System (GCMVS) with the Addition of XPN Validation Testing,” rev. 15 Jun 2016, section 6.2, found at http://csrc.nist.gov/groups/STM/cavp/documents/mac/gcmvs.pdf.

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

The evaluator shall test the authenticated encryption functionality of AES-GCM by supplying 15 sets of Key, Plaintext, AAD, IV, and Tag data for every combination of the following parameters as selected in the ST and supported by the implementation under test:

- **Key size in bits:** Each selected and supported key size (e.g., 128, 192, or 256 bits).
- **Plaintext length in bits:** Up to four values for plaintext length: Two values that are non-zero integer multiples of 128, if supported. And two values that are non-multiples of 128, if supported.
- **AAD length in bits:** Up to five values for AAD length: Zero-length, if supported. Two values that are non-zero integer multiples of 128, if supported. And two values that are integer non-multiples of 128, if supported.
- **IV length in bits:** Up to three values for IV length: 96 bits. Minimum and maximum supported lengths, if different.
- **MAC length in bits:** Each supported length (e.g., 128, 120, 112, 104, 96).

To determine correctness, the evaluator shall compare the resulting values to those obtained by submitting the same inputs to a known-good implementation.

The evaluator shall test the authenticated decrypt functionality of AES-GCM by supplying 15 Ciphertext-Tag pairs for every combination of the above parameters, replacing Plaintext length with Ciphertext length. For each parameter combination the evaluator shall introduce an error into either the Ciphertext or the Tag such that approximately half of the cases are correct and half the cases contain errors. To determine correctness, the evaluator shall compare the resulting pass/fail status and Plaintext values to the results obtained by submitting the same inputs to a knowngood implementation.

AES-CTR Known Answer Tests

The Counter (CTR) mode is a confidentiality mode that features the application of the forward cipher to a set of input blocks, called counters, to produce a sequence of output blocks that are exclusive-ORed with the plaintext to produce the ciphertext, and vice versa. Since the Counter Mode does not specify the counter that is used, it is not possible to implement an automated test for this mode. The generation and management of the counter is tested through FCS_SSH*_EXT.1.4. If CBC and/or GCM are selected in FCS_COP.1/DataEncryption, the test activities for those modes sufficiently demonstrate the correctness of the AES algorithm. If CTR is the only selection in FCS_COP.1/DataEncryption, the AES-CBC Known Answer Test, AES-GCM Known Answer Test, or the following test shall be performed (all of these tests demonstrate the correctness of the AES algorithm):

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

known good implementation.

KAT-1 To test the encrypt functionality, the evaluator shall supply a set of 5 plaintext values for each selected keysize and obtain the ciphertext value that results from encryption of the given plaintext using a key value of all zeros.

KAT-2 To test the encrypt functionality, the evaluator shall supply a set of 5 key values for each selected keysize and obtain the ciphertext value that results from encryption of an all zeros plaintext using the given key value.

KAT-3 To test the encrypt functionality, the evaluator shall supply a set of key values for each selected keysize as described below and obtain the ciphertext values that result from AES encryption of an all zeros plaintext using the given key values. A set of 128 128-bit keys, a set of 192 192-bit keys, and/or a set of 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].

KAT-4 To test the encrypt functionality, the evaluator shall supply the set of 128 plaintext values described below and obtain the ciphertext values that result from encryption of the given plaintext using each selected keysize with a key value of all zeros (e.g. 256 ciphertext values will be generated if 128 bits and 256 bits are selected and 384 ciphertext values will be generated if all keysizes are selected). 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].

AES-CTR Multi-Block Message Test

The evaluator shall test the encrypt functionality by encrypting an i-block message where 1 less-than i less-than-or-equal to 10 (test shall be performed using AES-ECB mode). For each i the evaluator shall choose a key 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 using a known good implementation. The evaluator shall perform this test using each selected keysize.

AES-CTR Monte-Carlo Test

The evaluator shall test the encrypt functionality using 100 plaintext/key pairs. 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-ECB-Encrypt(Key, PT) PT = CT[i]
```

The ciphertext computed in the 1000th iteration is the result for that trial. This result shall be compared to the result of running 1000 iterations with the same values using a known good implementation. The evaluator 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. The evaluator shall perform this test using each selected keysize.

There is no need to test the decryption engine.

SEED-CBC Tests

For the SEED-CBC tests described below, the plaintext, ciphertext, and IV values shall consist of 128-bit blocks. To determine correctness, the evaluator shall compare the resulting values to those obtained by submitting the same inputs to a known-good implementation.

SEED-CBC Known Answer Tests

KAT-1 (Variable Key): To test the encrypt functionality of SEED-CBC, the evaluator shall supply a set of 128-bit keys (as described below) and obtain the ciphertext value that results from SEED encryption of an all-zeros plaintext using each key and an IV of all zeros.

Key i in each set shall have the leftmost i bits set to ones and the remaining bits to zeros, for values of i from 1 to 128.

To test the decrypt functionality of SEED-CBC, the evaluator shall use the same keys as above to decrypt the ciphertext results from above. Each decryption should result in an all-zeros plaintext.

KAT-2 (Variable Text): To test the encrypt functionality of SEED-CBC, the evaluator shall supply a set of 128-bit plaintext values (as described below) and obtain the ciphertext values that result from SEED-CBC encryption of each plaintext value using a 128-bits key and IV consisting of all zeros.

Plaintext value i shall have the leftmost i bits set to ones and the remaining bits set to zeros, for values of i from 1 to 128.

To test the decrypt functionality of SEED-CBC, use the plaintext values from above as ciphertext input, and SEED-CBC decrypt each ciphertext value using a 128-bits key consisting of all zeros and an IV of all zeros.

SEED-CBC Multi-Block Message Tests

The evaluator shall test the encrypt functionality by encrypting nine i-block messages for a 128-bits key size, for $2 \leq i \leq 10$. For each test, the evaluator shall supply a key, an IV, and a plaintext message of length i blocks, and encrypt the message using SEED-CBC. The resulting ciphertext values shall be compared to the results of encrypting the plaintext messages using a known good implementation.

The evaluator shall test the decrypt functionality by decrypting nine i-block messages for a 128-bits key size, for $2 \leq i \leq 10$. For each test, the evaluator shall supply a key, an IV, and a ciphertext message of length i blocks, and decrypt the message using SEED-CBC. The resulting plaintext values shall be compared to the results of decrypting the ciphertext messages using a known good implementation.

SEED-CBC Monte Carlo Tests

The evaluator shall test the encrypt functionality for each selected key size using 100 3-tuples of pseudo-random values for plaintext, IVs, and keys.

The evaluator shall supply a single 3-tuple of pseudo-random values for 128-bits key. This 3-tuple of plaintext, IV, and key is provided as input to the below algorithm to generate the remaining 99 3-tuples, and to run each 3-tuple through 1000 iterations of SEED-CBC encryption.

```
Key[0] = Key, IV[0] = IV, PT[0] = PT
for i = 0 to 99  {
    Output Key[i], IV[i], PT[0]
    for j = 0 to 999 {
        if (j == 0) {
            CT[j] = SEED-CBC-Encrypt(Key[i], IV[i], PT[j])
            PT[j+1] = IV[i]
        } else {
            CT[j] = SEED-CBC-Encrypt(Key[i], PT[j])
            PT[j+1] = CT[j-1]
        }
    }
    Output CT[j]
    Key[i+1] = Key[i] xor CT[j]
    IV[i+1] = CT[j]
    PT[0] = CT[j-1]
}
```

The ciphertext computed in the 1000th iteration (CT[999]) is the result for each of the 100 3-tuples for a 128-bits key. This result shall be compared to the result of running 1000 iterations with the same values using a known good implementation.

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

SEED-CFB Tests

For the SEED-CFB tests described below, the plaintext, ciphertext, and IV values shall consist of 128-bit blocks. To determine correctness, the evaluator shall compare the resulting values to those obtained by submitting the same inputs to a known-good implementation.

SEED-CFB Known Answer Tests

KAT-1 (Variable Key): To test the encrypt functionality of SEED-CFB, the evaluator shall supply a set of 128-bit keys(as described below) and obtain the ciphertext value that results from SEED encryption of an all-zeros plaintext using each key and an IV of all zeros.

Key i in each set shall have the leftmost i bits set to ones and the remaining bits to zeros, for values of i from 1 to 128.

To test the decrypt functionality of SEED-CFB, the evaluator shall use the same keys as above to decrypt the ciphertext results from above. Each decryption should result in an all-zeros plaintext.

KAT-2 (Variable Text): To test the encrypt functionality of SEED-CFB, the evaluator shall supply a set of 128-bit IV values (as described below) and obtain the ciphertext values that result from SEED-CFB encryption of a plaintext value consisting of all zeros using a 128-bits key and each IV.

IV value i shall have the leftmost i bits set to ones and the remaining bits set to zeros, for values of i from 1 to 128.

To test the decrypt functionality of SEED-CFB, use the IV values from above as ciphertext input, and SEED-CFB decrypt each ciphertext value using a 128-bits key consisting of all zeros and a plaintext value consisting of all zeros.

SEED-CFB Multi-Block Message Tests

Refer to SEED-CBC Multi-Block Message Tests for the required SEED-CFB testing.

SEED-CFB Monte Carlo Tests

The evaluator shall test the encrypt functionality for each selected key size using 100 3-tuples of pseudo-random values for plaintext, IVs, and keys.

The evaluator shall supply a single 3-tuple of pseudo-random values for 128-bits key. This 3-tuple of plaintext, IV, and key is provided as input to the below algorithm to generate the remaining 99 3-tuples, and to run each 3-tuple through 1000 iterations of SEED-CFB encryption.

```
Key[0] = Key, IV[0] = IV, PT[0] = PT
for i = 0 to 99  {
    Output Key[i], IV[i], PT[0]
    for j = 0 to 999 {
        if (j == 0) {
            CT[j] = PT[j] xor SEED-CFB-Encrypt(Key[i], IV[i])
            PT[j+1] = IV[i]
        } else {
            CT[j] = PT[j] xor SEED-CFB-Encrypt(Key[i], CT[j-1])
            PT[j+1] = CT[j-1]
        }
    }
    Output CT[j]
    Key[i+1] = Key[i] xor CT[j]
    IV[i+1] = CT[j]
    PT[0] = CT[j-1]
}
```

The ciphertext computed in the 1000th iteration (CT[999]) is the result for each of the 100 3-tuples for a 128-bits key. This result shall be compared to the result of running 1000 iterations with the same values using a known good implementation.

The evaluator shall test the decrypt functionality using the same test as above, exchanging CT and PT, and replacing SEED-CFB-Encrypt with SEED-CFB-Decrypt.

SEED-OFB Tests

For the SEED-OFB tests described below, the plaintext, ciphertext, and IV values shall consist of 128-bit blocks. To determine correctness, the evaluator shall compare the resulting values to those obtained by submitting the same inputs to a known-good implementation.

SEED-OFB Known Answer Tests

Refer to SEED-CFB Known Answer Tests for the required SEED-OFB testing.

SEED-OFB Multi-Block Message Tests

Refer to SEED-CFB Multi-Block Message Tests for the required SEED-OFB testing.

SEED-OFB Monte Carlo Tests

The evaluator shall test the encrypt functionality for each selected key size using 100 3-tuples of pseudo-random values for plaintext, IVs, and keys.

The evaluator shall supply a single 3-tuple of pseudo-random values for 128-bits key. This 3-tuple of plaintext, IV, and key is provided as input to the below algorithm to generate the remaining 99 3-tuples, and to run each 3-tuple through 1000 iterations of SEED-OFB encryption.

```
Key[0] = Key, IV[0] = IV, PT[0] = PT
for i = 0 to 99  {
    Output Key[i], IV[i], PT[0]
    for j = 0 to 999 {
        if (j == 0) {
            OT[j] = SEED-OFB-Encrypt(Key[i], IV[i])
            CT[j] = PT[j] xor OT[j]
            PT[j+1] = IV[i]
        } else {
            OT[j] = SEED-OFB-Encrypt(Key[i], OT[j-1])
            CT[j] = PT[j] xor OT[j]
            PT[j+1] = CT[j-1]
        }
    }
    Output CT[j]
    Key[i+1] = Key[i] xor CT[j]
    IV[i+1] = CT[j]
    PT[0] = CT[j-1]
}
```

The ciphertext computed in the 1000th iteration (CT[999]) is the result for each of the 100 3-tuples for a 128-bits key. This result shall be compared to the result of running 1000 iterations with the same values using a known good implementation.

The evaluator shall test the decrypt functionality using the same test as above, exchanging CT and PT, and replacing SEED-OFB-Encrypt with SEED-OFB-Decrypt.

SEED-CTR Tests

For the SEED-CTR tests described below, the plaintext, ciphertext, and counters shall consist of 128-bit blocks. To determine correctness, the evaluator shall compare the resulting values to those obtained by submitting the same inputs to a known-good implementation.

SEED-CTR Known Answer Tests

Refer to SEED-CFB Known Answer Tests for the required SEED-CTR testing. The evaluator shall test the encrypt functionality using the same test as SEED-CFB Tests, replacing IVs with counters.

SEED-CTR Multi-Block Message Tests

Refer to SEED-CFB Multi-Block Message Tests for the required SEED-CTR testing. The evaluator shall test the encrypt functionality using the same test as SEED-CFB Tests, replacing IVs with counters.

SEED-CTR Monte Carlo Tests

The evaluator shall test the encrypt functionality for each selected key size using 100 3-tuples of pseudo-random values for plaintext, counters, and keys.

The evaluator shall supply a single 3-tuple of pseudo-random values for 128-bits key. This 3-tuple of plaintext, counter, and key is provided as input to the below algorithm to generate the remaining 99 3-tuples, and to run each 3-tuple through 1000 iterations of SEED-CFB encryption.

```
Key[0] = Key, CTR[0] = CTR, PT[0] = PT
for i = 0 to 99  {
    Output Key[i], CTR[i], PT[0]
    for j = 0 to 999 {
        CT[j] = PT[j] xor SEED-CTR-Encrypt(Key[i], CTR[0])
        CTR[0] = (CTR[0] + 1) mod 2^128
        PT[j+1] = CT[j]
    }
    Output CT[j]
    Key[i+1] = Key[i] xor CT[j]
    PT[0] = CT[j-1]
}
```

The ciphertext computed in the 1000th iteration (CT[999]) is the result for each of the 100 3-tuples for a 128-bits key. This result shall be compared to the result of running 1000 iterations with the same values using a known good implementation.

The evaluator shall test the decrypt functionality using the same test as above, exchanging CT and PT, and replacing SEED-CTR-Encrypt with SEED-CTR-Decrypt.

SEED-CCM Tests

These tests are intended to be equivalent to those described in the NIST document, “The CCM Validation System (CCMVS),” updated 9 Jan 2012, found at http://csrc.nist.gov/groups/STM/cavp/documents/mac/CCMVS.pdf.

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

Keys: 128-bits key is supported

Associated Data: Two or three values for associated data length: The minimum (≥ 0 bytes) and maximum (≤ 32 bytes) supported associated data lengths, and 2^16 (65536) bytes, if supported.

Payload: Two values for payload length: The minimum (≥ 0 bytes) and maximum (≤ 32 bytes) supported payload lengths.

Nonces: All supported nonce lengths (7, 8, 9, 10, 11, 12, 13) in bytes.

Tag: All supported tag lengths (4, 6, 8, 10, 12, 14, 16) in bytes.

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

Variable Associated Data Test

For 128-bits key and associated data length, and any supported payload length, nonce length, and tag length, the evaluator shall supply one key value, one nonce value, and 10 pairs of associated data and payload values, and obtain the resulting ciphertext.

Variable Payload Text

For 128-bits key and payload length, and any supported associated data length, nonce length, and tag length, the evaluator shall supply one key value, one nonce value, and 10 pairs of associated data and payload values, and obtain the resulting ciphertext.

Variable Nonce Test

For 128-bits key and nonce length, and any supported associated data length, payload length, and tag length, the evaluator shall supply one key value, one nonce value, and 10 pairs of associated data and payload values, and obtain the resulting ciphertext.

Variable Tag Test

For 128-bits key and tag length, and any supported associated data length, payload length, and nonce length, the evaluator shall supply one key value, one nonce value, and 10 pairs of associated data and payload values, and obtain the resulting ciphertext.

Decryption-Verification Process Test

To test the decryption-verification functionality of SEED-CCM, for each combination of supported associated data length, payload length, nonce length, and tag length, the evaluator shall supply a key value and 15 sets of input plus ciphertext, and obtain the decrypted payload. Ten of the 15 input sets supplied should fail verification and five should pass.

SEED-GCM Tests

These tests are intended to be equivalent to those described in the NIST document, “The Galois/Counter Mode (GCM) and GMAC Validation System (GCMVS) with the Addition of XPN Validation Testing,” rev. 15 Jun 2016, section 6.2, found at http://csrc.nist.gov/groups/STM/cavp/documents/mac/gcmvs.pdf.

The evaluator shall test the authenticated encryption functionality of SEED-GCM by supplying 15 sets of Key, Plaintext, AAD, IV, and Tag data for every combination of the following parameters as

selected in the ST and supported by the implementation under test:

Key size in bits: 128-bits key is supported.

Plaintext length in bits: Up to four values for plaintext length: Two values that are non-zero integer multiples of 128, if supported. And two values that are non-multiples of 128, if supported.

AAD length in bits: Up to five values for AAD length: Zero-length, if supported. Two values that are non-zero integer multiples of 128, if supported. And two values that are integer non-multiples of 128, if supported.

IV length in bits: Up to three values for IV length: 96 bits. Minimum and maximum supported lengths, if different.

Tag length in bits: Each supported length (128, 120, 112, 104, 96, 64, 32).

To determine correctness, the evaluator shall compare the resulting values to those obtained by submitting the same inputs to a known-good implementation.

The evaluator shall test the authenticated decrypt functionality of SEED-GCM by supplying 15 Ciphertext-Tag pairs for every combination of the above parameters, replacing Plaintext length with Ciphertext length. For each parameter combination the evaluator shall introduce an error into either the Ciphertext or the Tag such that approximately half of the cases are correct and half the cases contain errors. To determine correctness, the evaluator shall compare the resulting pass/fail status and Plaintext values to the results obtained by submitting the same inputs to a known-good implementation.

HIGHT-CBC Tests

For the HIGHT-CBC tests described below, the plaintext, ciphertext, and IV values shall consist of 64-bit blocks. To determine correctness, the evaluator shall compare the resulting values to those obtained by submitting the same inputs to a known-good implementation.

HIGHT-CBC Known Answer Tests

Refer to SEED-CBC Known Answer Tests for the required HIGHT-CBC testing. The evaluator shall test the encrypt functionality using the same test as SEED-CBC Tests, replacing SEED-CBC-Encrypt with HIGHT-ECB-Encrypt.

HIGHT-CBC Multi-Block Message Tests

Refer to SEED-CBC Multi-Block Message Tests for the required HIGHT-CBC testing. The evaluator shall test the encrypt functionality using the same test as SEED-CBC Tests, replacing SEED-CBC-Encrypt with HIGHT-ECB-Encrypt.

HIGHT-CBC Monte Carlo Tests

Refer to SEED-CBC Monte Carlo Tests for the required HIGHT-CBC testing. The evaluator shall test the encrypt functionality using the same test as SEED-CBC Tests, replacing SEED-CBC-Encrypt with HIGHT-ECB-Encrypt.

HIGHT-CFB Tests

For the HIGHT-CFB tests described below, the plaintext, ciphertext, and IV values shall consist of 64-bit blocks. To determine correctness, the evaluator shall compare the resulting values to those obtained by submitting the same inputs to a known-good implementation.

HIGHT-CFB Known Answer Tests

Refer to SEED-CFB Known Answer Tests for the required HIGHT-CFB testing. The evaluator shall test the encrypt functionality using the same test as SEED-CFB Tests, replacing SEED-CFB-Encrypt with HIGHT-CFB-Encrypt.

HIGHT-CFB Multi-Block Message Tests

Refer to SEED-CFB Multi-Block Message Tests for the required HIGHT-CFB testing. The evaluator shall test the encrypt functionality using the same test as SEED-CFB Tests, replacing SEED-CFB-Encrypt with HIGHT-CFB-Encrypt.

HIGHT-CFB Monte Carlo Tests

Refer to SEED-CFB Monte Carlo Tests for the required HIGHT-CFB testing. The evaluator shall test the encrypt functionality using the same test as SEED-CFB Tests, replacing SEED-CFB-Encrypt with HIGHT-CFB-Encrypt.

HIGHT-OFB Tests

For the HIGHT-OFB tests described below, the plaintext, ciphertext, and IV values shall consist of 64-bit blocks. To determine correctness, the evaluator shall compare the resulting values to those obtained by submitting the same inputs to a known-good implementation.

HIGHT-OFB Known Answer Tests

Refer to SEED-OFB Known Answer Tests for the required HIGHT-OFB testing. The evaluator shall test the encrypt functionality using the same test as SEED-OFB Tests, replacing SEED-OFB-Encrypt with HIGHT-OFB-Encrypt.

HIGHT-OFB Multi-Block Message Tests

Refer to SEED-OFB Multi-Block Message Tests for the required HIGHT-OFB testing. The evaluator shall test the encrypt functionality using the same test as SEED-OFB Tests, replacing SEED-OFB-Encrypt with HIGHT-OFB-Encrypt.

HIGHT-OFB Monte Carlo Tests

Refer to SEED-OFB Monte Carlo Tests for the required HIGHT-OFB testing. The evaluator shall test the encrypt functionality using the same test as SEED-OFB Tests, replacing SEED-OFB-Encrypt with HIGHT-OFB-Encrypt.

HIGHT-CTR Tests

For the HIGHT-CTR tests described below, the plaintext, ciphertext, and counters shall consist of 64-bit blocks. To determine correctness, the evaluator shall compare the resulting values to those

obtained by submitting the same inputs to a known-good implementation.

HIGHT-CTR Known Answer Tests

Refer to SEED-CTR Known Answer Tests for the required HIGHT-CTR testing. The evaluator shall test the encrypt functionality using the same test as SEED-CTR Tests, replacing SEED-CTR-Encrypt with HIGHT-CTR-Encrypt.

HIGHT-CTR Multi-Block Message Tests

Refer to SEED-CTR Multi-Block Message Tests for the required HIGHT-CTR testing. The evaluator shall test the encrypt functionality using the same test as SEED-CTR Tests, replacing SEED-CTR-Encrypt with HIGHT-CTR-Encrypt.

HIGHT-CTR Monte Carlo Tests

Refer to SEED-CTR Monte Carlo Tests for the required HIGHT-CTR testing. The evaluator shall test the encrypt functionality using the same test as SEED-CTR Tests, replacing SEED-CTR-Encrypt with HIGHT-CTR-Encrypt.

LEA-CBC Tests

For the LEA-CBC tests described below, the plaintext, ciphertext, and IV values shall consist of 128-bit blocks. To determine correctness, the evaluator shall compare the resulting values to those obtained by submitting the same inputs to a known-good implementation.

LEA-CBC Known Answer Tests

KAT-1 (Variable Key): To test the encrypt functionality of LEA-CBC, the evaluator shall supply a set of keys for each selected key size (as described below) and obtain the ciphertext value that results from LEA encryption of an all-zeros plaintext using each key and an IV of all zeros. Key i in each set shall have the leftmost i bits set to ones and the remaining bits to zeros, for values of i from 1 to the key size.

To test the decrypt functionality of LEA-CBC, the evaluator shall use the same keys as above to decrypt the ciphertext results from above. Each decryption should result in an all-zeros plaintext.

KAT-2 (Variable Text): To test the encrypt functionality of LEA-CBC, for each selected key size, the evaluator shall supply a set of 128-bit plaintext values (as described below) and obtain the ciphertext values that result from LEA-CBC encryption of each plaintext value using a 128-bits key and IV consisting of all zeros.

Plaintext value i shall have the leftmost i bits set to ones and the remaining bits set to zeros, for values of i from 1 to 128.

To test the decrypt functionality of LEA-CBC, use the plaintext values from above as ciphertext input, and LEA-CBC decrypt each ciphertext value using key of each size consisting of all zeros and an IV of all zeros.

LEA-CBC Multi-Block Message Tests

The evaluator shall test the encrypt functionality by encrypting nine i-block messages for each selected key size, for 2 ≤ i ≤ 10. For each test, the evaluator shall supply a key, an IV, and a plaintext message of length i blocks, and encrypt the message using LEA-CBC. The resulting ciphertext values shall be compared to the results of encrypting the plaintext messages using a known good implementation.

The evaluator shall test the decrypt functionality by decrypting nine i-block messages for each selected key size, for 2 ≤ i ≤ 10. For each test, the evaluator shall supply a key, an IV, and a ciphertext message of length i blocks, and decrypt the message using LEA-CBC. The resulting plaintext values shall be compared to the results of decrypting the ciphertext messages using a known good implementation.

LEA-CBC Monte Carlo Tests

The evaluator shall test the encrypt functionality for each selected key size using 100 3-tuples of pseudo-random values for plaintext, IVs, and keys.

The evaluator shall supply a single 3-tuple of pseudo-random values for for each selected key size. This 3-tuple of plaintext, IV, and key is provided as input to the below algorithm to generate the remaining 99 3-tuples, and to run each 3-tuple through 1000 iterations of LEA-CBC encryption.

```
Key[0] = Key, IV[0] = IV, PT[0] = PT
for i = 0 to 99  {
    Output Key[i], IV[i], PT[0]
    for j = 0 to 999 {
        if (j == 0) {
            CT[j] = LEA-CBC-Encrypt(Key[i], IV[i], PT[j])
            PT[j+1] = IV[i]
        } else {
            CT[j] = LEA-CBC-Encrypt(Key[i], PT[j])
            PT[j+1] = CT[j-1]
        }
    }
    Output CT[j]
    If(KeySize==128) Key[i+1] = Key[i] xor CT[j]
    If(KeySize==192) Key[i+1] = Key[i] xor (last 64 bits of CT[j-1]||CT[j])
    If(KeySize==256) Key[i+1] = Key[i] xor (CT[j-1]||CT[j])
    IV[i+1] = CT[j]
    PT[0] = CT[j-1]
}
```

The ciphertext computed in the 1000th iteration (CT[999]) is the result for each of the 100 3-tuples for each selected key size. This result shall be compared to the result of running 1000 iterations with the same values using a known good implementation.

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

LEA-CFB Tests

For the LEA-CFB tests described below, the plaintext, ciphertext, and IV values shall consist of 128-bit blocks. To determine correctness, the evaluator shall compare the resulting values to those obtained by submitting the same inputs to a known-good implementation.

LEA-CFB Known Answer Tests

KAT-1 (Variable Key): To test the encrypt functionality of SEED-CFB, the evaluator shall supply a set of keys for each selected key size (as described below) and obtain the ciphertext value that results from LEA encryption of an all-zeros plaintext using each key and an IV of all zeros. Key i in each set shall have the leftmost i bits set to ones and the remaining bits to zeros, for values of i from 1 to the key size.

To test the decrypt functionality of LEA-CFB, the evaluator shall use the same keys as above to decrypt the ciphertext results from above. Each decryption should result in an all-zeros plaintext.

KAT-2 (Variable Text): To test the encrypt functionality of LEA-CFB, the evaluator shall supply a set of 128-bit IV values (as described below) and obtain the ciphertext values that result from LEA-CFB encryption of a plaintext value consisting of all zeros using a key of each size and IV. IV value i shall have the leftmost i bits set to ones and the remaining bits set to zeros, for values of i from 1 to 128.

To test the decrypt functionality of LEA-CFB, use the IV values from above as ciphertext input, and LEA-CFB decrypt each ciphertext value using a key of each size consisting of all zeros and a plaintext value consisting of all zeros.

LEA-CFB Multi-Block Message Tests

Refer to LEA-CBC Multi-Block Message Tests for the required LEA-CFB testing.

LEA-CFB Monte Carlo Tests

The evaluator shall test the encrypt functionality for each selected key size using 100 3-tuples of pseudo-random values for plaintext, IVs, and keys.

The evaluator shall supply a single 3-tuple of pseudo-random values for selected key size. This 3-tuple of plaintext, IV, and key is provided as input to the below algorithm to generate the remaining 99 3-tuples, and to run each 3-tuple through 1000 iterations of LEA-CFB encryption.

```
Key[0] = Key, IV[0] = IV, PT[0] = PT
for i = 0 to 99  {
    Output Key[i], IV[i], PT[0]
    for j = 0 to 999 {
        if (j == 0) {
            CT[j] = PT[j] xor LEA-CFB-Encrypt(Key[i], IV[i])
            PT[j+1] = IV[i]
        } else {
            CT[j] = PT[j] xor LEA-CFB-Encrypt(Key[i], CT[j-1])
            PT[j+1] = CT[j-1]
        }
    }
    Output CT[j]
    If(KeySize==128) Key[i+1] = Key[i] xor CT[j]
    If(KeySize==192) Key[i+1] = Key[i] xor (last 64 bits of CT[j-1]||CT[j])
    If(KeySize==256) Key[i+1] = Key[i] xor (CT[j-1]||CT[j])
    IV[i+1] = CT[j]
    PT[0] = CT[j-1]
}
```

The ciphertext computed in the 1000th iteration (CT[999]) is the result for each of the 100 3-tuples for each selected key size. This result shall be compared to the result of running 1000 iterations with the same values using a known good implementation.

The evaluator shall test the decrypt functionality using the same test as above, exchanging CT and PT, and replacing LEA-CFB-Encrypt with LEA-CFB-Decrypt.

LEA-OFB Tests

For the LEA-OFB tests described below, the plaintext, ciphertext, and IV values shall consist of 128-bit blocks. To determine correctness, the evaluator shall compare the resulting values to those obtained by submitting the same inputs to a known-good implementation.

LEA-OFB Known Answer Tests

Refer to LEA-CFB Known Answer Tests for the required LEA-OFB testing.

LEA-OFB Multi-Block Message Tests

Refer to [LEA-CFB Multi-Block Message Tests] for the required LEA-OFB testing.

LEA-OFB Monte Carlo Tests

The evaluator shall test the encrypt functionality for each selected key size using 100 3-tuples of pseudo-random values for plaintext, IVs, and keys.

The evaluator shall supply a single 3-tuple of pseudo-random values for 128-bits key. This 3-tuple of plaintext, IV, and key is provided as input to the below algorithm to generate the remaining 99 3-tuples, and to run each 3-tuple through 1000 iterations of LEA-OFB encryption.

```
Key[0] = Key, IV[0] = IV, PT[0] = PT
for i = 0 to 99  {
    Output Key[i], IV[i], PT[0]
    for j = 0 to 999 {
        if (j == 0) {
            OT[j] = LEA-OFB-Encrypt(Key[i], IV[i])
            CT[j] = PT[j] xor OT[j]
            PT[j+1] = IV[i]
        } else {
            OT[j] = LEA-OFB-Encrypt(Key[i], OT[j-1])
            CT[j] = PT[j] xor OT[j]
            PT[j+1] = CT[j-1]
        }
    }
    Output CT[j]
    If(KeySize==128) Key[i+1] = Key[i] xor CT[j]
    If(KeySize==192) Key[i+1] = Key[i] xor (last 64 bits of CT[j-1]||CT[j])
    If(KeySize==256) Key[i+1] = Key[i] xor (CT[j-1]||CT[j])
    IV[i+1] = CT[j]
    PT[0] = CT[j-1]
}
```

The ciphertext computed in the 1000th iteration (CT[999]) is the result for each of the 100 3-tuples for each selected key size. This result shall be compared to the result of running 1000 iterations with the same values using a known good implementation.

The evaluator shall test the decrypt functionality using the same test as above, exchanging CT and PT, and replacing LEA-OFB-Encrypt with LEA-OFB-Decrypt.

LEA-CTR Tests

For the LEA-CTR tests described below, the plaintext, ciphertext, and counters shall consist of 128-bit blocks. To determine correctness, the evaluator shall compare the resulting values to those obtained by submitting the same inputs to a known-good implementation.

LEA-CTR Known Answer Tests

Refer to LEA-CFB Known Answer Tests for the required LEA-CTR testing. The evaluator shall test the encrypt functionality using the same test as LEA-CFB Tests, replacing IVs with counters.

LEA-CTR Multi-Block Message Tests

Refer to LEA-CFB Multi-Block Message Tests for the required LEA-CTR testing. The evaluator shall test the encrypt functionality using the same test as LEA-CFB Tests, replacing IVs with counters.

LEA-CTR Monte Carlo Tests

The evaluator shall test the encrypt functionality for each selected key size using 100 3-tuples of pseudo-random values for plaintext, counters, and keys.

The evaluator shall supply a single 3-tuple of pseudo-random values for 128-bits key. This 3-tuple of plaintext, counter, and key is provided as input to the below algorithm to generate the remaining 99 3-tuples, and to run each 3-tuple through 1000 iterations of SEED-CFB encryption.

```
Key[0] = Key, CTR[0] = CTR, PT[0] = PT
for i = 0 to 99  {
    Output Key[i], CTR[i], PT[0]
    for j = 0 to 999 {
        CT[j] = PT[j] xor SEED-CTR-Encrypt(Key[i], CTR[0])
        CTR[0] = (CTR[0] + 1) mod 2^128
        PT[j+1] = CT[j]
    }
    Output CT[j]
    If(KeySize==128) Key[i+1] = Key[i] xor CT[j]
    If(KeySize==192) Key[i+1] = Key[i] xor (last 64 bits of CT[j-1]||CT[j])
    If(KeySize==256) Key[i+1] = Key[i] xor (CT[j-1]||CT[j])
    PT[0] = CT[j-1]
}
```

The ciphertext computed in the 1000th iteration (CT[999]) is the result for each of the 100 3-tuples for each selected key size. This result shall be compared to the result of running 1000 iterations with the same values using a known good implementation.

The evaluator shall test the decrypt functionality using the same test as above, exchanging CT and PT, and replacing LEA-CTR-Encrypt with LEA-CTR-Decrypt.

LEA-CCM Tests

These tests are intended to be equivalent to those described in the NIST document, “The CCM Validation System (CCMVS),” updated 9 Jan 2012, found at http://csrc.nist.gov/groups/STM/cavp/documents/mac/CCMVS.pdf.

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

Keys: All supported and selected key sizes (e.g., 128, 192, 256 bits).

Associated Data: Two or three values for associated data length: The minimum (≥ 0 bytes) and maximum (≤ 32 bytes) supported associated data lengths, and 2^16 (65536) bytes, if supported.

Payload: Two values for payload length: The minimum (≥ 0 bytes) and maximum (≤ 32 bytes) supported payload lengths.

Nonces: All supported nonce lengths (7, 8, 9, 10, 11, 12, 13) in bytes.

Tag: All supported tag lengths (4, 6, 8, 10, 12, 14, 16) in bytes.

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

Variable Associated Data Test

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

Variable Payload Text

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

Variable Nonce Test

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

Variable Tag Test

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

Decryption-Verification Process Test

To test the decryption-verification functionality of LEA-CCM, for each combination of supported associated data length, payload length, nonce length, and tag length, the evaluator shall supply a key value and 15 sets of input plus ciphertext, and obtain the decrypted payload. Ten of the 15 input sets supplied should fail verification and five should pass.

LEA-GCM Tests

These tests are intended to be equivalent to those described in the NIST document, "The Galois/Counter Mode (GCM) and GMAC Validation System (GCMVS) with the Addition of XPN Validation Testing," rev. 15 Jun 2016, section 6.2, found at http://csrc.nist.gov/groups/STM/cavp/documents/mac/gcmvs.pdf.

The evaluator shall test the authenticated encryption functionality of LEA-GCM by supplying 15 sets of Key, Plaintext, AAD, IV, and Tag data for every combination of the following parameters as selected in the ST and supported by the implementation under test:

Key size in bits: All supported and selected key sizes (e.g., 128, 192, 256 bits).

Plaintext length in bits: Up to four values for plaintext length: Two values that are non-zero integer multiples of 128, if supported. And two values that are non-multiples of 128, if supported.

AAD length in bits: Up to five values for AAD length: Zero-length, if supported. Two values that are non-zero integer multiples of 128, if supported. And two values that are integer non-multiples of 128, if supported.

IV length in bits: Up to three values for IV length: 96 bits. Minimum and maximum supported lengths, if different.

Tag length in bits: Each supported length (128, 120, 112, 104, 96, 64, 32).

To determine correctness, the evaluator shall compare the resulting values to those obtained by submitting the same inputs to a known-good implementation. The evaluator shall test the authenticated decryption functionality of LEA-GCM by supplying 15 ciphertext-tag pairs for every combination of the above parameters, replacing plaintext length with ciphertext length. For each parameter combination the evaluator shall introduce an error into either the ciphertext or the tag such that approximately half of the cases are correct and half the cases contain errors. To determine correctness, the evaluator shall compare the resulting pass/fail status and Plaintext values to the results obtained by submitting the same inputs to a known-good implementation.

2.2.6. FCS_COP.1/SigGen Cryptographic Operation (Signature Generation and Verification)

2.2.6.1. TSS

The evaluator shall examine the TSS to determine that it specifies the cryptographic algorithm and key size supported by the TOE for signature services and the overall flow of the signature generation and verification.

2.2.6.2. Guidance Documentation

The evaluator shall verify that the AGD guidance instructs the administrator how to configure the TOE to use the selected cryptographic algorithm and key size defined in the Security Target supported by the TOE for signature services.

2.2.6.3. Tests

The evaluator shall include test cases of FCS_COP.1/SigGen to the test subset. Note that FCS_COP.1/SigGen may be not mapped to the specific interface(s) after evaluator's analysis during ADV_FSP.1.

The evaluator shall produce test documentation for test cases of FCS_COP.1/SigGen. If there is no explicit external interface(s) mapped to FCS_COP.1/SigGen, the evaluator shall employ an alternative test approach (refer to CEM, section 15.2.2.).

Each section below contains tests the evaluators shall perform for each selected digital signature scheme. Based on the assignments and selections in the requirement, the evaluators choose the specific activities that correspond to those selections.

Note: The tests detailed below are not required to be performed for cryptographic functions implemented in the Root of Trust for Secure Boot (FPT_SBT_EXT.1).

ECDSA Algorithm Tests

ECDSA FIPS 186-4 Signature Generation Test

For each supported NIST curve (i.e., P-256, P-384 and P-521) and SHA function pair, the evaluator shall generate 10 1024-bit long messages and obtain for each message a public key and the resulting signature values R and S. To determine correctness, the evaluator shall use the signature verification function of a known good implementation.

ECDSA FIPS 186-4 Signature Verification Test

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

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.

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

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.

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

KCDSA Tests

The following test require the developer to provide access to a test platform that provides the evaluator with tools that are not found on the TOE in its evaluated configuration. The following or equivalent steps shall be taken to test the TSF.

Signature Generation Test

For each domain parameter (i.e., (p=2048, q= 224), (p=2048, q=256)) and SHA function pair, the evaluator shall generate a set of 10 1024-bit message. The test passes only if all the signatures with submitted parameters result in successful signature verification.

Signature Verification Test

The KCDSA Signature Verification Test tests the ability of the TSF to recognize valid and invalid

signatures. The evaluator shall provide a modulus and associated key pair (x, y) for each combination of selected curve, modulus size, and hash size. Each private key (x) is used to sign 15 pseudorandom messages of 1024 bits. For eight of the fifteen messages, the message, IR format, padding, or signature is altered so that signature verification should fail. The test passes only if all the signatures made using unaltered parameters result in successful signature verification, and all the signatures made using altered parameters result in unsuccessful signature verification.

EC-KCDSA Tests

The following test require the developer to provide access to a test platform that provides the evaluator with tools that are not found on the TOE in its evaluated configuration. The following or equivalent steps shall be taken to test the TSF.

Signature Generation Test

For each supported NIST curve (i.e., P-224/P-256, B-233/B-283, K-233/K-283) and SHA function pair, the evaluator shall generate a set of 10 1024-bit message. The test passes only if all the signatures with submitted parameters result in successful signature verification.

Signature Verification Test

The EC-KCDSA Signature Verification Test tests the ability of the TSF to recognize valid and invalid signatures. The evaluator shall provide a modulus and associated key pair (x, y) for each combination of selected curve, modulus size, and hash size. Each private key (x) is used to sign 15 pseudorandom messages of 1024 bits. For eight of the fifteen messages, the message, IR format, padding, or signature is altered so that signature verification should fail. The test passes only if all the signatures made using unaltered parameters result in successful signature verification, and all the signatures made using altered parameters result in unsuccessful signature verification.

2.2.7. FCS_COP.1/Hash Cryptographic Operation (Hash Algorithm)

2.2.7.1. TSS

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.

2.2.7.2. Guidance Documentation

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

2.2.7.3. Tests

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

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 cPP.

Note: The tests detailed below are not required to be performed for cryptographic functions implemented in the Root of Trust for Secure Boot (FPT_SBT_EXT.1).

Short Messages Test - Bit-oriented Mode

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

Short Messages Test - Byte-oriented Mode

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

Selected Long Messages Test - Bit-oriented Mode

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

Selected Long Messages Test - Byte-oriented Mode

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

Pseudorandomly Generated Messages Test

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

2.2.8. FCS_RBG_EXT.1 Cryptographic Operation (Random Bit Generation)

Documentation shall be produced—and the evaluator shall perform the activities—in accordance with Appendix E of [HCDcPP].

2.2.8.1. TSS

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.

2.2.8.2. Guidance Documentation

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

2.2.8.3. Tests

The evaluator shall include test cases of FCS_RBG_EXT.1 to the test subset. Note that FCS_RBG_EXT.1 may be not mapped to the specific interface(s) after evaluator's analysis during ADV_FSP.1.

The evaluator shall produce test documentation for test cases of FCS_RBG_EXT.1. If there is no explicit external interface(s) mapped to FCS_RBG_EXT.1, the evaluator shall employ an alternative test approach (refer to CEM, section 15.2.2.).

The evaluator shall perform 15 trials for the RNG implementation. If the RNG is configurable, the evaluator shall perform 15 trials for each configuration considering the following parameters as selected in FCS_RBG_EXT.1.1 and supported by the implementation:

1. Mechanism: Hash_DRBG, HMAC_DRBG, CTR_DRBG
2. Option for Hash_DRBG and HMAC_DRBG: selected hash function and size
3. Option for CTR_DRBG: selected block cipher and whether or not a Derivation Function (df) is used
4. Prediction Resistance enabled or disabled
5. Entropy input length
6. Nonce length
7. Personalization String length
8. Additional Input length
9. Returned Bits length

If the RNG has prediction resistance enabled, each trial consists of steps (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 (i.e., step (1)). The next two are additional input and entropy input for the first call to generate (i.e., for step (2)). The final two are additional input and entropy input for the second call to generate (i.e., for step (3)). 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 SP800-90A). The evaluator shall use a known-good implementation to verify that the Returned Bits output from step (3) is the result expected.

If the RNG does not have prediction resistance, each trial consists of steps (1) instantiate DRBG, (2) reseed, (3) generate the first block of random bits, (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 (i.e., step (1)). The fifth value is additional input to the first call to generate (i.e., for step (2)). The sixth and seventh are additional input and entropy input to the call to reseed (i.e., for step (3)). The final value is additional input to the second generate call (i.e., for step (4)). The evaluator shall use a known-good implementation to verify that the Returned Bits output from step (4) is the result expected.

The implementation passes the DRBG test if the Returned Bits result matches the Returned Bits from the known-good implementation.

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.

2.3. User Data Protection (FDP)

2.3.1. FDP_ACC.1 Subset access control

It is covered by assurance activities for FDP_ACF.1.

2.3.2. FDP_ACF.1 Security attribute based access control

2.3.2.1. TSS

The evaluator shall check to ensure that the TSS describes the functions to realize SFP defined in Table 4 and Table 5 of [HCDcPP].

2.3.2.2. Guidance Documentation

The evaluator shall check to ensure that the operational guidance contains a description of the operation to realize the SFP defined in Tables 4 and Table 5 of [HCDcPP], which is consistent with the description in the TSS.

2.3.2.3. Tests

The evaluator shall perform tests to confirm the functions to realize the SFP defined in Table 4 and Table 5 of [HCDcPP] with each type of interface (e.g., operation panel, Web interfaces) to the TOE.

The evaluator testing should include the following viewpoints:

- representative sets of the operations against representative sets of the object types defined in Table 4 and Table 5 of [HCDcPP] (including some cases where operations are either permitted or denied)
- representative sets for the combinations of the setting for security attributes that are used in access control

2.4. Identification and Authentication (FIA)

2.4.1. FIA_AFL.1 Authentication failure handling

2.4.1.1. TSS

The evaluator shall check to ensure that the TSS contains a description of the actions in the case of authentication failure (types of authentication events, the number of unsuccessful authentication attempts, actions to be conducted), which is consistent with the definition of the SFR.

2.4.1.2. Guidance Documentation

The evaluator shall check to ensure that the administrator guidance describes the setting for actions to be taken in the case of authentication failure, if any are defined in the SFR.

2.4.1.3. Tests

The evaluator shall also perform the following tests:

1. The evaluator shall check to ensure that the subsequent authentication attempts do not succeed by the behavior according to the actions defined in the SFR when unsuccessful authentication attempts reach the status defined in the SFR.
2. The evaluator shall check to ensure that authentication attempts succeed when conditions to re-enable authentication attempts are defined in the SFR and when the conditions are fulfilled.
3. The evaluator shall perform the tests 1 and 2 described above for all the targeted authentication methods when there are multiple Internal Authentication methods (e.g., password authentication, biometric authentication).
4. The evaluator shall perform the tests 1 and 2 described above for all interfaces when there are multiple interfaces (e.g., operation panel, Web interfaces) that implement authentication attempts.

2.4.2. FIA_ATD.1 User attribute definition

2.4.2.1. TSS

The evaluator shall check to ensure that the TSS contains a description of the user security attributes that the TOE uses to implement the SFR, which is consistent with the definition of the SFR.

2.4.3. FIA_PMG_EXT.1 Password Management

2.4.3.1. Guidance Documentation

The evaluator shall examine the operational guidance to determine that it provides guidance to security administrators on the composition of passwords, and that it provides instructions on setting the minimum password length.

2.4.3.2. Tests

The evaluator shall also perform the following tests:

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

b. Test 2: The evaluator shall compose passwords that do not meet the requirements in some way. For each password, the evaluator shall verify that the TOE does not support the password. While the evaluator is not required (nor is it feasible) to test all possible compositions of passwords, the evaluator shall ensure that the TOE enforces the allowed characters and the minimum length listed in the requirement and justify the subset of those characters chosen for testing.

2.4.4. FIA_UAU.1 Timing of authentication

2.4.4.1. TSS

The evaluator shall check to ensure that the TSS describes all the identification and authentication mechanisms that the TOE provides (e.g., Internal Authentication and authentication by external servers).

The evaluator shall check to ensure that the TSS identifies all the interfaces to perform identification and authentication (e.g., identification and authentication from operation panel or via Web interfaces).

The evaluator shall check to ensure that the TSS describes the protocols (e.g., LDAP, Kerberos, OCSP) used in performing identification and authentication when the TOE exchanges identification and authentication with External Authentication servers.

The evaluator shall check to ensure that the TSS contains a description of the permitted actions before performing identification and authentication, which is consistent with the definition of the SFR.

2.4.4.2. Guidance Documentation

The evaluator shall check to ensure that the administrator guidance contains descriptions of identification and authentication methods that the TOE provides (e.g., External Authentication, Internal Authentication) as well as interfaces (e.g., identification and authentication from operation panel or via Web interfaces), which are consistent with the ST (TSS).

2.4.4.3. Tests

The evaluator shall also perform the following tests:

1. The evaluator shall check to ensure that identification and authentication succeeds, enabling the access to the TOE when using authorized data.
2. The evaluator shall check to ensure that identification and authentication fails, disabling the access to the TOE afterwards when using unauthorized data.

The evaluator shall perform the tests described above for each of the authentication methods that the TOE provides (e.g., External Authentication, Internal Authentication) as well as interfaces (e.g., identification and authentication from operation panel or via Web interfaces).

2.4.5. FIA_UAU.7 Protected authentication feedback

2.4.5.1. TSS

The evaluator shall check to ensure that the TSS contains a description of the authentication information feedback provided to users while the authentication is in progress, which is consistent with the definition of the SFR.

2.4.5.2. Tests

The evaluator shall also perform the following tests:

1. The evaluator shall check to ensure that only the information defined in the SFR is provided for feedback by attempting identification and authentication.
2. The evaluator shall perform the test 1 described above for all the interfaces that the TOE provides (e.g., operation panel, identification and authentication via Web interface).

2.4.6. FIA_UID.1 Timing of identification

It is covered by assurance activities for FIA_UAU.1.

2.4.7. FIA_USB.1 User-subject binding

2.4.7.1. TSS

The evaluator shall check to ensure that the TSS contains a description of rules for associating security attributes with the users who succeed identification and authentication, which is consistent with the definition of the SFR.

2.4.7.2. Tests

The evaluator shall also perform the following test:

The evaluator shall check to ensure that security attributes defined in the SFR are associated with the users who succeed identification and authentication (it is ensured in the tests of FDP_ACF.1) for each role that the TOE supports (e.g., User and Administrator).

2.5. Security Management (FMT)

2.5.1. FMT_MOF.1 Management of security functions behavior

2.5.1.1. TSS

The evaluator shall check to ensure that the TSS contains a description of the management functions that the TOE provides as well as user roles that are permitted to manage the functions, which is consistent with the definition of the SFR.

The evaluator shall check to ensure that the TSS identifies interfaces to operate the management functions.

2.5.1.2. Guidance Documentation

The evaluator shall check to ensure that the administrator guidance describes the operation methods for users of the given roles defined in the SFR to operate the management functions.

2.5.1.3. Tests

The evaluator shall also perform the following tests:

1. The evaluator shall check to ensure that users of the given roles defined in the SFR can operate the management functions in accordance with the operation methods specified in the administrator guidance.
2. The evaluator shall check to ensure that the operation results are appropriately reflected.
3. The evaluator shall check to ensure that U.NORMAL is not permitted to operate the management functions.

2.5.2. FMT_MSA.1 Management of security attributes

2.5.2.1. TSS

The evaluator shall check to ensure that the TSS contains a description of possible operations for security attributes and given roles to those security attributes, which is consistent with the definition of the SFR.

2.5.2.2. Guidance Documentation

The evaluator shall check to ensure that the administrator guidance contains a description of possible operations for security attributes and given roles to those security attributes, which is

consistent with the definition of the SFR.

The evaluator shall check to ensure that the administrator guidance describes the timing of modified security attributes.

2.5.2.3. Tests

The evaluator shall also perform the following tests:

1. The evaluator shall check to ensure that users of the given roles defined in the SFR can perform operations to the security attributes in accordance with the operation methods specified in the administrator guidance.
2. The evaluator shall check to ensure that the operation results are appropriately reflected as specified in the administrator guidance.
3. The evaluator shall check to ensure that a user that is not part of an authorized role defined in the SFR is not permitted to perform operations on the security attributes.

2.5.3. FMT_MSA.3 Static attribute initialization

2.5.3.1. TSS

The evaluator shall check to ensure that the TSS describes mechanisms to generate security attributes which have properties of default values, which are defined in the SFR.

2.5.3.2. Tests

If U.ADMIN is selected, then testing of this SFR is performed in the tests of FDP_ACF.1.

2.5.4. FMT_MTD.1 Management of TSF data

2.5.4.1. Guidance Documentation

The evaluator shall check to ensure that the administrator guidance identifies the management operations and authorized roles consistent with the SFR.

The evaluator shall check to ensure that the administrator guidance describes how the assignment of roles is managed.

The evaluator shall check to ensure that the administrator guidance describes how security attributes are assigned and managed.

The evaluator shall check to ensure that the administrator guidance describes how the security-related rules (.e.g., access control rules, timeout, number of consecutive logon failures) are configured.

2.5.4.2. Tests

The evaluator shall perform the following tests:

1. The evaluator shall check to ensure that users of the given roles defined in the SFR can perform

operations to TSF data in accordance with the operation methods specified in the administrator guidance.

2. The evaluator shall check to ensure that the operation results are appropriately reflected as specified in the administrator guidance.
3. The evaluator shall check to ensure that no users other than users of the given roles defined in the SFR can perform operations to TSF data.

2.5.5. FMT_SMF.1 Specification of Management Functions

2.5.5.1. TSS

The evaluator shall check the TSS to ensure that the management functions are consistent with the assignment in the SFR.

2.5.5.2. Guidance Documentation

The evaluator shall check the guidance documents to ensure that management functions are consistent with the assignment in the SFR, and that their operation is described.

2.5.6. FMT_SMR.1 Security roles

2.5.6.1. TSS

The evaluator shall check to ensure that the TSS contains a description of security related roles that the TOE maintains, which is consistent with the definition of the SFR.

2.5.6.2. Tests

As for tests of this SFR, it is performed in the tests of FMT_MOF.1, FMT_MSA.1, and FMT_MTD.1.

2.6. Protection of the TSF (FPT)

2.6.1. FPT_SBT_EXT.1 Secure Boot

2.6.1.1. TSS

The evaluator shall verify that the TSS describes each chain of trust and its associated Root of Trust. For each chain of trust, the evaluator shall verify:

- that the TSS describes the hash, digital signature, or message authentication verification performed by the TOE at boot.
- that the TSS describes data and/or key contained in the Root of Trust and how they are used for firmware/software integrity verification.
- that the TSS describes how the Root of Trust is immutable.

Note: Due to the proprietary nature of this information, the vendor may provide the information pertaining to the root of trust in a separate document. This document must be provided for review

to the evaluation lab and the scheme for review but will not be posted on the approved products list page.

2.6.1.2. Guidance

The evaluator shall examine the guidance documentation and verify that procedures are provided on the remediation of an integrity verification failure in a chain of trust.

Note: Acceptable actions for remediation of the device include reverting to a previous TOE image, reinstalling the TOE, performing a factory reset of the TOE, or contacting vendor support for assistance.

2.6.1.3. Tests

The evaluator shall carry out the following tests.

1. During initial boot of the TOE, the evaluator shall review the initialization output or audit records and verify that the TOE successfully performs verification of the firmware/software.
2. For every element in each chain of trust, the evaluator shall attempt to boot the TOE using firmware/software with an invalid hash, digital signature, or message authentication verification and verify that the verification check fails and the TOE doesn't execute corrupted firmware/software.

   Note: Verification of the Root of Trust is out of scope for Test 2.

3. For every element in each chain of trust, the evaluator shall attempt to boot the TOE using a corrupted firmware image and verify that upon failure, the TOE performs the action selected within FPT_SBT_EXT.1.4.

   Note: Corruption of the Root of Trust is out of scope for Test 3.

   a. (conditional) If 'revert to previous TOE image' is selected, the evaluator, following a failed boot attempt, shall review the guidance documentation, verifies that the TOE performed the action of reverting to a previous TOE image and confirms that the TOE returns to an operational state following the remediation action.

      Note: The administrator might need to take an action to perform the remediation action.

   b. (conditional) If 'reinstall TOE image' is selected, the evaluator, following a failed boot attempt, shall review the guidance documentation, verifies that the TOE performed the action of reinstalling the TOE image and confirms that the TOE returns to an operational state following the remediation action.

      Note: The administrator might need to take an action to perform the remediation action.

   c. (conditional) If 'perform factory reset' is selected, the evaluator, following a failed boot attempt, shall review the guidance documentation, verifies that the TOE performs a factory reset and confirms that the TOE returns to an initialized state where it can be returned into an operational state following the remediation action.

      Note: The administrator might need to take an action to perform the remediation action.

d. (conditional) If ‘indicate a need to contact vendor support’ is selected, the evaluator, following a failed boot attempt, shall review the guidance documentation and verifies that the TSF provides an indication to contact vendor support.

2.6.2. FPT_SKP_EXT.1 Protection of TSF Data

2.6.2.1. TSS

The evaluator shall examine the TSS to determine that it details how any pre-shared keys, symmetric keys, and private keys are stored and that they are unable to be viewed through an interface designed specifically for that purpose, as outlined in the application note. If these values are not stored in plaintext, the TSS shall describe how they are protected/obscured.

2.6.3. FPT_STM.1 Reliable time stamps

2.6.3.1. TSS

The evaluator shall check to ensure that the TSS describes mechanisms that provide reliable time stamps.

2.6.3.2. Guidance Documentation

The evaluator shall check to ensure that the guidance describes the method of setting the time.

2.6.3.3. Tests

The evaluator shall also perform the following tests:

1. The evaluator shall check to ensure that the time is correctly set up in accordance with the guidance or external network services (e.g., NTP).
2. The evaluator shall check to ensure that the time stamps are appropriately provided.

2.6.4. FPT_TST_EXT.1 TSF testing

2.6.4.1. TSS

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

2.6.4.2. Guidance Documentation

The evaluator shall also ensure that the operational guidance describes the possible errors that may result from such tests, and actions the administrator should take in response; these possible errors shall correspond to those described in the TSS.

2.6.4.3. Tests

Self-test is intended to detect malfunctions which may compromise the TSF. Since the integrity of the firmware/software is guaranteed by FPT_SBT_EXT, the function for FPT_TST_EXT should address the malfunction detection like DRBG self-test defined in ISO/IEC 18031:2011.

Although formal compliance is not mandated, the self-tests performed should aim for a level of confidence comparable to:

- [FIPS 140-2], chap. 4.9.1, Cryptographic algorithm test for the verification of the correct operation of cryptographic functions. Alternatively, national requirements of any CCRA member state for the security evaluation of cryptographic functions should be considered as appropriate.

The evaluator shall either verify that the self-tests described above are carried out during initial start-up or that the developer has justified any deviation from this.

2.6.5. FPT_TUD_EXT.1 Trusted Update

2.6.5.1. TSS

The evaluator shall check to ensure that the TSS contains a description of mechanisms that verify software for update when performing updates, which is consistent with the definition of the SFR.

The evaluator shall check to ensure that the TSS identifies interfaces for administrators to obtain the current version of the TOE as well as interfaces to perform updates.

2.6.5.2. Guidance Documentation

The evaluator shall check to ensure that the administrator guidance contains descriptions of the operation methods to obtain the TOE version as well as the operation methods to start update processing, which are consistent with the description of the TSS.

2.6.5.3. Tests

The evaluator shall also perform the following tests:

1. The evaluator shall check to ensure the current version of the TOE can be appropriately obtained by means of the operation methods specified by the administrator guidance.
2. The evaluator shall check to ensure that the verification of the data for updates of the TOE succeeds using authorized data for updates by means of the operation methods specified by the administrator guidance.
3. The evaluator shall check to ensure that only administrators can implement the application for updates using authorized data for updates.
4. The evaluator shall check to ensure that the updates are correctly performed by obtaining the current version of the TOE after the normal updates finish.
5. The evaluator shall check to ensure that the verification of the data for updates of the TOE fails using unauthorized data for updates by means of the operation methods specified by the administrator guidance. (The evaluator shall also check those cases where hash verification

mechanism, if selected in FPT_TUD_EXT.1.3, and digital signature verification mechanism fail.)

2.7. TOE Access (FTA)

2.7.1. FTA_SSL.3 TSF-initiated termination

2.7.1.1. TSS

The evaluator shall check to ensure that the TSS describes the types of user sessions to be terminated (e.g., user sessions via operation panel or Web interfaces) after a specified period of user inactivity.

2.7.1.2. Guidance Documentation

The evaluator shall check to ensure that the guidance describes the default time interval and, if it is settable, the method of setting the time intervals until the termination of the session.

2.7.1.3. Tests

The evaluator shall also perform the following tests:

1. If it is settable, the evaluator shall check to ensure that the time until the termination of the session can be set up by the method of setting specified in the administrator guidance.
2. The evaluator shall check to ensure that the session terminates after the specified time interval.
3. The evaluator shall perform the tests 1 and 2 described above for all the user sessions identified in the TSS.

2.8. Trusted Channels (FTP)

2.8.1. FTP_ITC.1 Inter-TSF trusted channel

2.8.1.1. TSS

The evaluator shall examine the TSS to determine that, for all communications with authorized IT entities identified in the requirement, each secure communications mechanism is identified in terms of the allowed protocols for that IT entity. The evaluator shall also confirm that all protocols listed in the TSS are specified and included in the requirements in the ST.

2.8.1.2. Guidance Documentation

The evaluator shall confirm that the operational guidance contains instructions for establishing the allowed protocols with each authorized IT entity, and that it contains recovery instructions should a connection be unintentionally broken.

2.8.1.3. Tests

The evaluator shall also perform the following tests:

1. The evaluators shall ensure that communications using each protocol with each authorized IT entity is tested during the course of the evaluation, setting up the connections as described in the operational guidance and ensuring that communication is successful.
2. For each protocol that the TOE or trusted IT entity can initiate as defined in the requirement, the evaluator shall follow the operational guidance to ensure that in fact the communication channel can be initiated from the TOE / trusted IT entity.
3. The evaluator shall ensure, for each communication channel with an authorized IT entity, the channel data are not sent in plaintext.
4. The evaluator shall ensure, for each protocol associated with each authorized IT entity tested during test 1, the connection is physically interrupted. The evaluator shall ensure that when physical connectivity is restored, communications are appropriately protected.

Further assurance activities are associated with the specific protocols.

2.8.2. FTP_TRP.1/Admin Trusted path (for Administrators)

2.8.2.1. TSS

The evaluator shall examine the TSS to determine that the methods of remote TOE administration are indicated, along with how those communications are protected. The evaluator shall also confirm that all protocols listed in the TSS in support of TOE administration are consistent with those specified in the requirement, and are included in the requirements in the ST.

2.8.2.2. Guidance Documentation

The evaluator shall confirm that the operational guidance contains instructions for establishing the remote administrative sessions for each supported method.

2.8.2.3. Tests

The evaluator shall also perform the following tests:

1. The evaluators shall ensure that communications using each specified (in the operational guidance) remote administration method is tested during the course of the evaluation, setting up the connections as described in the operational guidance and ensuring that communication is successful.
2. For each method of remote administration supported, the evaluator shall follow the operational guidance to ensure that there is no available interface that can be used by a remote user to establish a remote administrative session without invoking the trusted path.
3. The evaluator shall ensure, for each method of remote administration, the channel data are not sent in plaintext.

Further assurance activities are associated with the specific protocols.

Chapter 3. Evaluation Activities for Conditionally Mandatory Requirements

3.1. Confidential Data on Nonvolatile Storage Devices

3.1.1. FPT_KYP_EXT.1 Protection of Key and Key Material

3.1.1.1. TSS

The evaluator shall examine the TSS and verify it identifies the methods used to protect keys stored in non-volatile memory.

3.1.1.2. KMD

The evaluator shall verify the KMD to ensure it describes the storage location of all keys and the protection of all keys stored in non-volatile memory. The description of the key chain shall be reviewed to ensure the selected method is followed for the storage of wrapped or encrypted keys in non-volatile memory and plaintext keys in non-volatile memory meet one of the exemptions under the selection “only store plaintext keys that meet any one of the following criteria”.

3.1.2. FCS_KYC_EXT.1 Key Chaining

3.1.2.1. TSS

The evaluator shall verify that the TSS contains a high-level description of the BEV sizes – that it supports BEV outputs of no fewer 128 bits for products that support only AES-128, and that no fewer than 256 bits for products that support AES-256.

3.1.2.2. KMD

The evaluator shall examine the KMD to ensure that it describes a high level description of the key hierarchy for all accepted BEVs. The evaluator shall examine the KMD to ensure it describes the key chain in detail. The description of the key chain shall be reviewed to ensure it maintains a chain of keys using key wrap, submask combining, or key encryption.

The evaluator shall examine the KMD to ensure that it describes how the key chain process functions, such that it does not expose any material that might compromise any key in the chain. (e.g., using a key directly as a compare value against a TPM) This description must include a diagram illustrating the key hierarchy implemented and detail where all keys and keying material is stored or what it is derived from. The evaluator shall examine the key hierarchy to ensure that at no point the chain could be broken without a cryptographic exhaust or the initial authorization value and the effective strength of the BEV is maintained throughout the key chain.

The evaluator shall verify the KMD includes a description of the strength of keys throughout the key chain.

The evaluator shall verify the KMD description describes how the start (or root) of key chain is

protected if it is protected by another key that is not in the key chain or a protected storage device. One approach the evaluator can take is to make sure that the device, which protects the start (or root) of key chain, is a protected storage device or separate co-processor by inspection of type identifier of the device or published document referred from KMD. But the evaluator does not need to verify the protection method in detail for the start (or root) of key chain if the key to protect is not in the key chain specified in FCS_KYC_EXT.1. Also the evaluator does not need to verify the protection method implemented in the protected storage device if the key is stored in the protected storage device.

3.1.3. FDP_DSK_EXT.1 Protection of Data on Disk

In the EAs, below, “Device” refers to the Nonvolatile Storage Device from FDP_DSK_EXT.1. If the TOE contains more than one applicable Device, then the EAs are performed as necessary on each such Device.

3.1.3.1. TSS

If the self-encrypting device option is selected, the Device must be certified in conformance to the current Full Disk Encryption collaborative Protection Profile. The tester shall confirm that the specific SED is listed in the TSS, documented and verified to be CC certified against the FDE EE cPP.

The evaluator shall examine the TSS to ensure that the description is comprehensive in how the data is written to the Device and the point at which the encryption function is applied.

For the cryptographic functions that are provided by the Operational Environment, the evaluator shall check the TSS to ensure it describes the interface(s) used by the TOE to invoke this functionality.

The evaluator shall verify that the TSS describes the initialization of the Device at shipment of the TOE, or by the activities the TOE performs to ensure that it encrypts all the storage devices entirely when a user or administrator first provisions the Device. The evaluator shall verify the TSS describes areas of the Device that it does not encrypt (e.g., portions that do not contain confidential data boot loaders, partition tables, etc.).

If data (e.g., D.USER.JOB, D.TSF.PROT) other than D.USER.DOC and D.TSF.CONF are encrypted, the evaluator shall verify that TSS identifies all such data and states that no other customer-supplied data are encrypted.

If any D.USER.DOC or D.TSF.CONF are transparently encrypted and written to disk via mechanisms other than operating TSFI, the evaluator shall verify that the TSS identifies those mechanisms and describes at a high level how the associated data are encrypted. The swap files and core dump may potentially contain D.USER.DOC or D.TSF.CONF and should be considered.

3.1.3.2. Guidance Documentation

The evaluator shall review the AGD guidance to determine that it describes the initial steps needed to enable the Device encryption function, including any necessary preparatory steps. The guidance shall provide instructions that are sufficient to ensure that all Devices will be encrypted when encryption is enabled or at shipment of the TOE. If the TOE supports multiple Device encryptions, the evaluator shall examine the administration guidance to ensure the initialization procedure

encrypts all Devices.

3.1.3.3. KMD

The evaluator shall verify the KMD includes a description of the data encryption engine, its components, and details about its implementation (e.g., for hardware: integrated within the device's main SOC or separate co-processor, for software: initialization of the Device, drivers, libraries (if applicable), logical interfaces for encryption/decryption, and areas which are not encrypted (e.g., boot loaders, portions that do not contain confidential data, partition tables, etc.)). The evaluator shall verify the KMD provides a functional (block) diagram showing the main components (such as memories and processors) and the data path between, for hardware, the Device's interface and the Device's persistent media storing the data, or for software, the initial steps needed to the activities the TOE performs to ensure it encrypts the storage device entirely when a user or administrator first provisions the product. The hardware encryption diagram shall show the location of the data encryption engine within the data path. The evaluator shall validate that the hardware encryption diagram contains enough detail showing the main components within the data path and that it clearly identifies the data encryption engine.

The evaluator shall verify the KMD provides sufficient instructions to ensure that when the encryption is enabled, the TOE encrypts all Nonvolatile Storage Devices. The evaluator shall verify that the KMD describes the data flow from the interface to the Device's persistent media storing the data. The evaluator shall verify that the KMD provides information on those conditions in which the data bypasses the data encryption engine (e.g., read-write operations to an unencrypted area).

The evaluator shall verify that the KMD provides a description of the boot initialization, the encryption initialization process, and at what moment the product enables the encryption. If encryption can be enabled and disabled, the evaluator shall ensure the KMD contains a description of how data is protected from transfer before cryptographic initialization.

3.1.3.4. Tests

The evaluator shall perform the following tests:

Test 1. Write data to Storage device: Perform writing to the storage device with operating TSFI which enforce write process of D.USER.DOC and D.TSF.CONF.

Test 2. Confirm that written data are encrypted: Verify there are no plaintext data present in the encrypted range written by Test 1; and, verify that the data can be decrypted by proper key and key material.

All TSFIs for writing D.USER.DOC and D.TSF.CONF should be tested by Test 1 and Test 2 above.

Test 3. [Conditional: If the ST author claims FPT_WIPE_EXT.1 with cryptographic erase, and if data other than D.USER.DOC and D.TSF.CONF are encrypted] Write the data to the storage device with operating TSFI which enforce write process of the data.

Test 4. [Conditional: If the ST author claims FPT_WIPE_EXT.1 with cryptographic erase, and if data other than D.USER.DOC and D.TSF.CONF are encrypted] Verify that the data written in Test 3 is not in plaintext form; and verify that the data can be decrypted by proper key and key material.

Test 5. [Conditional: If any D.USER.DOC or D.TSF.CONF are transparently encrypted and written to disk via mechanisms other than operating TSFI] Using a special tooling that the developer shall provide, the evaluator shall write the known data to the storage through transparent encryption.

Test 6. [Conditional: If any D.USER.DOC or D.TSF.CONF are transparently encrypted and written to disk via mechanisms other than operating TSFI] Verify that the data written in Test 5 is not in plaintext form; and verify that the data can be decrypted by proper key and key material.

Test 5 and Test 6 should be performed for each mechanism not involving the operation of TSFIs described in the TSS.

3.2. PSTN Fax-Network Separation

3.2.1. FDP_FXS_EXT.1 Fax separation

The following assurance activities are required when the TOE has a fax communication function to transmit and receive via PSTN.

3.2.1.1. TSS

The evaluator shall check the TSS to ensure that it describes:

1. The fax interface use cases
2. The capabilities of the fax modem and the supported fax protocols
3. The data that is allowed to be sent or received via the fax interface
4. How the TOE can only be used for transmitting or receiving User Data using fax protocols

3.2.1.2. Guidance Documentation

The evaluator shall check to ensure that the operational guidance contains a description of the fax interface in terms of usage and available features.

3.2.1.3. Tests

The evaluator shall test to ensure that the fax interface can only be used for transmitting or receiving User Data using fax protocols. Testing will be dependent upon how the TOE enforces this requirement. The following tests shall be used and supplemented with additional testing or a rationale as to why the following tests are sufficient:

1. Verify that the TOE accepts incoming calls using fax carrier protocols and rejects calls that use data carriers. For example, this may be achieved using a terminal application to issue modem commands directly to the TOE from a PC modem (issue terminal command: ‘ATDT <TOE Fax Number>’) – the TOE should answer the call and disconnect.
2. Verify TOE negotiates outgoing calls using fax carrier protocols and rejects negotiation of data carriers. For example, this may be achieved by using a PC modem to attempt to receive a call from the TOE (submit a fax job from the TOE to <PC modem number>, at PC issue terminal command: ‘ATA’) – the TOE should disconnect without negotiating a carrier.

3.3. Asymmetric Key Generation

3.3.1. FCS_CKM.1/AKG Cryptographic Key Generation (for asymmetric keys)

3.3.1.1. TSS

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

The TSS may refer to the KMD, described in Appendix F of [HCDcPP], that may not be made available to the public.

The evaluator shall examine the TOE summary specification to verify that it describes how the TOE obtains a key based on input from a random bit generator as specified in FCS_RBG_EXT.1. The evaluator shall review the TOE summary specification to verify that it describes how the functionality described by FCS_RBG_EXT.1 is invoked.

If the TOE uses the generated key in a key chain/hierarchy, then the evaluator shall examine the TOE summary specification to confirm that it describes the followings:

1. if KCDSA is selected, then the TOE summary specification describes which methods for generating p and q are used, and
2. how the key is used as part of the key chain/hierarchy

3.3.1.2. Guidance Documentation

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.

3.3.1.3. Tests

The evaluator shall include test cases of FCS_CKM.1/AKG to the test subset. Note that FCS_CKM.1/AKG may be not mapped to the specific interface(s) after evaluator's analysis during ADV_FSP.1.

The evaluator shall produce test documentation for test cases of FCS_CKM.1/AKG. If there is no explicit external interface(s) mapped to FCS_CKM.1/AKG, the evaluator shall employ an alternative test approach (refer to CEM, section 15.2.2.)

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

The evaluator shall verify the implementation of RSA Key Generation by the TOE using the Key

Generation test. This test verifies the ability of the TSF to correctly produce values for the key components including the public verification exponent *e*, the private prime factors *p* and *q*, the public modulus *n* and the calculation of the private signature exponent *d*.

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

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

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

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

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)

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

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

the field prime p:

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

and two ways to generate the cryptographic group generator g:

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

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

- 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.

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

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

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

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

for each FFC parameter set and key pair.

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

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

Key Generation for KCDSA

The evaluator shall verify the implementation of KCDSA 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 private signing exponent x, the public verification exponent y, the private prime factor p and q, the generator g.

FIPS 186-4 Key Pair generation specifies 2 methods for generating the primes p and q.

These are:

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

and two ways to generate the cryptographic group generator g:

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

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

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

To test the key generation method for the Probable primes method and Provable primes method, the evaluator must seed the TSF key generation routine with sufficient data to deterministically generate the KCDSA key pair.

For each key length supported, the evaluator shall have the TSF generate 10 key pairs. The evaluator shall verify the correctness of the TSF's implementation by comparing values generated by the TSF with those generated by a known good implementation using the same input parameters.

If the TOE generates Random Probable Primes then if possible, the Random Probable primes method should also be verified against a known good implementation as described above. If verification against a known good implementation is not possible, the evaluator shall have the TSF generate 25 key pairs for each supported key length and verify that all of the following are true:

1. 2^(len(p)-1)<p<2^(len(p))
2. 2^(len(q)-1)<p<2^(len(q))
3. q divides p-1
4. 1<g<p-1
5. g^((p-1)/q) mod p>1
6. 0<x<q
7. g^q mod p =1
8. g^x mod p=y.

Key Generation for EC-KCDSA

ECC Key Generation Test

For each selected curve, and for each key pair generation method as described in FIPS 186-4, section B.4, the evaluator shall require the implementation under test to generate 10 private/public key pairs (d, Q=d^(-1) G), where G is a base point. The private key, d, shall be generated using a random bit generator as specified in FCS_RBG_EXT.1. The private key, d, is used to compute the

public key, Q'. The evaluator shall confirm that 0<d<n (where n is the order of the group), and the computed value Q' is then compared to the generated public/private key pairs' public key, Q, to confirm that Q is equal to Q'.

Public Key Validation (PKV) Test

For each supported curve, the evaluator shall generate 10 private/public key pairs using the key generation function of a known good implementation and modify some public key values so that they are incorrect, leaving values unchanged (i.e., correct). To determine correctness, the evaluator shall submit the 10 key pairs to the public key validation (PKV) function of the TOE and shall confirm that the results correspond as expected to the modified and unmodified values.

3.4. Network Communications

3.4.1. FTP_TRP.1/NonAdmin Trusted path (for Non-administrators)

3.4.1.1. TSS

The evaluator shall examine the TSS to determine that the methods of remote TOE access for non-administrative users are indicated, along with how those communications are protected.

The evaluator shall also confirm that all protocols listed in the TSS in support of remote TOE access are consistent with those specified in the requirement, and are included in the requirements in the ST.

3.4.1.2. Guidance Documentation

The evaluator shall confirm that the operational guidance contains instructions for establishing the remote user sessions for each supported method.

3.4.1.3. Tests

The evaluator shall also perform the following tests:

1. The evaluators shall ensure that communications using each specified (in the operational guidance) remote user access method is tested during the course of the evaluation, setting up the connections as described in the operational guidance and ensuring that communication is successful.
2. For each method of remote access supported, the evaluator shall follow the operational guidance to ensure that there is no available interface that can be used by a remote user to establish a remote user session without invoking the trusted path.
3. The evaluator shall ensure, for each method of remote user access, the channel data are not sent in plaintext.

Further EAs are associated with the specific protocols.

Chapter 4. Evaluation Activities for Optional Requirements

4.1. USER.DOC Unavailability

4.1.1. FDP_UDU_EXT Document Unavailability

4.1.1.1. TSS

The evaluator shall examine the TSS to ensure that the description is comprehensive in describing where image data is stored and how and when it is overwritten.

The evaluator shall examine the TSS to ensure that the TSS states whether the TOE stores D.USER.DOC on any wear-leveling storage device.

The evaluator shall examine the TSS to ensure that the TSS describes the type(s) of overwrite (e.g., single overwrite with zeros) of D.USER.DOC that the TOE performs.

4.1.1.2. Guidance Documentation

The evaluator shall check to ensure that the operational guidance describes the type(s) of overwrite (e.g., single overwrite with zeros) of user document data that the TOE performs.

4.1.1.3. Tests

The evaluator shall include tests related to this function in the set of tests performed in FMT_SMF.1.

4.2. Data Wiping

4.2.1. FPT_WIPE_EXT Data Wiping

4.2.1.1. TSS

The evaluator shall examine the TSS to ensure that the description is comprehensive in describing what customer-supplied data is to be wiped, where it is stored, and how it is made unavailable.

If a media-specific method is selected, the evaluator shall ensure that the TSS describes the method in sufficient detail to determine whether any encrypted D.USER or D.TSF remains on the storage media.

If multiple methods are specified, the evaluator shall ensure that the description clearly states what methods are used for each type of storage and/or customer-supplied data.

The evaluator shall ensure the storage medium(s) subject to overwrite are identified and the storage medium(s) leverage functionality that matches the selection on wear-leveling.

The evaluator shall examine the TSS to ensure that the TSS describes the type(s) of overwrite (e.g.,

single overwrite with zeros) of D.USER.DOC that the TOE performs.

If FPT_WIPE_EXT.1 claims all the customer-supplied information is made unavailable using cryptographic erase only, the evaluator shall confirm that all the customer-supplied information is encrypted by the TSF according to FDP_DSK_EXT.1.

4.2.1.2. Guidance Documentation

The evaluator shall check to ensure that the operational guidance contains instructions for initiating the Wipe function.

The evaluator shall check to ensure that the operational guidance describes the type(s) of information that is wiped, and the method(s) used.

The evaluator shall check to ensure that the operational guidance describes the indication to administrators that the wipe is in progress (optional) and that the wipe completes.

If a wipe method is used that leaves any encrypted D.USER or D.TSF on the storage media, the evaluator shall check to ensure that the operational guidance includes a statement cautioning the user about that condition.

The evaluator shall check to ensure that the operational guidance describes the type(s) of overwrite (e.g., single overwrite with zeros) of user document data that the TOE performs.

4.2.1.3. Tests

Tests activities consists of the six tests below. Test 2 and 6 are mandatory in all cases. Tests 3, 4, and 5 are conditional on the wipe method selected by the ST author.

Test 1: [Conditional: D.USER can be stored in non-volatile memory] Cause the TOE to contain and retain some type of D.USER containing known text strings or byte array in non-volatile memory (e.g., submit a print job when no paper is loaded in the HCD. This will ensure that the job will remain in the HCD for reference in Test 6). Verify that the D.USER data is present on the TOE. For example, it is sufficient to verify that a print job is on the HCD.

Test 2: The evaluator shall perform the following steps to verify that a wipe function can be initiated by an authorized administrator and that the TOE provides feedback about the status of the wipe operation.

1. Initiate a wipe.
2. If the guidance documentation indicates that status information about the wipe operation is provided during the process, the evaluator shall verify that the status information is provided as described.
3. The evaluator shall verify that the indication of completion of the wipe operation is provided as described in the guidance documentation.

All the following tests are dependent on Test 2 being performed.

Test 3: [*Conditional: If a media-specific method or block erase is selected*] Using a debug log or special tooling that the developer shall provide, the evaluator shall verify that the media-specific method

or block erase command is executed.

This test verifies the execution of the media-specific method or block erase command.

Test 4: [*Conditional: If overwrite is selected*] The evaluator shall verify that locations on the storage media available for D.USER and/or D.TSF have been set to the pattern specified in the SFR. This test may require forensic tools to be installed on the HCD, or for the storage media to be moved to a separate system equipped with forensic tools. At minimum, the evaluator shall examine the storage locations specified in the following table for the specified storage types.

Storage Media Type	**Locations Examined**
Magnetic Media	1. First 2049 sectors of the disk/partition 2. 129 sectors before and after the middle of the disk/partition, including the middle sector 3. Last 1025 sectors of the disk/partition 4. 10 separate sectors chosen by the tester at random in the remaining area of the disk/partition 5. If the disk/partition has over 268,435,456 sectors (28-bit Logical Block Addressing (LBA) limit) then sectors 268,435,327 to 268,435,585 sectors of the disk/partition 6. If the disk/partition has over 4,294,967,296 sectors (32-bit LBA limit) then sectors 4,294,967,167 to 4,294,967,425 sectors of the disk/partition 7. Sectors that the tester believes should be tested, if any

Storage Media Type	Locations Examined
Flash-based storage (including wear-leveling media)	1. First 2049 logical sectors of the disk 2. 129 logical sectors before and after the middle of the disk, including the middle logical sector 3. Last 1025 logical sectors of the disk 4. 10 separate logical sectors chosen by the tester at random in the remaining area of the disk 5. If the disk has over 268,435,456 logical sectors (28-bit Logical Block Addressing (LBA) limit) then logical sectors 268,435,327 to 268,435,585 logical sectors of the disk 6. If the disk has over 4,294,967,296 logical sectors (32-bit LBA limit) then logical sectors 4,294,967,167 to 4,294,967,425 logical sectors of the disk 7. Logical sectors that the tester believes should be tested, if any.

Test 5: [Conditional: The case cryptographic erase is selected] The evaluator shall verify that the appropriate key(s) according to the KMD description required by FCS_CKM_EXT.4 has been destroyed in the method described in the Test Assurance Activity for FCS_CKM.4.

Test 6: The evaluator shall verify that known text strings or byte array for D.USER and D.TSF are not found on the non-volatile storage media after wipe has been performed. In some cases, cryptographic erase would make unable to read data from the storage media. In that case, “failed to read data with error” can be considered as “not found”. This test may require special tools to be installed on the TOE, or for the storage media to be moved to a separate system equipped with special tools provided by the TOE developer as necessary. The evaluator shall examine the storage media as specified in the following table for the indicated search patterns. This test is not applicable to protected storage devices (e.g., TPMs) that do not provide direct access to their storage.

Data Type	Search Patterns
D.USER	Known text strings or byte array from Test 1 that was stored on the TOE prior to the wipe being performed. Note: If the TOE does not store D.USER data in non-volatile memory (reference Test 1) the non-existence of this D.USER data does not have to be verified.

Data Type	Search Patterns
D.TSF	1. The text representation of the IP address assigned to the TOE when it was in operation 2. The hexadecimal representation of the IP address assigned to the TOE when it was in operation 3. The text representation of an administrator account that was configured for the TOE when it was in operation.

4.3. Protected Communications

4.3.1. FCS_TLSC_EXT.2 TLS Client support for mutual authentication

4.3.1.1. TSS

4.3.1.1.1. FCS_TLSC_EXT.2.1

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

4.3.1.2. Guidance Documentation

4.3.1.2.1. FCS_TLSC_EXT.2.1

If the TSS indicates that mutual authentication using X.509v3 certificates is used, the evaluator shall verify that the AGD guidance includes instructions for configuring the client-side certificates for TLS mutual authentication.

4.3.1.3. Tests

For all tests in this chapter the TLS server used for testing of the TOE shall be configured to require mutual authentication.

4.3.1.3.1. FCS_TLSC_EXT.2.1

(covered by FCS_TLSC_EXT.1.1 Test 1 and testing for FIA_X.509_EXT.*)

4.3.2. FCS_TLSS_EXT.2 TLS Server support for mutual authentication

4.3.2.1. TSS

4.3.2.1.1. FCS_TLSS_EXT.2.1 and FCS_TLSS_EXT.2.2

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

The evaluator shall verify the TSS describes how the TSF uses certificates to authenticate the TLS client. The evaluator shall verify the TSS describes if the TSF supports any fallback authentication functions (e.g. username/password, challenge response) the TSF uses to authenticate TLS clients that do not present a certificate. If fallback authentication functions are supported, the evaluator shall verify the TSS describes whether the fallback authentication functions can be disabled.

4.3.2.1.2. FCS_TLSS_EXT.2.3

The evaluator shall verify that the TSS describes which types of identifiers are supported during client authentication (e.g. Fully Qualified Domain Name (FQDN)). If FQDNs are supported, the evaluator shall verify that the TSS describes that corresponding identifiers are matched according to RFC6125. For all other types of identifiers, the evaluator shall verify that the TSS describes how these identifiers are parsed from the certificate, what the expected identifiers are and how the parsed identifiers from the certificate are matched against the expected identifiers.

4.3.2.2. Guidance Documentation

4.3.2.2.1. FCS_TLSS_EXT.2.1 and FCS_TLSS_EXT.2.2

If the TSS indicates that mutual authentication using X.509v3 certificates is used, the evaluator shall verify that the AGD guidance includes instructions for configuring the client-side certificates for TLS mutual authentication.

The evaluator shall verify the guidance describes how to configure the TLS client certificate authentication function. If the TSF supports fallback authentication functions, the evaluator shall verify the guidance provides instructions for configuring the fallback authentication functions. If fallback authentication functions can be disabled, the evaluator shall verify the guidance provides instructions for disabling the fallback authentication functions.

4.3.2.2.2. FCS_TLSS_EXT.2.3

The evaluator shall ensure that the AGD guidance describes the configuration of expected identifier(s) for X.509 certificate-based authentication of TLS clients. The evaluator ensures this description includes all types of identifiers described in the TSS and, if claimed, configuration of the TOE to use a directory server.

4.3.2.3. Tests

For all tests in this chapter the TLS client used for testing of the TOE shall support mutual authentication.

4.3.2.3.1. FCS_TLSS_EXT.2.1 and FCS_TLSS_EXT.2.2

Test 1a [conditional]: If the TOE requires or can be configured to require a client certificate, the evaluator shall configure the TOE to require a client certificate and send a Certificate Request to the client. The evaluator shall attempt a connection while sending a certificate_list structure with a length of zero in the Client Certificate message. The evaluator shall verify that the handshake is not finished successfully and no application data flows.

Test 1b [conditional]: If the TOE supports fallback authentication functions and these functions

cannot be disabled, the evaluator shall configure the fallback authentication functions on the TOE and configure the TOE to send a Certificate Request to the client. The evaluator shall attempt a connection while sending a certificate_list structure with a length of zero in the Client Certificate message. The evaluator shall verify the TOE authenticates the connection using the fallback authentication functions as described in the TSS.

Note: Testing the validity of the client certificate is performed as part of X.509 testing.

Test 2 [conditional]: If TLS 1.2 is claimed for the TOE, the evaluator shall configure the server to send a certificate request to the client without the supported_signature_algorithm used by the client's certificate. The evaluator shall attempt a connection using the client certificate and verify that the connection is denied.

Test 3: The aim of this test is to check the response of the server when it receives a client identity certificate that is signed by an impostor CA (either Root CA or intermediate CA). To carry out this test the evaluator shall configure the client to send a client identity certificate with an issuer field that identifies a CA recognised by the TOE as a trusted CA, but where the key used for the signature on the client certificate does not correspond to the CA certificate trusted by the TOE (meaning that the client certificate is invalid because its certification path does not terminate in the claimed CA certificate). The evaluator shall verify that the attempted connection is denied.

Test 4: The evaluator shall configure the client to send a certificate with the Client Authentication purpose in the extendedKeyUsage field and verify that the server accepts the attempted connection. The evaluator shall repeat this test without the Client Authentication purpose and shall verify that the server denies the connection. Ideally, the two certificates should be identical except for the Client Authentication purpose.

Test 5: The evaluator shall perform the following modifications to the traffic:

a. Configure the server to require mutual authentication and then connect to the server with a client configured to send a client certificate that is signed by a Certificate Authority trusted by the TOE. The evaluator shall verify that the server accepts the connection.
b. Configure the server to require mutual authentication and then modify a byte in the signature block of the client's Certificate Verify handshake message (see RFC5246 Sec 7.4.8). The evaluator shall verify that the server rejects the connection.

Note: Testing the validity of the client certificate is performed as part of X.509 testing.

The evaluator shall demonstrate that using an invalid certificate results in the function failing as follows:

Test 6: Using the administrative guidance, the evaluator shall load a CA certificate or certificates needed to validate the presented certificate used to authenticate an external entity and demonstrate that the function succeeds, and a trusted channel can be established.

Test 7: The evaluator shall then change the presented certificate(s) so that validation fails and show that the certificate is not automatically accepted. The evaluator shall repeat this test to cover the selected types of failure defined in the SFR (i.e. the selected ones from failed matching of the reference identifier, failed validation of the certificate path, failed validation of the expiration date, failed determination of the revocation status). The evaluator performs the action indicated in the

SFR selection observing the TSF resulting in the expected state for the trusted channel (e.g. trusted channel was established) covering the types of failure for which an override mechanism is defined.

Test 8 [conditional]: The purpose of this test is to verify that only selected certificate validation failures could be administratively overridden. If any override mechanism is defined for failed certificate validation, the evaluator shall configure a new presented certificate that does not contain a valid entry in one of the mandatory fields or parameters (e.g. inappropriate value in extendedKeyUsage field) but is otherwise valid and signed by a trusted CA. The evaluator shall confirm that the certificate validation fails (i.e. certificate is rejected), and there is no administrative override available to accept such certificate.

4.3.2.3.2. FCS_TLSS_EXT.2.3

The evaluator shall send a client certificate with an identifier that does not match an expected identifier and verify that the server denies the connection.

4.3.3. FCS_DTLSC_EXT.2 DTLS Client support for mutual authentication

4.3.3.1. TSS

4.3.3.1.1. FCS_DTLSC_EXT.2.1

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

4.3.3.1.2. FCS_DTLSC_EXT.2.2

The evaluator shall verify that the TSS describes the actions that take place if a message received from the DTLS Server fails the MAC integrity check.

4.3.3.1.3. FCS_DTLSC_EXT.2.3

The evaluator shall verify that TSS describes how replay is detected and silently discarded for DTLS records that have previously been received and too old to fit in the sliding window.

4.3.3.2. Guidance Documentation

4.3.3.2.1. FCS_DTLSC_EXT.2.1

If the TSS indicates that mutual authentication using X.509v3 certificates is used, the evaluator shall verify that the AGD guidance includes instructions for configuring the client-side certificates for DTLS mutual authentication.

4.3.3.3. Tests

For all tests in this chapter the TLS server used for testing of the TOE shall be configured to require mutual authentication.

4.3.3.3.1. FCS_DTLSC_EXT.2.1

(covered by FCS_DTLSC_EXT.1.1 Test 1 and testing for FIA_X.509_EXT.*)

4.3.3.3.2. FCS_DTLSC_EXT.2.2

Test 1: The evaluator shall establish a DTLS connection. The evaluator will then modify at least one byte in a record message and verify that the Client discards the record or terminates the DTLS session.

4.3.3.3.3. FCS_DTLSC_EXT.2.3

Test 1: The evaluator shall set up a DTLS connection with a DTLS Server. The evaluator shall then capture traffic sent from the DTLS Server to the TOE. The evaluator shall retransmit copies of this traffic to the TOE in order to impersonate the DTLS Server. The evaluator shall observe that the TSF does not take action in response to receiving these packets and that the audit log indicates that the replayed traffic was discarded.

4.3.4. FCS_DTLSS_EXT.2 DTLS Server support for mutual authentication

4.3.4.1. TSS

4.3.4.1.1. FCS_DTLSS_EXT.2.1 and FCS_DTLSS_EXT.2.2

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

The evaluator shall verify the TSS describes how the TSF uses certificates to authenticate the DTLS client. The evaluator shall verify the TSS describes whether the TSF supports any fallback authentication functions (e.g. username/password, challenge response) the TSF uses to authenticate DTLS clients that do not present a certificate. If fallback authentication functions are supported, the evaluator shall verify the TSS describes whether the fallback authentication functions can be disabled.

4.3.4.1.2. FCS_DTLSS_EXT.2.3

The evaluator shall verify that the TSS describes which types of identifiers are supported for during client authentication (e.g. Fully Qualified Domain Name (FQDN)). If FQDNs are supported, the evaluator shall verify that the TSS describes that corresponding identifiers are matched according to RFC6125. For all other types of identifiers, the evaluator shall verify that the TSS describes how these identifiers are parsed from the certificate, what the expected identifiers are and how the parsed identifiers from the certificate are matched against the expected identifiers.

4.3.4.2. Guidance Documentation

4.3.4.2.1. FCS_DTLSS_EXT.2.1 and FCS_DTLSS_EXT.2.2

If the TSS indicates that mutual authentication using X.509v3 certificates is used, the evaluator shall verify that the AGD guidance includes instructions for configuring the client-side certificates for DTLS mutual authentication.

The evaluator shall verify the guidance describes how to configure the DTLS client certificate authentication function. If the TSF supports fallback authentication functions, the evaluator shall verify the guidance provides instructions for configuring the fallback authentication functions. If fallback authentication functions can be disabled, the evaluator shall verify the guidance provides

instructions for disabling the fallback authentication functions.

4.3.4.2.2. FCS_DTLSS_EXT.2.3

The evaluator shall ensure that the AGD guidance describes the configuration of expected identifier(s) for X.509 certificate-based authentication of DTLS clients. The evaluator ensures this description includes all types of identifiers described in the TSS and, if claimed, configuration of the TOE to use a directory server.

4.3.4.3. Tests

For all tests in this chapter the TLS client used for testing of the TOE shall support mutual authentication.

4.3.4.3.1. FCS_DTLSS_EXT.2.1 and FCS_DTLSS_EXT.2.2

Test 1a [conditional]: If the TOE requires or can be configured to require a client certificate, the evaluator shall configure the TOE to require a client certificate and send a Certificate Request to the client. The evaluator shall attempt a connection while sending a certificate_list structure with a length of zero in the Client Certificate message. The evaluator shall verify that the handshake is not finished successfully and no application data flows.

Test 1b [conditional]: If the TOE supports fallback authentication functions and these functions cannot be disabled, the evaluator shall configure the fallback authentication functions on the TOE and configure the TOE to send a Certificate Request to the client. The evaluator shall attempt a connection while sending a certificate_list structure with a length of zero in the Client Certificate message. The evaluator shall verify the TOE authenticates the connection using the fallback authentication functions as described in the TSS.

Note: Testing the validity of the client certificate is performed as part of X.509 testing.

Test 2 [conditional]: If DTLS 1.2 is claimed for the TOE, the evaluator shall configure the server to send a certificate request to the client without the supported_signature_algorithm used by the client's certificate. The evaluator shall attempt a connection using the client certificate and verify that the connection is denied.

Test 3: The aim of this test is to check the response of the server when it receives a client identity certificate that is signed by an impostor CA (either Root CA or intermediate CA). To carry out this test the evaluator shall configure the client to send a client identity certificate with an issuer field that identifies a CA recognised by the TOE as a trusted CA, but where the key used for the signature on the client certificate does not correspond to the CA certificate trusted by the TOE (meaning that the client certificate is invalid because its certification path does not terminate in the claimed CA certificate). The evaluator shall verify that the attempted connection is denied.

Test 4: The evaluator shall configure the client to send a certificate with the Client Authentication purpose in the extendedKeyUsage field and verify that the server accepts the attempted connection. The evaluator shall repeat this test without the Client Authentication purpose and shall verify that the server denies the connection. Ideally, the two certificates should be identical except for the Client Authentication purpose.

Test 5: The evaluator shall perform the following modifications to the traffic:

a. Configure the server to require mutual authentication and then connect to the server with a client configured to send a client certificate that is signed by a Certificate Authority trusted by the TOE. The evaluator shall verify that the server accepts the connection.
b. Configure the server to require mutual authentication and then modify a byte in the signature block of the client's Certificate Verify handshake message (see RFC5246 Sec 7.4.8). The evaluator shall verify that the server rejects the connection.

Note: Testing the validity of the client certificate is performed as part of X.509 testing.

The evaluator shall demonstrate that using an invalid certificate results in the function failing as follows:

Test 6: Using the administrative guidance, the evaluator shall load a CA certificate or certificates needed to validate the presented certificate used to authenticate an external entity and demonstrate that the function succeeds, and a trusted channel can be established.

Test 7: The evaluator shall then change the presented certificate(s) so that validation fails and show that the certificate is not automatically accepted. The evaluator shall repeat this test to cover the selected types of failure defined in the SFR (i.e. the selected ones from failed matching of the reference identifier, failed validation of the certificate path, failed validation of the expiration date, failed determination of the revocation status). The evaluator performs the action indicated in the SFR selection observing the TSF resulting in the expected state for the trusted channel (e.g. trusted channel was established) covering the types of failure for which an override mechanism is defined.

Test 8 [conditional]: The purpose of this test is to verify that only selected certificate validation failures could be administratively overridden. If any override mechanism is defined for failed certificate validation, the evaluator shall configure a new presented certificate that does not contain a valid entry in one of the mandatory fields or parameters (e.g. inappropriate value in extendedKeyUsage field) but is otherwise valid and signed by a trusted CA. The evaluator shall confirm that the certificate validation fails (i.e. certificate is rejected), and there is no administrative override available to accept such certificate.

4.3.4.3.2. FCS_DTLSS_EXT.2.3

The evaluator shall send a client certificate with an identifier that does not match an expected identifier and verify that the server denies the connection.

Chapter 5. Evaluation Activities for Selection-Based Requirements

5.1. Confidential Data on Nonvolatile Storage Devices

5.1.1. FCS_COP.1/StorageEncryption Cryptographic operation (Data Encryption/Decryption)

5.1.1.1. TSS

The evaluator shall verify the TSS includes a description of the key size used for encryption and the mode used for encryption.

5.1.1.2. Guidance Documentation

If multiple encryption modes are supported, the evaluator examines the guidance documentation to determine that the method of choosing a specific mode/key size by the end user is described.

5.1.1.3. Tests

The following tests are conditional based upon the selections made in the SFR.

Note: The tests detailed below are not required to be performed for cryptographic functions implemented in the Root of Trust for Secure Boot (FPT_SBT_EXT.1).

AES-CBC Tests

For the AES-CBC tests described below, the plaintext, ciphertext, and IV values shall consist of 128-bit blocks. To determine correctness, the evaluator shall compare the resulting values to those obtained by submitting the same inputs to a known-good implementation.

These tests are intended to be equivalent to those described in NIST's AES Algorithm Validation Suite (AESAVS) (http://csrc.nist.gov/groups/STM/cavp/documents/aes/AESAVS.pdf). It is not recommended that evaluators use values obtained from static sources such as the example NIST's AES Known Answer Test Values from the AESAVS document, or use values not generated expressly to exercise the AES-CBC implementation.

AES-CBC Known Answer Tests

KAT-1 (GFSBox): To test the encrypt functionality of AES-CBC, the evaluator shall supply a set of five different plaintext values for each selected key size 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.

To test the decrypt functionality of AES-CBC, the evaluator shall supply a set of five different ciphertext values for each selected key size and obtain the plaintext value that results from AES-CBC decryption of the given ciphertext using a key value of all zeros and an IV of all zeros.

KAT-2 (KeySBox): To test the encrypt functionality of AES-CBC, the evaluator shall supply a set of

five different key values for each selected key size 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.

To test the decrypt functionality of AES-CBC, the evaluator shall supply a set of five different key values for each selected key size and obtain the plaintext that results from AES-CBC decryption of an all-zeros ciphertext using the given key and an IV of all zeros.

KAT-3 (Variable Key): To test the encrypt functionality of AES-CBC, the evaluator shall supply a set of keys for each selected key size (as described below) and obtain the ciphertext value that results from AES encryption of an all-zeros plaintext using each key and an IV of all zeros.

Key i in each set shall have the leftmost i bits set to ones and the remaining bits to zeros, for values of i from 1 to the key size. The keys and corresponding ciphertext are listed in AESAVS, Appendix E.

To test the decrypt functionality of AES-CBC, the evaluator shall use the same keys as above to decrypt the ciphertext results from above. Each decryption should result in an all-zeros plaintext.

KAT-4 (Variable Text): To test the encrypt functionality of AES-CBC, for each selected key size, the evaluator shall supply a set of 128-bit plaintext values (as described below) and obtain the ciphertext values that result from AES-CBC encryption of each plaintext value using a key of each size and IV consisting of all zeros.

Plaintext value i shall have the leftmost i bits set to ones and the remaining bits set to zeros, for values of i from 1 to 128. The plaintext values are listed in AESAVS, Appendix D.

To test the decrypt functionality of AES-CBC, for each selected key size, use the plaintext values from above as ciphertext input, and AES-CBC decrypt each ciphertext value using key of each size consisting of all zeros and an IV of all zeros.

AES-CBC Multi-Block Message Test

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

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

AES-CBC Monte Carlo Tests

The evaluator shall test the encrypt functionality for each selected key size using 100 3-tuples of pseudo-random values for plaintext, IVs, and keys.

The evaluator shall supply a single 3-tuple of pseudo-random values for each selected key size. This 3-tuple of plaintext, IV, and key is provided as input to the below algorithm to generate the remaining 99 3-tuples, and to run each 3-tuple through 1000 iterations of AES-CBC encryption.

```
# Input: PT, IV, Key
Key[0] = Key
IV[0] = IV
PT[0] = PT
for i = 0 to 99 {
    Output Key[i], IV[i], PT[0]
    for j = 0 to 999 {
        if (j == 0) {
            CT[j] = AES-CBC-Encrypt(Key[i], IV[i], PT[j])
            PT[j+1] = IV[i]
        } else {
            CT[j] = AES-CBC-Encrypt(Key[i], PT[j])
            PT[j+1] = CT[j-1]
        }
    }
    Output CT[j]
    If (KeySize == 128) Key[i+1] = Key[i] xor CT[j]
    If (KeySize == 192) Key[i+1] = Key[i] xor (last 64 bits of CT[j-1] || CT[j])
    If (KeySize == 256) Key[i+1] = Key[i] xor ((CT[j-1] | CT[j])
    IV[i+1] = CT[j]
    PT[0] = CT[j-1]
}
```

The ciphertext computed in the 1000th iteration (CT[999]) is the result for each of the 100 3-tuples for each selected key size. This result shall be compared to the result of running 1000 iterations with the same values using a known good implementation.

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

AES-GCM Test

These tests are intended to be equivalent to those described in the NIST document, "The Galois/Counter Mode (GCM) and GMAC Validation System (GCMVS) with the Addition of XPN Validation Testing," rev. 15 Jun 2016, section 6.2, found at http://csrc.nist.gov/groups/STM/cavp/documents/mac/gcmvs.pdf.

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

The evaluator shall test the authenticated encryption functionality of AES-GCM by supplying 15 sets of Key, Plaintext, AAD, IV, and Tag data for every combination of the following parameters as selected in the ST and supported by the implementation under test:

- **Key size in bits:** Each selected and supported key size (e.g., 128, 192, or 256 bits).
- **Plaintext length in bits:** Up to four values for plaintext length: Two values that are non-zero integer multiples of 128, if supported. And two values that are non-multiples of 128, if supported.

- **AAD length in bits:** Up to five values for AAD length: Zero-length, if supported. Two values that are non-zero integer multiples of 128, if supported. And two values that are integer non-multiples of 128, if supported.
- **IV length in bits:** Up to three values for IV length: 96 bits. Minimum and maximum supported lengths, if different.
- **MAC length in bits:** Each supported length (e.g., 128, 120, 112, 104, 96).

To determine correctness, the evaluator shall compare the resulting values to those obtained by submitting the same inputs to a known-good implementation.

The evaluator shall test the authenticated decrypt functionality of AES-GCM by supplying 15 Ciphertext-Tag pairs for every combination of the above parameters, replacing Plaintext length with Ciphertext length. For each parameter combination the evaluator shall introduce an error into either the Ciphertext or the Tag such that approximately half of the cases are correct and half the cases contain errors. To determine correctness, the evaluator shall compare the resulting pass/fail status and Plaintext values to the results obtained by submitting the same inputs to a knowngood implementation.

XTS-AES Test

These tests are intended to be equivalent to those described in the NIST document, "The XTS-AES Validation System (XTSVS)," updated 5 Sept 2013, found at http://csrc.nist.gov/groups/STM/cavp/documents/aes/XTSVS.pdf

It is not recommended that evaluators use values obtained from static sources such as the XTS-AES test vectors at http://csrc.nist.gov/groups/STM/cavp/documents/aes/XTSTestVectors.zip or use values not generated expressly to exercise the XTS-AES implementation.

The evaluator shall generate test values as follows:

For each supported key size (256 bit (for AES-128) and 512 bit (for AES-256) keys), the evaluator shall provide up to five data lengths:

- Two data lengths divisible by the 128-bit block size, If data unit lengths of complete block sizes are supported.
- Two data lengths not divisible by the 128-bit block size, if data unit lengths of partial block sizes are supported.
- The largest data length supported by the implementation, or 2^16 (65536), whichever is larger.

The evaluator shall specify whether the implementation supports tweak values of 128-bit hexadecimal strings or a data unit sequence numbers, or both.

For each combination of key size and data length, the evaluator shall provide 100 sets of input data and obtain the ciphertext that results from XTS-AES encryption. If both kinds of tweak values are supported then each type of tweak value shall be used in half of every 100 sets of input data, for all combinations of key size and data length. The evaluator shall verify that the resulting ciphertext matches the results from submitting the same inputs to a known-good implementation of XTS-AES.

The evaluator shall test the decrypt functionality of XTS-AES using the same test as for encrypt,

replacing plaintext values with ciphertext values and XTS-AES encrypt with XTS- AES decrypt.

The evaluator shall check that the full length keys are created by methods that ensure that the two halves are different and independent.

SEED, HIGHT, LEA Test

The evaluator shall refer to Section 2.2.5. “FCS_COP.1/DataEncryption Cryptographic Operation (Data Encryption/Decryption)” for the required following testing.

- SEED-CBC Tests
- SEED-CFB Tests
- SEED-OFB Tests
- SEED-CTR Tests
- SEED-CCM Tests
- SEED-GCM Tests
- HIGHT-CBC Tests
- HIGHT-CFB Tests
- HIGHT-OFB Tests
- HIGHT-CTR Tests
- LEA-CBC Tests
- LEA-CFB Tests
- LEA-OFB Tests
- LEA-CTR Tests
- LEA-CCM Tests
- LEA-GCM Tests

5.1.2. FCS_COP.1/KeyWrap Cryptographic operation (Key Wrapping)

5.1.2.1. TSS

The evaluator shall verify the TSS includes a description of the key wrap function(s) and shall verify the key wrap uses an approved key wrap algorithm according to the appropriate specification.

If “SEED-GCM”, “SEED-CCM”, “LEA-GCM”, or “LEA-CCM” is selected, the evaluator shall examine the TOE summary specification to confirm that it describes how the IV is generated and that the same IV is never reused to encrypt different plaintext pairs under the same key. Moreover, in the case of GCM, the evaluator shall confirm that, at each invocation of GCM, the length of the plaintext is at most (2^32)-2 blocks.

5.1.2.2. Guidance Documentation

The evaluator shall check the operational user guidance to confirm that the instructions for establishing the evaluated configuration use only those key wrap function(s) selected in the ST. If

multiple key access modes are supported, the evaluator shall examine the operational user guidance to determine that the method of choosing a specific mode/key size by the end user is described.

5.1.2.3. KMD

The evaluator shall review the KMD to ensure that all keys are wrapped using the approved method and a description of when the key wrapping occurs.

5.1.2.4. Tests

The evaluator shall include test cases of FCS_COP.1/KeyWrap to the test subset. Note that FCS_COP.1/KeyWrap may be not mapped to the specific interface(s) after evaluator's analysis during ADV_FSP.1.

The evaluator shall produce test documentation for test cases of FCS_COP.1/KeyWrap. If there is no explicit external interface(s) mapped to FCS_COP.1/KeyWrap, the evaluator shall employ an alternative test approach (refer to CEM, section 15.2.2.).

The evaluator shall perform the following tests or witness respective tests executed by the developer if technically possible, otherwise an analysis of the implementation representation has to be performed.

Preconditions for testing

1. Specification of wrapping keys as input parameter to the function to be tested
2. Specification of further required input parameters if required
3. Specification of keys to be wrapped (plaintext, as function's argument)
4. Direct access to wrapped key (ciphertext), e.g. in the non-volatile memory

AES-KW, AES-KWP, AES-GCM, AES-CCM

The evaluator shall ensure the wrapped key is wrapped as specified in this SFR using reference implementation of wrapping in accordance with AES in the modes and key size specified in this SFR. This reference implementation of wrapping algorithm may be a tool or program provided by the evaluator or the developer, this implementation is dependent on the KMD description provided by the developer.

SEED-KW [SP 800-38F, sec. 6.2]

The evaluator shall test the authenticated-encryption functionality of SEED-KW (Key Wrap) for each combination of the following input parameters:

1. Supported key lengths selected in the ST (128 bits)
2. Five plaintext lengths:
    a. Two lengths that are non-zero multiples of 128 bits (two semi-block lengths)
    b. Two lengths that are odd multiples of the semi-block length (64 bits)
    c. The largest supported plaintext length less than or equal to 4096 bits.

For each set of the above parameters the evaluator shall generate a set of 100 key and plaintext pairs and obtain the ciphertext that results from SEED-KW authenticated encryption. To determine correctness, the evaluator shall compare the results with those obtained from the SEED-KW authenticated-encryption function of a known good implementation.

The evaluator shall test the authenticated-decryption functionality of SEED-KW using the same test as for authenticated-encryption, replacing plaintext values with ciphertext values and SEED-KW authenticated-encryption (KW-AE) with SEED-KW authenticated-decryption (KW-AD). For the authenticated-decryption test, 20 out of the 100 trials per plaintext length must have ciphertext values that are not authentic; that is, they fail authentication.

Additionally, the evaluator shall perform the following negative tests:

Test 1 (invalid plaintext length)

Determine the valid plaintext lengths of the implementation from the TOE specification. Verify that the implementation of KW-AE in the TOE rejects plaintexts of invalid length by testing plaintext of the following lengths: 1) plaintext length greater than 64 semi-blocks, 2) plaintext bit-length not divisible by 64, 3) plaintext with length 0, and 4) plaintext with one semi-block.

Test 2 (invalid ciphertext length)

Determine the valid ciphertext lengths of the implementation from the TOE specification. Verify that the implementation of KW-AD in the TOE rejects ciphertexts of invalid length by testing ciphertext of the following lengths: 1) ciphertext with length greater than 65 semi-blocks, 2) ciphertext with bit-length not divisible by 64, 3) ciphertext with length 0, 4) ciphertext with length of one semi-block, and 5) ciphertext with length of two semi-blocks.

Test 3 (invalid ICV1)

Test that the implementation detects invalid ICV1 values by encrypting any plaintext value eight times using a different value for ICV1 each time as follows: Start with a base ICV1 of 0xA6A6A6A6A6A6A6A6. For each of the eight tests change a different byte to a different value, so that each of the eight bytes is changed once. Verify that the implementation of KW-AD in the TOE outputs FAIL for each test.

SEED-KWP [SP 800-38F, sec. 6.3]

The tests below are derived from “The Key Wrap Validation System (KWVS), Updated: June 20, 2014” from the National Institute of Standards and Technology.

The evaluator shall test the authenticated-encryption functionality of SEED-KWP (Key Wrap with Padding) for each combination of the following input parameters:

1. Supported key lengths selected in the ST (128 bits)
2. Five plaintext lengths:
    a. Four lengths that are multiple of 8 bits
    b. The largest supported length less than or equal to 4096 bits.

The evaluator shall test the authenticated-decryption (KWP-AD) functionality of SEED-KWP using

the same test as for authenticated-encryption, replacing plaintext values with ciphertext values and SEED-KWP authenticated-encryption with SEED-KWP authenticated-decryption. For the authenticated-decryption test, 20 out of the 100 trials per plaintext length must have ciphertext values that are not authentic; that is, they fail authentication.

Additionally, the evaluator shall perform the following negative tests:

Test 1 (invalid plaintext length)

Determine the valid plaintext lengths of the implementation from the TOE specification. Verify that the implementation of KWP-AE in the TOE rejects plaintexts of invalid length by testing plaintext of the following lengths: 1) plaintext length greater than 64 semi-blocks, 2) plaintext bit-length not divisible by 8, 3) plaintext with length 0.

Test 2 (invalid ciphertext length)

Determine the valid ciphertext lengths of the implementation from the TOE specification. Verify that the implementation of KWP-AD in the TOE rejects ciphertexts of invalid length by testing ciphertext of the following lengths: 1) ciphertext with length greater than 65 semi-blocks, 2) ciphertext with bit-length not divisible by 64, 3) ciphertext with length 0, 4) ciphertext with length of one semi-block.

Test 3 (invalid ICV2)

Test that the implementation detects invalid ICV2 values by encrypting any plaintext value eight times using a different value for ICV2 each time as follows: Start with a base ICV2 of 0xA65959A6. For each of the four tests change a different byte to a different value, so that each of the four bytes is changed once. Verify that the implementation of KWP-AD in the TOE outputs FAIL for each test.

Test 4 (invalid padding length)

Generate one ciphertext using algorithm KWP-AE with substring $[len(P)/8]_{32}$ of S replaced by each of the following 32-bit values, where $len(P)$ is the length of P in bits and $[]_{32}$ denotes representation of an integer in 32 bits:

- $[0]_{32}$
- $[len(P)/8 \square 8]_{32}$
- $[len(P)/8 + 8]_{32}$
- $[513]_{32}$

Verify that the implementation of KWP-AD in the TOE outputs FAIL on those inputs.

Test 5 (invalid padding bits)

If the implementation supports plaintext of length not a multiple of 64-bits, then

```
for each PAD length [1..7]
    for each byte in PAD
        set a zero PAD value;
        replace current byte by a non-zero value and use the resulting  plaintext as
input to algorithm KWP-AE to generate  ciphertexts;

        Verify that the implementation of KWP-AD in the TOE outputs FAIL on this
input.
```

SEED-GCM [ISO 19772, clause 11]

Refer to FCS_COP.1/StorageEncryption for the required SEED-GCM testing. Each distinct SEED-GCM implementation shall be tested separately.

SEED-CCM [ISO 19772, clause 8]

Refer to FCS_COP.1/StorageEncryption for the required SEED-CCM testing. Each distinct SEED-CCM implementation shall be tested separately.

LEA-KW [SP 800-38F, sec. 6.2]

Refer to [SEED-KW] for the required LEA-KW testing. The evaluator shall test the authenticated encryption functionality using the same test as SEED-KW Tests, replacing SEED algorithm with LEA algorithm as PRF. In the LEA-KW testing, supported key lengths should be selected in the ST (e.g., 128bits, 192bits, 256bits).

LEA-KWP [SP 800-38F, sec. 6.3]

Refer to [SEED-KWP] for the required LEA-KWP testing. The evaluator shall test the authenticated encryption functionality using the same test as SEED-KWP Tests, replacing SEED algorithm with LEA algorithm as PRF. In the LEA-KWP testing, supported key lengths should be selected in the ST (e.g., 128bits, 192bits, 256bits).

LEA-GCM [ISO 19772, clause 11]

Refer to FCS_COP.1/StorageEncryption for the required LEA-GCM testing. Each distinct LEA-GCM implementation shall be tested separately.

LEA-CCM [ISO 19772, clause 8]

Refer to FCS_COP.1/StorageEncryption for the required LEA-CCM testing. Each distinct LEA-CCM implementation shall be tested separately.

5.1.3. FCS_COP.1/KeyEnc Cryptographic operation (Key Encryption)

5.1.3.1. TSS

The evaluator shall verify the TSS includes a description of the key encryption function(s) and shall verify the key encryption uses an approved algorithm according to the appropriate specification.

5.1.3.2. KMD

The evaluator shall review the KMD to ensure that all keys are encrypted using the approved method and a description of when the key encryption occurs is provided.

5.1.3.3. Tests

The evaluator shall use tests in FCS_COP.1/StorageEncryption to verify encryption.

5.1.4. FCS_COP.1/KeyTransport Cryptographic operation (Key Transport)

5.1.4.1. TSS

The evaluator shall verify the TSS provides a high level description of the RSA scheme and the cryptographic key size that is being used, and that the asymmetric algorithm being used for key transport is RSA. If more than one scheme/key size are allowed, then the evaluator shall make sure and test all combinations of scheme and key size. There may be more than one key size to specify – an RSA modulus size (and/or encryption exponent size), an AES key size, hash sizes, MAC key/MAC tag size.

If the KTS-OAEP scheme was selected, the evaluator shall verify that the TSS identifies the hash function, the mask generating function, the random bit generator, the encryption primitive and decryption primitive.

If the KTS-KEM-KWS scheme was selected, the evaluator shall verify that the TSS identifies the key derivation method, the AES-based key wrapping method, the secret value encapsulation technique, and the random number generator.

5.1.4.2. Guidance Documentation

There are no AGD evaluation activities for this SFR.

5.1.4.3. KMD

There are no KMD evaluation activities for this SFR.

5.1.4.4. Tests

For each supported key transport schema, the evaluator shall initiate at least 25 sessions that require key transport with an independently developed remote instance of a key transport entity, using known RSA key-pairs. The evaluator shall observe traffic passed from the sender-side and to the receiver-side of the TOE, and shall perform the following tests, specific to which key transport scheme was employed.

If the KTS-OAEP scheme was selected, the evaluator shall perform the following tests:

1. The evaluator shall inspect each cipher text, C, produced by the RSA-OAEP encryption operation of the TOE and make sure it is the correct length, either 256 or 384 bytes depending on RSA key size. The evaluator shall also feed into the TOE's RSA-OAEP decryption operation some cipher texts that are the wrong length and verify that the erroneous input is detected and that the decryption operation exits with an error code.

2. The evaluator shall convert each cipher text, C, produced by the RSA-OAEP encryption operation of the TOE to the correct cipher text integer, c, and use the decryption primitive to compute em = RSADP((n,d),c) and convert em to the encoded message EM. The evaluator shall then check that the first byte of EM is 0x00. The evaluator shall also feed into the TOE's RSA-OAEP decryption operation some cipher texts where the first byte of EM was set to a value other than 0x00, and verify that the erroneous input is detected and that the decryption operation exits with an error code.
3. The evaluator shall decrypt each cipher text, C, produced by the RSA-OAEP encryption operation of the TOE using RSADP, and perform the OAEP decoding operation (described in SP800-56B section 7.2.2.4) to recover HA' || X. For each HA', the evaluator shall take the corresponding A and the specified hash algorithm and verify that HA' = Hash(A). The evaluator shall also force the TOE to perform some RSA-OAEP decryptions where the A value is passed incorrectly, and the evaluator shall verify that an error is detected.
4. The evaluator shall check the format of the 'X' string recovered in OAEP. Test.3 to ensure that the format is of the form PS || 01 || K, where PS consists of zero or more consecutive 0x00 bytes and K is the transported keying material. The evaluator shall also feed into the TOE's RSA-OAEP decryption operation some cipher texts for which the resulting 'X' strings do not have the correct format (i.e., the leftmost non-zero byte is not 0x01). These incorrectly formatted 'X' variables shall be detected by the RSA-OAEP decrypt function.
5. The evaluator shall trigger all detectable decryption errors and validate that the returned error codes are the same and that no information is given back to the sender on which type of error occurred. The evaluator shall also validate that no intermediate results from the TOE's receiver-side operations are revealed to the sender.

If the KTS-KEM-KWS scheme was selected, the evaluator shall perform the following tests:

1. The evaluator shall inspect each cipher text, C, produced by RSA-KEM-KWS encryption operation of the TOE and make sure the length (in bytes) of the cipher text, cLen, is greater than nLen (the length, in bytes, of the modulus of the RSA public key) and that cLen - nLen is consistent with the byte lengths supported by the key wrapping algorithm. The evaluator shall feed into the RSA-KEM-KWS decryption operation a cipher text of unsupported length and verify that an error is detected and that the decryption process stops.
2. The evaluator shall separate the cipher text, C, produced by the sender-side of the TOE into its C0 and C1 components and use the RSA decryption primitive to recover the secret value, Z, from C0. The evaluator shall check that the unsigned integer represented by Z is greater than 1 and less than n-1, where n is the modulus of the RSA public key. The evaluator shall construct examples where the cipher text is created with a secret value Z = 1 and make sure the RSA-KEM-KWS decryption process detects the error. Similarly, the evaluator shall construct examples where the cipher text is created with a secret value Z = n – 1 and make sure the RSA-KEM-KWS decryption process detects the error.
3. The evaluator shall attempt to successfully recover the secret value Z, derive the key wrapping key, KWK, and unwrap the KWA-cipher text following the RSA-KEM-KWS decryption process given in SP800-56B section 7.2.3.4. If the key-wrapping algorithm is AES-CCM, the evaluator shall verify that the value of any (unwrapped) associated data, A, that was passed with the wrapped keying material is correct. The evaluator shall feed into the TOE's RSA-KEM-KWS decryption operation examples of incorrect cipher text and verify that a decryption error is detected. If the

key-wrapping algorithm is AES-CCM, the evaluator shall attempt at least one decryption where the wrong value of A is given to the RSA-KEM-KWS decryption operation and verify that a decryption error is detected. Similarly, if the key-wrapping algorithm is AES-CCM, the evaluator shall attempt at least one decryption where the wrong nonce is given to the RSA-KEM-KWS decryption operation and verify that a decryption error is detected.

4. The evaluator shall trigger all detectable decryption errors and validate that the resulting error codes are the same and that no information is given back to the sender on which type of error occurred. The evaluator shall also validate that no intermediate results from the TOE's receiver-side operations (in particular, no Z values) are revealed to the sender.

5.1.5. FCS_SMC_EXT.1 Submask Combining

5.1.5.1. TSS

If keys are XORed together to form an intermediate key, the TSS section shall identify how this is performed (e.g., if there are ordering requirements, checks performed, etc.). The evaluator shall also confirm that the TSS describes how the length of the output produced is at least the same as that of the DEK.

5.1.5.2. KMD

The evaluator shall review the KMD to ensure that an approved combination is used and does not result in the weakening or exposure of key material.

5.1.5.3. Tests

(conditional): If there is more than one authorization factor, the evaluator shall ensure that failure to supply a required authorization factor does not result in access to the encrypted data.

5.2. Protected Communications

5.2.1. FCS_IPSEC_EXT.1 IPsec selected

5.2.1.1. TSS

5.2.1.1.1. FCS_IPSEC_EXT.1.1

The evaluator shall examine the TSS and determine that it describes what takes place when a packet is processed by the TOE, e.g., the algorithm used to process the packet. The TSS describes how the SPD is implemented and the rules for processing both inbound and outbound packets in terms of the IPsec policy. The TSS describes the rules that are available and the resulting actions available after matching a rule. The TSS describes how those rules and actions form the SPD in terms of the BYPASS (e.g., no encryption), DISCARD (e.g., drop the packet), and PROTECT (e.g., encrypt the packet) actions defined in RFC 4301.

As noted in section 4.4.1 of RFC 4301, the processing of entries in the SPD is non-trivial and the evaluator shall determine that the description in the TSS is sufficient to determine which rules will be applied given the rule structure implemented by the TOE. For example, if the TOE allows

specification of ranges, conditional rules, etc., the evaluator shall determine that the description of rule processing (for both inbound and outbound packets) is sufficient to determine the action that will be applied, especially in the case where two different rules may apply. This description shall cover both the initial packets (that is, no SA is established on the interface or for that particular packet) as well as packets that are part of an established SA.

5.2.1.1.2. FCS_IPSEC_EXT.1.3

The evaluator checks the TSS to ensure it states that the VPN can be established to operate in transport mode and/or tunnel mode (as identified in FCS_IPSEC_EXT.1.3).

5.2.1.1.3. FCS_IPSEC_EXT.1.4

The evaluator shall examine the TSS to verify that the selected algorithms are implemented. In addition, the evaluator ensures that the SHA-based HMAC algorithm conforms to the algorithms specified in FCS_COP.1/KeyedHash Cryptographic Operation (for keyed-hash message authentication) and if the SHA-based HMAC function truncated output is utilized it must also be described.

5.2.1.1.4. FCS_IPSEC_EXT.1.5

The evaluator shall examine the TSS to verify that IKEv1 and/or IKEv2 are implemented.

For IKEv1 implementations, the evaluator shall examine the TSS to ensure that, in the description of the IPsec protocol, it states that aggressive mode is not used for IKEv1 Phase 1 exchanges, and that only main mode is used. It may be that this is a configurable option.

5.2.1.1.5. FCS_IPSEC_EXT.1.6

The evaluator shall ensure the TSS identifies the algorithms used for encrypting the IKEv1 and/or IKEv2 payload, and that the algorithms chosen in the selection of the requirement are included in the TSS discussion.

5.2.1.1.6. FCS_IPSEC_EXT.1.7

The evaluator shall ensure the TSS identifies the lifetime configuration method used for limiting the IKEv1 Phase 1 SA lifetime and/or the IKEv2 SA lifetime. The evaluator shall verify that the selection made here corresponds to the selection in FCS_IPSEC_EXT.1.5.

5.2.1.1.7. FCS_IPSEC_EXT.1.8

The evaluator shall ensure the TSS identifies the lifetime configuration method used for limiting the IKEv1 Phase 2 SA lifetime and/or the IKEv2 Child SA lifetime. The evaluator shall verify that the selection made here corresponds to the selection in FCS_IPSEC_EXT.1.5.

5.2.1.1.8. FCS_IPSEC_EXT.1.9

The evaluator shall check to ensure that, for each DH group supported, the TSS describes the process for generating "x". The evaluator shall verify that the TSS indicates that the random number generated that meets the requirements in this cPP is used, and that the length of "x" meets the stipulations in the requirement.

5.2.1.1.9. FCS_IPSEC_EXT.1.10

If the first selection is chosen, the evaluator shall check to ensure that, for each DH group supported, the TSS describes the process for generating each nonce. The evaluator shall verify that the TSS indicates that the random number generated that meets the requirements in this cPP is used, and that the length of the nonces meet the stipulations in the requirement.

If the second selection is chosen, the evaluator shall check to ensure that, for each PRF hash supported, the TSS describes the process for generating each nonce. The evaluator shall verify that the TSS indicates that the random number generated that meets the requirements in this cPP is used, and that the length of the nonces meet the stipulations in the requirement.

5.2.1.1.10. FCS_IPSEC_EXT.1.11

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

5.2.1.1.11. FCS_IPSEC_EXT.1.12

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

5.2.1.1.12. FCS_IPSEC_EXT.1.13

The evaluator ensures that the TSS identifies RSA and/or ECDSA as being used to perform peer authentication. The description must be consistent with the algorithms as specified in FCS_COP.1/SigGen Cryptographic Operation (for cryptographic signature).

If pre-shared keys are chosen in the selection, the evaluator shall check to ensure that the TSS describes how pre-shared keys are established and used in authentication of IPsec connections. The description in the TSS shall also indicate how pre-shared key establishment is accomplished for TOEs that can generate a pre-shared key as well as TOEs that simply use a pre-shared key.

5.2.1.1.13. FCS_IPSEC_EXT.1.14

The evaluator shall ensure that the TSS describes how the TOE compares the peer's presented identifier to the reference identifier. This description shall include which field(s) of the certificate are used as the presented identifier (DN, Common Name, or SAN). If the TOE simultaneously supports the same identifier type in the CN and SAN, the TSS shall describe how the TOE prioritizes the comparisons (e.g. the result of comparison if CN matches but SAN does not). If the location (e.g. CN or SAN) of non-DN identifier types must explicitly be configured as part of the reference identifier, the TSS shall state this. If the ST author assigned an additional identifier type, the TSS description shall also include a description of that type and the method by which that type is compared to the peer's presented certificate, including what field(s) are compared and which fields take precedence in the comparison.

5.2.1.2. Guidance Documentation

5.2.1.2.1. FCS_IPSEC_EXT.1.1

The evaluator shall examine the guidance documentation to verify it instructs the Administrator how to construct entries into the SPD that specify a rule for processing a packet. The description includes all three cases – a rule that ensures packets are encrypted/decrypted, dropped, and flow through the TOE without being encrypted. The evaluator shall determine that the description in the guidance documentation is consistent with the description in the TSS, and that the level of detail in the guidance documentation is sufficient to allow the administrator to set up the SPD in an unambiguous fashion. This includes a discussion of how ordering of rules impacts the processing of an IP packet.

5.2.1.2.2. FCS_IPSEC_EXT.1.3

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

5.2.1.2.3. FCS_IPSEC_EXT.1.4

The evaluator checks the guidance documentation to ensure it provides instructions on how to configure the TOE to use the algorithms selected.

5.2.1.2.4. FCS_IPSEC_EXT.1.5

The evaluator shall check the guidance documentation to ensure it instructs the administrator how to configure the TOE to use IKEv1 and/or IKEv2 (as selected), and how to configure the TOE to perform NAT traversal (if selected).

If the IKEv1 Phase 1 mode requires configuration of the TOE prior to its operation, the evaluator shall check the guidance documentation to ensure that instructions for this configuration are contained within that guidance.

5.2.1.2.5. FCS_IPSEC_EXT.1.6

The evaluator ensures that the guidance documentation describes the configuration of all selected algorithms in the requirement.

5.2.1.2.6. FCS_IPSEC_EXT.1.7

The evaluator shall verify that the values for SA lifetimes can be configured and that the instructions for doing so are located in the guidance documentation. If time-based limits are supported, the evaluator ensures that the Administrator is able to configure Phase 1 SA values for 24 hours. Currently there are no values mandated for the number of bytes, the evaluator just ensures that this can be configured if selected in the requirement.

5.2.1.2.7. FCS_IPSEC_EXT.1.8

The evaluator shall verify that the values for SA lifetimes can be configured and that the instructions for doing so are located in the guidance documentation. If time-based limits are supported, the evaluator ensures that the Administrator is able to configure Phase 2 SA values for 8

hours. Currently there are no values mandated for the number of bytes, the evaluator just ensures that this can be configured if selected in the requirement.

5.2.1.2.8. FCS_IPSEC_EXT.1.11

The evaluator ensures that the guidance documentation describes the configuration of all algorithms selected in the requirement.

5.2.1.2.9. FCS_IPSEC_EXT.1.13

The evaluator ensures the guidance documentation describes how to set up the TOE to use certificates with RSA and/or ECDSA signatures and public keys.

The evaluator shall check that the guidance documentation describes how pre-shared keys are to be generated and established. The description in the guidance documentation shall also indicate how pre-shared key establishment is accomplished for TOEs that can generate a pre-shared key as well as TOEs that simply use a pre-shared key.

The evaluator will ensure that the guidance documentation describes how to configure the TOE to connect to a trusted CA and ensure a valid certificate for that CA is loaded into the TOE and marked “trusted”.

5.2.1.2.10. FCS_IPSEC_EXT.1.14

The evaluator shall ensure that the operational guidance describes all supported identifiers, explicitly states whether the TOE supports the SAN extension or not and includes detailed instructions on how to configure the reference identifier(s) used to check the identity of peer(s). If the identifier scheme implemented by the TOE does not guarantee unique identifiers, the evaluator shall ensure that the operational guidance provides a set of warnings and/or CA policy recommendations that would result in secure TOE use.

5.2.1.3. Tests

5.2.1.3.1. FCS_IPSEC_EXT.1.1

The evaluator uses the guidance documentation to configure the TOE to carry out the following tests:

a. Test 1: The evaluator shall configure the SPD such that there is a rule for dropping a packet, encrypting a packet, and (if configurable) allowing a packet to flow in plaintext. The selectors used in the construction of the rule shall be different such that the evaluator can generate a packet and send packets to the gateway with the appropriate fields (fields that are used by the rule - e.g., the IP addresses, TCP/UDP ports) in the packet header. The evaluator performs both positive and negative test cases for each type of rule (e.g. a packet that matches the rule and another that does not match the rule). The evaluator observes via the audit trail, and packet captures that the TOE exhibited the expected behaviour: appropriate packets were dropped, allowed to flow without modification, encrypted by the IPsec implementation.
b. Test 2: The evaluator shall devise several tests that cover a variety of scenarios for packet processing. As with Test 1, the evaluator ensures both positive and negative test cases are constructed. These scenarios must exercise the range of possibilities for SPD entries and

processing modes as outlined in the TSS and guidance documentation. Potential areas to cover include rules with overlapping ranges and conflicting entries, inbound and outbound packets, and packets that establish SAs as well as packets that belong to established SAs. The evaluator shall verify, via the audit trail and packet captures, for each scenario that the expected behavior is exhibited, and is consistent with both the TSS and the guidance documentation.

5.2.1.3.2. FCS_IPSEC_EXT.1.2

The evaluator uses the guidance documentation to configure the TOE to carry out the following tests:

The evaluator shall configure the SPD such that there is a rule for dropping a packet, encrypting a packet, and (if configurable) allowing a packet to flow in plaintext. The evaluator may use the SPD that was created for verification of FCS_IPSEC_EXT.1.1. The evaluator shall construct a network packet that matches the rule to allow the packet to flow in plaintext and send that packet. The evaluator should observe that the network packet is passed to the proper destination interface with no modification. The evaluator shall then modify a field in the packet header; such that it no longer matches the evaluator-created entries (there may be a “TOE created” final entry that discards packets that do not match any previous entries). The evaluator sends the packet and observes that the packet was dropped.

5.2.1.3.3. FCS_IPSEC_EXT.1.3

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

a. Test 1: [conditional] If tunnel mode is selected, the evaluator uses the guidance documentation to configure the TOE to operate in tunnel mode and also configures a VPN peer to operate in tunnel mode. The evaluator configures the TOE and the VPN peer to use any of the allowable cryptographic algorithms, authentication methods, etc. to ensure an allowable SA can be negotiated. The evaluator shall then initiate a connection from the TOE to connect to the VPN peer. The evaluator observes (for example, in the audit trail and the captured packets) that a successful connection was established using the tunnel mode.
b. Test 2: [conditional] If transport mode is selected, the evaluator uses the guidance documentation to configure the TOE to operate in transport mode and also configures a VPN peer to operate in transport mode. The evaluator configures the TOE and the VPN peer to use any of the allowed cryptographic algorithms, authentication methods, etc. to ensure an allowable SA can be negotiated. The evaluator then initiates a connection from the TOE to connect to the VPN peer. The evaluator observes (for example, in the audit trail and the captured packets) that a successful connection was established using the transport mode.

5.2.1.3.4. FCS_IPSEC_EXT.1.4

The evaluator shall configure the TOE as indicated in the guidance documentation configuring the TOE to use each of the supported algorithms, attempt to establish a connection using ESP, and verify that the attempt succeeds.

5.2.1.3.5. FCS_IPSEC_EXT.1.5

Tests are performed in conjunction with the other IPsec evaluation activities.

a. Test 1: If IKEv1 is selected, the evaluator shall configure the TOE as indicated in the guidance documentation and attempt to establish a connection using an IKEv1 Phase 1 connection in aggressive mode. This attempt should fail. The evaluator should then show that main mode exchanges are supported.

b. Test 2: If NAT traversal is selected within the IKEv2 selection, the evaluator shall configure the TOE so that it will perform NAT traversal processing as described in the TSS and RFC 5996, section 2.23. The evaluator shall initiate an IPsec connection and determine that the NAT is successfully traversed.

5.2.1.3.6. FCS_IPSEC_EXT.1.6

The evaluator shall configure the TOE to use the ciphersuite under test to encrypt the IKEv1 and/or IKEv2 payload and establish a connection with a peer device, which is configured to only accept the payload encrypted using the indicated ciphersuite. The evaluator will confirm the algorithm was that used in the negotiation.

5.2.1.3.7. FCS_IPSEC_EXT.1.7

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

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

a. Test 1: [conditional] If ‘number of bytes’ is selected as the SA lifetime measure, the evaluator shall configure a maximum lifetime in terms of the number of bytes allowed following the guidance documentation. The evaluator shall configure a test peer with a byte lifetime that exceeds the lifetime of the TOE. The evaluator shall establish a SA between the TOE and the test peer, and determine that once the allowed number of bytes through this SA is exceeded, a new SA is negotiated. The evaluator shall verify that the TOE initiates a Phase 1 negotiation.

b. Test 2: [conditional] If ‘length of time’ is selected as the SA lifetime measure, the evaluator shall configure a maximum lifetime of 24 hours for the Phase 1 SA following the guidance documentation. The evaluator shall configure a test peer with a lifetime that exceeds the lifetime of the TOE. The evaluator shall establish a SA between the TOE and the test peer, maintain the Phase 1 SA for 24 hours, and determine that a new Phase 1 SA is negotiated on or before 24 hours has elapsed. The evaluator shall verify that the TOE initiates a Phase 1 negotiation.

5.2.1.3.8. FCS_IPSEC_EXT.1.8

When testing this functionality, the evaluator needs to ensure that both sides are configured appropriately. From the RFC “A difference between IKEv1 and IKEv2 is that in IKEv1 SA lifetimes were negotiated. In IKEv2, each end of the SA is responsible for enforcing its own lifetime policy on

the SA and rekeying the SA when necessary. If the two ends have different lifetime policies, the end with the shorter lifetime will end up always being the one to request the rekeying. If the two ends have the same lifetime policies, it is possible that both will initiate a rekeying at the same time (which will result in redundant SAs). To reduce the probability of this happening, the timing of rekeying requests SHOULD be jittered."

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

a. Test 1: [conditional] If 'number of bytes' is selected as the SA lifetime measure, the evaluator shall configure a maximum lifetime in terms of the number of bytes allowed following the guidance documentation. The evaluator shall configure a test peer with a byte lifetime that exceeds the lifetime of the TOE. The evaluator shall establish a SA between the TOE and the test peer, and determine that once the allowed number of bytes through this SA is exceeded, a new SA is negotiated. The evaluator shall verify that the TOE initiates a Phase 2 negotiation.
b. Test 2: [conditional] If 'length of time' is selected as the SA lifetime measure, the evaluator shall configure a maximum lifetime of 8 hours for the Phase 2 SA following the guidance documentation. The evaluator shall configure a test peer with a lifetime that exceeds the lifetime of the TOE. The evaluator shall establish a SA between the TOE and the test peer, maintain the Phase 1 SA for 8 hours, and determine that once a new Phase 2 SA is negotiated when or before 8 hours has lapsed. The evaluator shall verify that the TOE initiates a Phase 2 negotiation.

5.2.1.3.9. FCS_IPSEC_EXT.1.10

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

a. Test 1: If the first selection is chosen, the evaluator shall check to ensure that, for each DH group supported, the TSS describes the process for generating each nonce. The evaluator shall verify that the TSS indicates that the random number generated that meets the requirements in this cPP is used, and that the length of the nonces meet the stipulations in the requirement.
b. Test 2: If the second selection is chosen, the evaluator shall check to ensure that, for each PRF hash supported, the TSS describes the process for generating each nonce. The evaluator shall verify that the TSS indicates that the random number generated that meets the requirements in this cPP is used, and that the length of the nonces meet the stipulations in the requirement.

5.2.1.3.10. FCS_IPSEC_EXT.1.11

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

5.2.1.3.11. FCS_IPSEC_EXT.1.12

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

a. Test 1: This test shall be performed for each version of IKE supported. The evaluator shall successfully negotiate an IPsec connection using each of the supported algorithms and hash functions identified in the requirements.

b. Test 2: This test shall be performed for each version of IKE supported. The evaluator shall attempt to establish a SA for ESP that selects an encryption algorithm with more strength than that being used for the IKE SA (i.e., symmetric algorithm with a key size larger than that being used for the IKE SA). Such attempts should fail.

c. Test 3: This test shall be performed for each version of IKE supported. The evaluator shall attempt to establish an IKE SA using an algorithm that is not one of the supported algorithms and hash functions identified in the requirements. Such an attempt should fail.

d. Test 4: This test shall be performed for each version of IKE supported. The evaluator shall attempt to establish a SA for ESP (assumes the proper parameters where used to establish the IKE SA) that selects an encryption algorithm that is not identified in FCS_IPSEC_EXT.1.4. Such an attempt should fail.

5.2.1.3.12. FCS_IPSEC_EXT.1.13

For efficiency sake, the testing is combined with the testing for FIA_X509_EXT.1, FIA_X509_EXT.2 (for IPsec connections), and FCS_IPSEC_EXT.1.1.

5.2.1.3.13. FCS_IPSEC_EXT.1.14

In the context of the tests below, a valid certificate is a certificate that passes FIA_X509_EXT.1 validation checks but does not necessarily contain an authorized subject.

The evaluator shall perform the following tests:

- Test 1: [conditional] For each CN/identifier type combination selected, the evaluator shall configure the peer's reference identifier on the TOE (per the administrative guidance) to match the field in the peer's presented certificate and shall verify that the IKE authentication succeeds. If the TOE prioritizes CN checking over SAN (through explicit configuration of the field when specifying the reference identifier or prioritization rules), the evaluator shall also configure the SAN so it contains an incorrect identifier of the correct type (e.g. the reference identifier on the TOE is example.com, the CN=example.com, and the SAN:FQDN=otherdomain.com) and verify that IKE authentication succeeds.
- Test 2: [conditional] For each SAN/identifier type combination selected, the evaluator shall configure the peer's reference identifier on the TOE (per the administrative guidance) to match the field in the peer's presented certificate and shall verify that the IKE authentication succeeds. If the TOE prioritizes SAN checking over CN (through explicit specification of the field when specifying the reference identifier or prioritization rules), the evaluator shall also configure the CN so it contains an incorrect identifier formatted to be the same type (e.g. the reference identifier on the TOE is DNS-ID; identify certificate has an identifier in SAN with correct DNS-ID, CN with incorrect DNS-ID (and not a different type of identifier)) and verify that IKE authentication succeeds.
- Test 3: [conditional] For each CN/identifier type combination selected, the evaluator shall:
  a. Create a valid certificate with the CN so it contains the valid identifier followed by nul character (ASCII null character in hex is '0x00'). If the TOE prioritizes CN checking over SAN (through explicit specification of the field when specifying the reference identifier or prioritization rules) for the same identifier type, the evaluator shall configure the SAN so it matches the reference identifier.

b. Configure the peer's reference identifier on the TOE (per the administrative guidance) to match the CN without the nul character (ASCII null character in hex is '0x00') and verify that IKE authentication fails.

- Test 4: [conditional] For each SAN/identifier type combination selected, the evaluator shall:
  a. Create a valid certificate with an incorrect identifier in the SAN. The evaluator shall configure a string representation of the correct identifier in the DN. If the TOE prioritizes CN checking over SAN (through explicit specification of the field when specifying the reference identifier or prioritization rules) for the same identifier type, the addition/modification shall be to any non-CN field of the DN. Otherwise, the addition/modification shall be to the CN.
  b. Configure the peer's reference identifier on the TOE (per the administrative guidance) to match the correct identifier (expected in the SAN) and verify that IKE authentication fails.
- Test 5: [conditional] If the TOE supports DN identifier types, the evaluator shall configure the peer's reference identifier on the TOE (per the administrative guidance) to match the subject DN in the peer's presented certificate and shall verify that the IKE authentication succeeds.
- Test 6: [conditional] If the TOE supports DN identifier types, to demonstrate a bit-wise comparison of the DN, the evaluator shall create the following valid certificates and verify that the IKE authentication fails when each certificate is presented to the TOE:
  a. Duplicate the CN field, so the otherwise authorized DN contains two identical CNs.
  b. Append '\0' to a non-CN field of an otherwise authorized DN.

5.2.2. FCS_TLSC_EXT.1 TLS Client Protocol without mutual authentication

5.2.2.1. TSS

5.2.2.1.1. FCS_TLSC_EXT.1.1

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

5.2.2.1.2. FCS_TLSC_EXT.1.2

The evaluator shall ensure that the TSS describes the client's method of establishing all reference identifiers from the administrator/application-configured reference identifier, including which types of reference identifiers are supported (e.g. application-specific Subject Alternative Names) and whether IP addresses and wildcards are supported.

If IP addresses are supported in the CN as reference identifiers, the evaluator shall ensure that the TSS describes the TOE's conversion of the text representation of the IP address in the CN to a binary representation of the IP address in network byte order. The evaluator shall also ensure that the TSS describes whether canonical format (RFC 5952 for IPv6, RFC 3986 for IPv4) is enforced.

5.2.2.1.3. FCS_TLSC_EXT.1.4

The evaluator shall verify that TSS describes the Supported Elliptic Curves/Supported Groups Extension and whether the required behaviour is performed by default or may be configured.

5.2.2.2. Guidance Documentation

5.2.2.2.1. FCS_TLSC_EXT.1.1

The evaluator shall check the guidance documentation to ensure that it contains instructions on configuring the TOE so that TLS conforms to the description in the TSS.

5.2.2.2.2. FCS_TLSC_EXT.1.2

The evaluator shall ensure that the operational guidance describes all supported identifiers, explicitly states whether the TOE supports the SAN extension or not and includes detailed instructions on how to configure the reference identifier(s) used to check the identity of peer(s). If the identifier scheme implemented by the TOE includes support for IP addresses, the evaluator shall ensure that the operational guidance provides a set of warnings and/or CA policy recommendations that would result in secure TOE use.

5.2.2.2.3. FCS_TLSC_EXT.1.4

If the TSS indicates that the Supported Elliptic Curves/Supported Groups Extension must be configured to meet the requirement, the evaluator shall verify that AGD guidance includes configuration of the Supported Elliptic Curves/Supported Groups Extension.

5.2.2.3. Tests

For all tests in this chapter the TLS server used for testing of the TOE shall be configured not to require mutual authentication.

5.2.2.3.1. FCS_TLSC_EXT.1.1

Test 1: The evaluator shall establish a TLS connection using each of the ciphersuites specified by the requirement. This connection may be established as part of the establishment of a higher-level protocol, e.g., as part of an HTTPS session. It is sufficient to observe the successful negotiation of a ciphersuite to satisfy the intent of the test; it is not necessary to examine the characteristics of the encrypted traffic to discern the ciphersuite being used (for example, that the cryptographic algorithm is 128-bit AES and not 256-bit AES).

Test 2: The evaluator shall attempt to establish the connection using a server with a server certificate that contains the Server Authentication purpose in the extendedKeyUsage field and verify that a connection is established. The evaluator will then verify that the client rejects an otherwise valid server certificate that lacks the Server Authentication purpose in the extendedKeyUsage field, and a connection is not established. Ideally, the two certificates should be identical except for the extendedKeyUsage field.

Test 3: The evaluator shall send a server certificate in the TLS connection that does not match the server-selected ciphersuite (for example, send an ECDSA certificate while using the TLS_RSA_WITH_AES_128_CBC_SHA ciphersuite). The evaluator shall verify that the TOE disconnects after receiving the server's Certificate handshake message.

Test 4: The evaluator shall perform the following 'negative tests':

a. The evaluator shall configure the server to select the TLS_NULL_WITH_NULL_NULL ciphersuite

and verify that the client denies the connection.

b. Modify the server's selected ciphersuite in the Server Hello handshake message to be a ciphersuite not presented in the Client Hello handshake message. The evaluator shall verify that the client rejects the connection after receiving the Server Hello.

c. [conditional]: If the TOE presents the Supported Elliptic Curves/Supported Groups Extension the evaluator shall configure the server to perform an ECDHE or DHE key exchange in the TLS connection using a non-supported curve/group (for example P-192) and shall verify that the TOE disconnects after receiving the server's Key Exchange handshake message.

Test 5: The evaluator performs the following modifications to the traffic:

a. Change the TLS version selected by the server in the Server Hello to a non-supported TLS version and verify that the client rejects the connection.

b. [conditional]: If using DHE or ECDH, modify the signature block in the Server's Key Exchange handshake message, and verify that the handshake does not finished successfully, and no application data flows. This test does not apply to cipher suites using RSA key exchange. If a TOE only supports RSA key exchange in conjunction with TLS, then this test shall be omitted.

Test 6: The evaluator performs the following 'scrambled message tests':

a. Modify a byte in the Server Finished handshake message and verify that the handshake does not finish successfully and no application data flows.

b. Send a garbled message from the server after the server has issued the ChangeCipherSpec message and verify that the handshake does not finish successfully and no application data flows.

c. Modify at least one byte in the server's nonce in the Server Hello handshake message and verify that the client rejects the Server Key Exchange handshake message (if using a DHE or ECDHE ciphersuite) or that the server denies the client's Finished handshake message.

5.2.2.3.2. FCS_TLSC_EXT.1.2

Note that the following tests are marked conditional and are applicable under the following conditions:

a. For TLS-based trusted channel communications according to FTP_ITC.1 where RFC 6125 is selected, tests 1-6 are applicable.

   or

b. For TLS-based trusted path communications according to FTP_TRP where RFC 6125 is selected, tests 1-6 are applicable

Note that for some tests additional conditions apply.

IP addresses are binary values that must be converted to a textual representation when presented in the CN of a certificate. When testing IP addresses in the CN, the evaluator shall follow the following formatting rules:

- IPv4: The CN contains a single address that is represented a 32-bit numeric address (IPv4) is

written in decimal as four numbers that range from 0-255 separated by periods as specified in RFC 3986.

- IPv6: The CN contains a single IPv6 address that is represented as eight colon separated groups of four lowercase hexadecimal digits, each group representing 16 bits as specified in RFC 4291. Note: Shortened addresses, suppressed zeros, and embedded IPv4 addresses are not tested.

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

a. Test 1 [conditional]: The evaluator shall present a server certificate that contains a CN that does not match the reference identifier and does not contain the SAN extension. The evaluator shall verify that the connection fails. The evaluator shall repeat this test for each identifier type (e.g. IPv4, IPv6, FQDN) supported in the CN. When testing IPv4 or IPv6 addresses, the evaluator shall modify a single decimal or hexadecimal digit in the CN. Remark: Some systems might require the presence of the SAN extension. In this case the connection would still fail but for the reason of the missing SAN extension instead of the mismatch of CN and reference identifier. Both reasons are acceptable to pass Test 1.

b. Test 2 [conditional]: The evaluator shall present a server certificate that contains a CN that matches the reference identifier, contains the SAN extension, but does not contain an identifier in the SAN that matches the reference identifier. The evaluator shall verify that the connection fails. The evaluator shall repeat this test for each supported SAN type (e.g. IPv4, IPv6, FQDN, URI). When testing IPv4 or IPv6 addresses, the evaluator shall modify a single decimal or hexadecimal digit in the SAN.

c. Test 3 [conditional]: If the TOE does not mandate the presence of the SAN extension, the evaluator shall present a server certificate that contains a CN that matches the reference identifier and does not contain the SAN extension. The evaluator shall verify that the connection succeeds. The evaluator shall repeat this test for each identifier type (e.g. IPv4, IPv6, FQDN) supported in the CN. If the TOE does mandate the presence of the SAN extension, this Test shall be omitted.

d. Test 4 [conditional]: The evaluator shall present a server certificate that contains a CN that does not match the reference identifier but does contain an identifier in the SAN that matches. The evaluator shall verify that the connection succeeds. The evaluator shall repeat this test for each supported SAN type (e.g. IPv4, IPv6, FQDN, SRV).

e. Test 5 [conditional]: The evaluator shall perform the following wildcard tests with each supported type of reference identifier that includes a DNS name (i.e. CN-ID with DNS, DNS-ID, SRV-ID, URI-ID):

    1. [conditional]: The evaluator shall present a server certificate containing a wildcard that is not in the left-most label of the presented identifier (e.g. foo.*.example.com) and verify that the connection fails.

    2. [conditional]: The evaluator shall present a server certificate containing a wildcard in the left-most label (e.g. *.example.com). The evaluator shall configure the reference identifier with a single left-most label (e.g. foo.example.com) and verify that the connection succeeds, if wildcards are supported, or fails if wildcards are not supported. The evaluator shall configure the reference identifier without a left-most label as in the certificate (e.g. example.com) and verify that the connection fails. The evaluator shall configure the

reference identifier with two left-most labels (e.g. bar.foo.example.com) and verify that the connection fails. (Remark: Support for wildcards was always intended to be optional. It is sufficient to state that the TOE does not support wildcards and observe rejected connection attempts to satisfy corresponding assurance activities.)

f. Test 6 [conditional]: If IP addresses are supported, the evaluator shall present a server certificate that contains a CN that matches the reference identifier, except one of the groups has been replaced with an asterisk (*) (e.g. CN=192.168.1.* when connecting to 192.168.1.20, CN=2001:0DB8:0000:0000:0008:0800:200C:* when connecting to 2001:0DB8:0000:0000:0008:0800:200C:417A). The certificate shall not contain the SAN extension. The evaluator shall verify that the connection fails. The evaluator shall repeat this test for each supported IP address version (e.g. IPv4, IPv6).

   Remark: Some systems might require the presence of the SAN extension. In this case the connection would still fail but for the reason of the missing SAN extension instead of the mismatch of CN and reference identifier. Both reasons are acceptable to pass Test 6.

5.2.2.3.3. FCS_TLSC_EXT.1.3

The evaluator shall demonstrate that using an invalid certificate results in the function failing as follows:

Test 1: Using the administrative guidance, the evaluator shall load a CA certificate or certificates needed to validate the presented certificate used to authenticate an external entity and demonstrate that the function succeeds, and a trusted channel can be established.

Test 2: The evaluator shall then change the presented certificate(s) so that validation fails and show that the certificate is not automatically accepted. The evaluator shall repeat this test to cover the selected types of failure defined in the SFR (i.e. the selected ones from failed matching of the reference identifier, failed validation of the certificate path, failed validation of the expiration date, failed determination of the revocation status). The evaluator performs the action indicated in the SFR selection observing the TSF resulting in the expected state for the trusted channel (e.g. trusted channel was established) covering the types of failure for which an override mechanism is defined.

Test 3 [conditional]: The purpose of this test to verify that only selected certificate validation failures could be administratively overridden. If any override mechanism is defined for failed certificate validation, the evaluator shall configure a new presented certificate that does not contain a valid entry in one of the mandatory fields or parameters (e.g. inappropriate value in extendedKeyUsage field) but is otherwise valid and signed by a trusted CA. The evaluator shall confirm that the certificate validation fails (i.e. certificate is rejected), and there is no administrative override available to accept such certificate.

5.2.2.3.4. FCS_TLSC_EXT.1.4

Test 1 [conditional]: If the TOE presents the Supported Elliptic Curves/Supported Groups Extension, the evaluator shall configure the server to perform ECDHE or DHE (as applicable) key exchange using each of the TOE's supported curves and/or groups. The evaluator shall verify that the TOE successfully connects to the server.

5.2.3. FCS_TLSS_EXT.1 TLS Server Protocol without mutual authentication

5.2.3.1. TSS

5.2.3.1.1. FCS_TLSS_EXT.1.1

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

5.2.3.1.2. FCS_TLSS_EXT.1.2

The evaluator shall verify that the TSS contains a description of how the TOE technically prevents the use of old SSL and TLS versions.

5.2.3.1.3. FCS_TLSS_EXT.1.3

If using ECDHE and/or DHE ciphers, the evaluator shall verify that the TSS lists all EC Diffie-Hellman curves and/or Diffie-Hellman groups used in the key establishment by the TOE when acting as a TLS Server. For example, if the TOE supports TLS_DHE_RSA_WITH_AES_128_CBC_SHA cipher and Diffie-Hellman parameters with size 2048 bits, then list Diffie-Hellman Group 14.

5.2.3.1.4. FCS_TLSS_EXT.1.4

The evaluator shall verify that the TSS describes if session resumption based on session IDs is supported (RFC 4346 and/or RFC 5246) and/or if session resumption based on session tickets is supported (RFC 5077).

If session tickets are supported, the evaluator shall verify that the TSS describes that the session tickets are encrypted using symmetric algorithms consistent with FCS_COP.1/DataEncryption. The evaluator shall verify that the TSS identifies the key lengths and algorithms used to protect session tickets.

If session tickets are supported, the evaluator shall verify that the TSS describes that session tickets adhere to the structural format provided in section 4 of RFC 5077 and if not, a justification shall be given of the actual session ticket format.

If the TOE claims a (D)TLS server capable of session resumption (as a single context, or across multiple contexts), the evaluator verifies that the TSS describes how session resumption operates (i.e. what would trigger a full handshake, e.g. checking session status, checking Session ID, etc.). If multiple contexts are used the TSS describes how session resumption is coordinated across those contexts. In case session establishment and session resumption are always using a separate context, the TSS shall describe how the contexts interact with respect to session resumption (in particular regarding the session ID). It is acceptable for sessions established in one context to be resumable in another context.

5.2.3.2. Guidance Documentation

5.2.3.2.1. FCS_TLSS_EXT.1.1

The evaluator shall check the guidance documentation to ensure that it contains instructions on

configuring the TOE so that TLS conforms to the description in the TSS (for instance, the set of ciphersuites advertised by the TOE may have to be restricted to meet the requirements).

5.2.3.2.2. FCS_TLSS_EXT.1.2

The evaluator shall verify that any configuration necessary to meet the requirement must be contained in the AGD guidance.

5.2.3.2.3. FCS_TLSS_EXT.1.3

The evaluator shall verify that any configuration necessary to meet the requirement must be contained in the AGD guidance.

5.2.3.2.4. FCS_TLSS_EXT.1.4

The evaluator shall verify that any configuration necessary to meet the requirement must be contained in the AGD guidance.

5.2.3.3. Tests

5.2.3.3.1. FCS_TLSS_EXT.1.1

Test 1: The evaluator shall establish a TLS connection using each of the ciphersuites specified by the requirement. This connection may be established as part of the establishment of a higher-level protocol, e.g., as part of an HTTPS session. It is sufficient to observe the successful negotiation of a ciphersuite to satisfy the intent of the test; it is not necessary to examine the characteristics of the encrypted traffic to discern the ciphersuite being used (for example, that the cryptographic algorithm is 128-bit AES and not 256-bit AES).

Test 2: The evaluator shall send a Client Hello to the server with a list of ciphersuites that does not contain any of the ciphersuites in the server's ST and verify that the server denies the connection. Additionally, the evaluator shall send a Client Hello to the server containing only the TLS_NULL_WITH_NULL_NULL ciphersuite and verify that the server denies the connection.

Test 3: The evaluator shall perform the following modifications to the traffic:

a. Modify a byte in the Client Finished handshake message, and verify that the server rejects the connection and does not send any application data.
b. (Test Intent: The intent of this test is to ensure that the server's TLS implementation immediately makes use of the key exchange and authentication algorithms to: a) Correctly encrypt (D)TLS Finished message and b) Encrypt every (D)TLS message after session keys are negotiated.)

   The evaluator shall use one of the claimed ciphersuites to complete a successful handshake and observe transmission of properly encrypted application data. The evaluator shall verify that no Alert with alert level Fatal (2) messages were sent.

   The evaluator shall verify that the Finished message (Content type hexadecimal 16 and handshake message type hexadecimal 14) is sent immediately after the server's ChangeCipherSpec (Content type hexadecimal 14) message. The evaluator shall examine the

Finished message (encrypted example in hexadecimal of a TLS record containing a Finished message, 16 03 03 00 40 11 22 33 44 55…) and confirm that it does not contain unencrypted data (unencrypted example in hexadecimal of a TLS record containing a Finished message, 16 03 03 00 40 14 00 00 0c…), by verifying that the first byte of the encrypted Finished message does not equal hexadecimal 14 for at least one of three test messages. There is a chance that an encrypted Finished message contains a hexadecimal value of '14' at the position where a plaintext Finished message would contain the message type code '14'. If the observed Finished message contains a hexadecimal value of '14' at the position where the plaintext Finished message would contain the message type code, the test shall be repeated three times in total. In case the value of '14' can be observed in all three tests it can be assumed that the Finished message has indeed been sent in plaintext and the test has to be regarded as 'failed'. Otherwise it has to be assumed that the observation of the value '14' has been due to chance and that the Finished message has indeed been sent encrypted. In that latter case the test shall be regarded as ‘passed’.

5.2.3.3.2. FCS_TLSS_EXT.1.2

The evaluator shall send a Client Hello requesting a connection for all mandatory and selected protocol versions in the SFR (e.g. by enumeration of protocol versions in a test client) and verify that the server denies the connection for each attempt.

5.2.3.3.3. FCS_TLSS_EXT.1.3

Test 1: [conditional] If ECDHE ciphersuites are supported:

a. The evaluator shall repeat this test for each supported elliptic curve. The evaluator shall attempt a connection using a supported ECDHE ciphersuite and a single supported elliptic curve specified in the Elliptic Curves Extension. The Evaluator shall verify (though a packet capture or instrumented client) that the TOE selects the same curve in the Server Key Exchange message and successfully establishes the connection.
b. The evaluator shall attempt a connection using a supported ECDHE ciphersuite and a single unsupported elliptic curve (e.g. secp192r1 (0x13)) specified in RFC4492, chap. 5.1.1. The evaluator shall verify that the TOE does not send a Server Hello message and the connection is not successfully established.

Test 2: [conditional] If DHE ciphersuites are supported, the evaluator shall repeat the following test for each supported parameter size. If any configuration is necessary, the evaluator shall configure the TOE to use a supported Diffie-Hellman parameter size. The evaluator shall attempt a connection using a supported DHE ciphersuite. The evaluator shall verify (through a packet capture or instrumented client) that the TOE sends a Server Key Exchange Message where p Length is consistent with the message are the ones configured Diffie-Hellman parameter size(s).

Test 3: [conditional] If RSA key establishment ciphersuites are supported, the evaluator shall repeat this test for each RSA key establishment key size. If any configuration is necessary, the evaluator shall configure the TOE to perform RSA key establishment using a supported key size (e.g. by loading a certificate with the appropriate key size). The evaluator shall attempt a connection using a supported RSA key establishment ciphersuite. The evaluator shall verify (through a packet capture or instrumented client) that the TOE sends a certificate whose modulus is consistent with the configured RSA key size.

5.2.3.3.4. FCS_TLSS_EXT.1.4

Test Objective: To demonstrate that the TOE will not resume a session for which the client failed to complete the handshake (independent of TOE support for session resumption).

Test 1 [conditional]: If the TOE does not support session resumption based on session IDs according to RFC4346 (TLS1.1) or RFC5246 (TLS1.2) or session tickets according to RFC5077, the evaluator shall perform the following test:

a. The client sends a Client Hello with a zero-length session identifier and with a SessionTicket extension containing a zero-length ticket.

b. The client verifies the server does not send a NewSessionTicket handshake message (at any point in the handshake).

c. The client verifies the Server Hello message contains a zero-length session identifier or passes the following steps:

   Note: The following steps are only performed if the ServerHello message contains a non-zero length SessionID.

d. The client completes the TLS handshake and captures the SessionID from the ServerHello.

e. The client sends a ClientHello containing the SessionID captured in step d). This can be done by keeping the TLS session in step d) open or start a new TLS session using the SessionID captured in step d).

f. The client verifies the TOE (1) implicitly rejects the SessionID by sending a ServerHello containing a different SessionID and by performing a full handshake (as shown in Figure 1 of RFC 4346 or RFC 5246), or (2) terminates the connection in some way that prevents the flow of application data.

Remark: If multiple contexts are supported for session resumption, the session ID or session ticket may be obtained in one context for resumption in another context. It is possible that one or more contexts may only permit the construction of sessions to be reused in other contexts but not actually permit resumption themselves. For contexts which do not permit resumption, the evaluator is required to verify this behaviour subject to the description provided in the TSS. It is not mandated that the session establishment and session resumption share context. For example, it is acceptable for a control channel to establish and application channel to resume the session

Test 2 [conditional]: If the TOE supports session resumption using session IDs according to RFC4346 (TLS1.1) or RFC5246 (TLS1.2), the evaluator shall carry out the following steps (note that for each of these tests, it is not necessary to perform the test case for each supported version of TLS):

a. The evaluator shall conduct a successful handshake and capture the TOE-generated session ID in the Server Hello message. The evaluator shall then initiate a new TLS connection and send the previously captured session ID to show that the TOE resumed the previous session by responding with ServerHello containing the same SessionID immediately followed by ChangeCipherSpec and Finished messages (as shown in Figure 2 of RFC 4346 or RFC 5246).

b. The evaluator shall initiate a handshake and capture the TOE-generated session ID in the Server Hello message. The evaluator shall then, within the same handshake, generate or force an unencrypted fatal Alert message immediately before the client would otherwise send its

ChangeCipherSpec message thereby disrupting the handshake. The evaluator shall then initiate a new Client Hello using the previously captured session ID, and verify that the server (1) implicitly rejects the session ID by sending a ServerHello containing a different SessionID and performing a full handshake (as shown in figure 1 of RFC 4346 or RFC 5246), or (2) terminates the connection in some way that prevents the flow of application data.

Remark: If multiple contexts are supported for session resumption, for each of the above test cases, the session ID may be obtained in one context for resumption in another context. There is no requirement that the session ID be obtained and replayed within the same context subject to the description provided in the TSS. All contexts that can reuse a session ID constructed in another context must be tested. It is not mandated that the session establishment and session resumption share context. For example, it is acceptable for a control channel to establish and application channel to resume the session.

Test 3 [conditional]: If the TOE supports session tickets according to RFC5077, the evaluator shall carry out the following steps (note that for each of these tests, it is not necessary to perform the test case for each supported version of TLS):

a. The evaluator shall permit a successful TLS handshake to occur in which a session ticket is exchanged with the non-TOE client. The evaluator shall then attempt to correctly reuse the previous session by sending the session ticket in the ClientHello. The evaluator shall confirm that the TOE responds with a ServerHello with an empty SessionTicket extension, NewSessionTicket, ChangeCipherSpec and Finished messages (as seen in figure 2 of RFC 5077).
b. The evaluator shall permit a successful TLS handshake to occur in which a session ticket is exchanged with the non-TOE client. The evaluator will then modify the session ticket and send it as part of a new Client Hello message. The evaluator shall confirm that the TOE either (1) implicitly rejects the session ticket by performing a full handshake (as shown in figure 3 or 4 of RFC 5077), or (2) terminates the connection in some way that prevents the flow of application data.

5.2.4. FCS_DTLSC_EXT.1 DTLS Client Protocol without mutual authentication

5.2.4.1. TSS

5.2.4.1.1. FCS_DTLSC_EXT.1.1

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

5.2.4.1.2. FCS_DTLSC_EXT.1.2

The evaluator shall ensure that the TSS describes the client's method of establishing all reference identifiers from the administrator/application-configured reference identifier, including which types of reference identifiers are supported (e.g. application-specific Subject Alternative Names) and whether IP addresses and wildcards are supported.

If IP addresses are supported in the CN as reference identifiers, the evaluator shall ensure that the TSS describes the TOE's conversion of the text representation of the IP address in the CN to a binary

representation of the IP address in network byte order. The evaluator shall also ensure that the TSS describes whether canonical format (RFC 5952 for IPv6, RFC 3986 for IPv4) is enforced.

5.2.4.1.3. FCS_DTLSC_EXT.1.4

The evaluator shall verify that TSS describes the Supported Elliptic Curves/Supported Groups Extension and whether the required behaviour is performed by default or may be configured.

5.2.4.2. Guidance Documentation

5.2.4.2.1. FCS_DTLSC_EXT.1.1

The evaluator shall also check the guidance documentation to ensure that it contains instructions on configuring the TOE so that DTLS conforms to the description in the TSS.

5.2.4.2.2. FCS_DTLSC_EXT.1.2

The evaluator shall ensure that the operational guidance describes all supported identifiers, explicitly states whether the TOE supports the SAN extension or not and includes detailed instructions on how to configure the reference identifier(s) used to check the identity of peer(s). If the identifier scheme implemented by the TOE includes support for IP addresses, the evaluator shall ensure that the operational guidance provides a set of warnings and/or CA policy recommendations that would result in secure TOE use.

5.2.4.2.3. FCS_DTLSC_EXT.1.4

If the TSS indicates that the Supported Elliptic Curves/Supported Groups Extension must be configured to meet the requirement, the evaluator shall verify that AGD guidance includes configuration of the Supported Elliptic Curves/Supported Groups Extension.

5.2.4.3. Tests

For all tests in this chapter the DTLS server used for testing of the TOE shall be configured not to require mutual authentication.

For clarification: DTLS communication packets might be received in a different order than sent due to the use of the UDP protocol. All tests requiring a specific order of test steps ("before", "after") are therefore referring to the sequence numbering of DTLS packets.

5.2.4.3.1. FCS_DTLSC_EXT.1.1

Test 1: The evaluator shall establish a DTLS connection using each of the ciphersuites specified by the requirement. This connection may be established as part of the establishment of a higher-level application protocol, e.g., as part of a syslog session. It is sufficient to observe the successful negotiation of a ciphersuite to satisfy the intent of the test; it is not necessary to examine the characteristics of the encrypted traffic in an attempt to discern the ciphersuite being used (for example, that the cryptographic algorithm is 128-bit AES and not 256-bit AES). 11 The goal of the following test is to verify that the TOE accepts only certificates with appropriate values in the extendedKeyUsage extension, and implicitly that the TOE correctly parses the extendedKeyUsage extension as part of X.509v3 server certificate validation.

Test 2: The evaluator shall attempt to establish the connection using a server with a server certificate that contains the Server Authentication purpose in the extendedKeyUsage extension and verify that a connection is established. The evaluator shall repeat this test using a different, but otherwise valid and trusted, certificate that lacks the Server Authentication purpose in the extendedKeyUsage extension and ensure that a connection is not established. Ideally, the two certificates should be similar in structure, the types of identifiers used, and the chain of trust.

Test 3: The evaluator shall send a server certificate in the DTLS connection that does not match the server-selected ciphersuite (for example, send an ECDSA certificate while using the TLS_RSA_WITH_AES_128_CBC_SHA ciphersuite). The evaluator shall verify that the TOE disconnects after receiving the server's Certificate handshake message.

Test 4: The evaluator shall perform the following 'negative tests':

a. The evaluator shall configure the server to select the TLS_NULL_WITH_NULL_NULL ciphersuite and verify that the client denies the connection.
b. Modify the server's selected ciphersuite in the Server Hello handshake message to be a ciphersuite not presented in the Client Hello handshake message. The evaluator shall verify that the client rejects the connection after receiving the Server Hello.
c. [conditional]: If the TOE presents the Supported Elliptic Curves/Supported Groups Extension the evaluator shall configure the server to perform an ECDHE or DHE key exchange in the DTLS connection using a non-supported curve/group (for example P-192) and shall verify that the TOE disconnects after receiving the server's Key Exchange handshake message.

Test 5: The evaluator performs the following modifications to the traffic:

a. Change the DTLS version selected by the server in the Server Hello to a non-supported DTLS version and verify that the client rejects the connection.
b. [conditional]: If using DHE or ECDH, modify the signature block in the Server's Key Exchange handshake message, and verify that the handshake is not finished successfully and no application data flows. This test does not apply to cipher suites using RSA key exchange. If a TOE only supports RSA key exchange in conjunction with DTLS, then this test shall be omitted.

Test 6: The evaluator performs the following 'scrambled message tests':

a. Modify a byte in the Server Finished handshake message and verify that the handshake is not finished successfully and no application data flows.
b. Send a garbled message from the Server after the Server has issued the ChangeCipherSpec message and verify that the handshake is not finished successfully and no application data flows.
c. Modify at least one byte in the server's nonce in the Server Hello handshake message and verify that the client rejects the Server Key Exchange handshake message (if using a DHE or ECDHE ciphersuite) or that the server denies the client's Finished handshake message.

5.2.4.3.2. FCS_DTLSC_EXT.1.2

Note that tests 1-6 are only applicable to:

- DTLS-based trusted channel communications according to FTP_ITC.1 and trusted path communications according to FTP_TRP.1

IP addresses are binary values that must be converted to a textual representation when presented in the CN of a certificate. When testing IP addresses in the CN, the evaluator shall follow the following formatting rules:

- IPv4: The CN contains a single address that is represented a 32-bit numeric address (IPv4) is written in decimal as four numbers that range from 0-255 separated by periods as specified in RFC 3986.
- IPv6: The CN contains a single IPv6 address that is represented as eight colon separated groups of four lowercase hexadecimal digits, each group representing 16 bits as specified in RFC 4291. Note: Shortened addresses, suppressed zeros, and embedded IPv4 addresses are not tested..

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

a. Test 1 [conditional]: The evaluator shall present a server certificate that contains a CN that does not match the reference identifier and does not contain the SAN extension. The evaluator shall verify that the connection fails. The evaluator shall repeat this test for each identifier type (e.g. IPv4, IPv6, FQDN) supported in the CN. When testing IPv4 or IPv6 addresses, the evaluator shall modify a single decimal or hexadecimal digit in the CN. Remark: Some systems might require the presence of the SAN extension. In this case the connection would still fail but for the reason of the missing SAN extension instead of the mismatch of CN and reference identifier. Both reasons are acceptable to pass Test 1.

b. Test 2 [conditional]: The evaluator shall present a server certificate that contains a CN that matches the reference identifier, contains the SAN extension, but does not contain an identifier in the SAN that matches the reference identifier. The evaluator shall verify that the connection fails. The evaluator shall repeat this test for each supported SAN type (e.g. IPv4, IPv6, FQDN, URI). When testing IPv4 or IPv6 addresses, the evaluator shall modify a single decimal or hexadecimal digit in the SAN.

c. Test 3 [conditional]: If the TOE does not mandate the presence of the SAN extension, the evaluator shall present a server certificate that contains a CN that matches the reference identifier and does not contain the SAN extension. The evaluator shall verify that the connection succeeds. The evaluator shall repeat this test for each identifier type (e.g. IPv4, IPv6, FQDN) supported in the CN. If the TOE does mandate the presence of the SAN extension, this test shall be omitted.

d. Test 4 [conditional]: The evaluator shall present a server certificate that contains a CN that does not match the reference identifier but does contain an identifier in the SAN that matches. The evaluator shall verify that the connection succeeds. The evaluator shall repeat this test for each supported SAN type (e.g. IPv4, IPv6, FQDN, SRV).

e. Test 5 [conditional]: The evaluator shall perform the following wildcard tests with each supported type of reference identifier that includes a DNS name (i.e. CN-ID with DNS, DNS-ID, SRV-ID, URI-ID):

    1. [conditional]: The evaluator shall present a server certificate containing a wildcard that is not in the left-most label of the presented identifier (e.g. foo.*.example.com) and verify that

the connection fails.

2. [conditional]: The evaluator shall present a server certificate containing a wildcard in the left-most label (e.g. *.example.com). The evaluator shall configure the reference identifier with a single left-most label (e.g. foo.example.com) and verify that the connection succeeds, if wildcards are supported, or fails if wildcards are not supported. The evaluator shall configure the reference identifier without a left-most label as in the certificate (e.g. example.com) and verify that the connection fails. The evaluator shall configure the reference identifier with two left-most labels (e.g. bar.foo.example.com) and verify that the connection fails. (Remark: Support for wildcards was always intended to be optional. It is sufficient to state that the TOE does not support wildcards and observe rejected connection attempts to satisfy corresponding assurance activities).

f. Test 6:[conditional] If IP address identifiers supported in the SAN or CN, the evaluator shall present a server certificate that contains a CN that matches the reference identifier, except one of the groups has been replaced with a wildcard asterisk (*) (e.g. CN=*.168.0.1 when connecting to 192.168.0.1… Objective: The objective of this test is to ensure the TOE is able to differentiate between IP address identifiers that are not allowed to contain wildcards and other types of identifiers that may contain wildcards.

Remark: Some systems might require the presence of the SAN extension. In this case the connection would still fail but for the reason of the missing SAN extension instead of the mismatch of CN and reference identifier. Both reasons are acceptable to pass Test 6.

5.2.4.3.3. FCS_DTLSC_EXT.1.3

The evaluator shall demonstrate that using an invalid certificate results in the function failing as follows:

Test 1: Using the administrative guidance, the evaluator shall load a CA certificate or certificates needed to validate the presented certificate used to authenticate an external entity and demonstrate that the function succeeds and a trusted channel can be established.

Test 2: The evaluator shall then change the presented certificate(s) so that validation fails and show that the certificate is not automatically accepted. The evaluator shall repeat this test to cover the selected types of failure defined in the SFR (i.e. the selected ones from failed matching of the reference identifier, failed validation of the certificate path, failed validation of the expiration date, failed determination of the revocation status). The evaluator performs the action indicated in the SFR selection observing the TSF resulting in the expected state for the trusted channel (e.g. trusted channel was established) covering the types of failure for which an override mechanism is defined.

Test 3 [conditional]: The purpose of this test to verify that only selected certificate validation failures could be administratively overridden. If any override mechanism is defined for failed certificate validation, the evaluator shall configure a new presented certificate that does not contain a valid entry in one of the mandatory fields or parameters (e.g. inappropriate value in extendedKeyUsage field) but is otherwise valid and signed by a trusted CA. The evaluator shall confirm that the certificate validation fails (i.e. certificate is rejected), and there is no administrative override available to accept such certificate.

5.2.4.3.4. FCS_DTLSC_EXT.1.4

Test 1 [conditional]: If the TOE presents the Supported Elliptic Curves/Supported Groups Extension, the evaluator shall configure the server to perform ECDHE or DHE (as applicable) key exchange using each of the TOE's supported curves and/or groups. The evaluator shall verify that the TOE successfully connects to the server.

5.2.5. FCS_DTLSS_EXT.1 DTLS Server Protocol without mutual authentication

5.2.5.1. TSS

5.2.5.1.1. FCS_DTLSS_EXT.1.1

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

5.2.5.1.2. FCS_DTLSS_EXT.1.3

The evaluator shall verify that the TSS describes how the DTLS Client IP address is validated prior to issuing a ServerHello message.

5.2.5.1.3. FCS_DTLSS_EXT.1.4

If using ECDHE or DHE ciphers, the evaluator shall verify that the TSS describes the key agreement parameters of the server Key Exchange message.

5.2.5.1.4. FCS_DTLSS_EXT.1.5

The evaluator shall verify that the TSS describes the actions that take place if a message received from the DTLS Client fails the MAC integrity check.

5.2.5.1.5. FCS_DTLSS_EXT.1.6

The evaluator shall verify that TSS describes how replay is detected and silently discarded for DTLS records that have previously been received and too old to fit in the sliding window.

5.2.5.1.6. FCS_DTLSS_EXT.1.7

The evaluator shall verify that the TSS describes if session resumption based on session IDs is supported (RFC 4346 and/or RFC 5246) and/or if session resumption based on session tickets is supported (RFC 5077).

If session tickets are supported, the evaluator shall verify that the TSS describes that the session tickets are encrypted using symmetric algorithms consistent with FCS_COP.1/DataEncryption. The evaluator shall verify that the TSS identifies the key lengths and algorithms used to protect session tickets.

If session tickets are supported, the evaluator shall verify that the TSS describes that session tickets adhere to the structural format provided in section 4 of RFC 5077 and if not, a justification shall be

given of the actual session ticket format.

If the TOE claims a (D)TLS server capable of session resumption (as a single context, or across multiple contexts), the evaluator verifies that the TSS describes how session resumption operates (i.e. what would trigger a full handshake, e.g. checking session status, checking Session ID, etc.). If multiple contexts are used the TSS describes how session resumption is coordinated across those contexts. In case session establishment and session resumption are always using a separate context, the TSS shall describe how the contexts interact with respect to session resumption (in particular regarding the session ID). It is acceptable for sessions established in one context to be resumable in another context.

5.2.5.2. Guidance Documentation

5.2.5.2.1. FCS_DTLSS_EXT.1.1

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

5.2.5.2.2. FCS_DTLSS_EXT.1.4

The evaluator shall verify that any configuration necessary to meet the requirement must be contained in the AGD guidance.

5.2.5.3. Tests

For clarification: For DTLS communication packets might be received in a different order than sent due to the use of the UDP protocol. All tests requiring a specific order of test steps ("before", "after") are therefore referring to the sequence numbering of DTLS packets.

5.2.5.3.1. FCS_DTLSS_EXT.1.1

Test 1: The evaluator shall establish a DTLS connection using each of the ciphersuites specified by the requirement. This connection may be established as part of the establishment of a higher-level application protocol, e.g., as part of a syslog session. It is sufficient to observe the successful negotiation of a ciphersuite to satisfy the intent of the test; it is not necessary to examine the characteristics of the encrypted traffic in an attempt to discern the ciphersuite being used (for example, that the cryptographic algorithm is 128-bit AES and not 256-bit AES).

Test 2: The evaluator shall send a Client Hello to the server with a list of ciphersuites that does not contain any of the ciphersuites in the server's ST and verify that the server denies the connection. Additionally, the evaluator shall send a Client Hello to the server containing only the TLS_NULL_WITH_NULL_NULL ciphersuite and verify that the server denies the connection.

Test 3: The evaluator shall perform the following modifications to the traffic:

a. Modify a byte in the Client Finished handshake message and verify that the server rejects the connection and does not send any application data.
b. (Test Intent: The intent of this test is to ensure that the server's TLS implementation immediately makes use of the key exchange and authentication algorithms to: a) Correctly

encrypt (D)TLS Finished message and b) Encrypt every (D)TLS message after session keys are negotiated.)

The evaluator shall use one of the claimed ciphersuites to complete a successful handshake and observe transmission of properly encrypted application data. The evaluator shall verify that no Alert with alert level Fatal (2) messages were sent.

The evaluator shall verify that the Finished message (Content type hexadecimal 16 and handshake message type hexadecimal 14) is sent immediately after the server's ChangeCipherSpec (Content type hexadecimal 14) message. The evaluator shall examine the Finished message (encrypted example in hexadecimal of a TLS record containing a Finished message, 16 03 03 00 40 11 22 33 44 55…) and confirm that it does not contain unencrypted data (unencrypted example in hexadecimal of a TLS record containing a Finished message, 16 03 03 00 40 14 00 00 0c…), by verifying that the first byte of the encrypted Finished message does not equal hexadecimal 14 for at least one of three test messages. There is a chance that an encrypted Finished message contains a hexadecimal value of '14' at the position where a plaintext Finished message would contain the message type code '14'. If the observed Finished message contains a hexadecimal value of '14' at the position where the plaintext Finished message would contain the message type code, the test shall be repeated three times in total. In case the value of '14' can be observed in all three tests it can be assumed that the Finished message has indeed been sent in plaintext and the test has to be regarded as 'failed'. Otherwise it has to be assumed that the observation of the value '14' has been due to chance and that the Finished message has indeed been sent encrypted. In that latter case the test shall be regarded as 'passed'.

5.2.5.3.2. FCS_DTLSS_EXT.1.3

Modify at least one byte in the cookie from the Server's HelloVerifyRequest message and verify that the Server rejects the Client's handshake message.

5.2.5.3.3. FCS_DTLSS_EXT.1.4

Test 1 [conditional]: If ECDHE ciphersuites are supported:

a. The evaluator shall repeat this test for each supported elliptic curve. The evaluator shall attempt a connection using a supported ECDHE ciphersuite and a single supported elliptic curve specified in the Elliptic Curves Extension. The evaluator shall verify (through a packet capture or instrumented client) that the TOE selects the same curve in the Server Key Exchange message and successfully establishes the connection.
b. The evaluator shall attempt a connection using a supported ECDHE ciphersuite and a single unsupported elliptic curve (e.g. secp192r1 (0x13)) specified in RFC4492, chap. 5.1.1. The evaluator shall verify that the TOE does not send a Server Hello message and the connection is not successfully established.

Test 2 [conditional]: If DHE ciphersuites are supported, the evaluator shall repeat the following test for each supported parameter size. If any configuration is necessary, the evaluator shall configure the TOE to use a supported Diffie-Hellman parameter size. The evaluator shall attempt a connection using a supported DHE ciphersuite. The evaluator shall verify (through a packet capture or instrumented client) that the TOE sends a Server Key Exchange Message where p Length is

consistent with the configured Diffie-Hellman parameter size(s).

Test 3 [conditional]: If RSA key establishment ciphersuites are supported, the evaluator shall repeat this test for each RSA key establishment key size. If any configuration is necessary, the evaluator shall configure the TOE to perform RSA key establishment using a supported key size (e.g. by loading a certificate with the appropriate key size). The evaluator shall attempt a connection using a supported RSA key establishment ciphersuite. The evaluator shall verify (through a packet capture or instrumented client) that the TOE sends a certificate whose modulus is consistent with the configured RSA key size.

5.2.5.3.4. FCS_DTLSS_EXT.1.5

The evaluator shall establish a connection using a client. The evaluator will then modify at least one byte in a record message and verify that the Server discards the record or terminates the DTLS session.

5.2.5.3.5. FCS_DTLSS_EXT.1.6

The evaluator shall set up a DTLS connection. The evaluator shall then capture traffic sent from the DTLS Client to the TOE. The evaluator shall retransmit copies of this traffic to the TOE in order to impersonate the DTLS Client. The evaluator shall observe that the TSF does not take action in response to receiving these packets and that the audit log indicates that the replayed traffic was discarded.

5.2.5.3.6. FCS_DTLSS_EXT.1.7

Test Objective: To demonstrate that the TOE will not resume a session for which the client failed to complete the handshake (independent of TOE support for session resumption)

Test 1 [conditional]: If the TOE does not support session resumption based on session IDs according to RFC4346 (TLS1.1) or RFC5246 (TLS1.2) or session tickets according to RFC5077, the evaluator shall perform the following test:

a. The client sends a Client Hello with a zero-length session identifier and with a SessionTicket extension containing a zero-length ticket.
b. The client verifies the server does not send a NewSessionTicket handshake message (at any point in the handshake).
c. The client verifies the Server Hello message contains a zero-length session identifier or passes the following steps:

   Note: The following steps are only performed if the ServerHello message contains a non-zero length SessionID.

d. The client completes the TLS handshake and captures the SessionID from the ServerHello.
e. The client sends a ClientHello containing the SessionID captured in step d). This can be done by keeping the TLS session from step d) open or by starting a new TLS session using the SessionID captured in step d).
f. The client verifies the TOE (1) implicitly rejects the SessionID by sending a ServerHello containing a different SessionID and by performing a full handshake (as shown in Figure 1 of

RFC 4346 or RFC 5246), or (2) terminates the connection in some way that prevents the flow of application data.

Remark: If multiple contexts are supported for session resumption, the session ID or session ticket may be obtained in one context for resumption in another context. It is possible that one or more contexts may only permit the construction of sessions to be reused in other contexts but not actually permit resumption themselves. For contexts which do not permit resumption, the evaluator is required to verify this behaviour subject to the description provided in the TSS. It is not mandated that the session establishment and session resumption share context. For example, it is acceptable for a control channel to establish and application channel to resume the session.

Test 2 [conditional]: If the TOE supports session resumption using session IDs according to RFC4346 (TLS1.1) or RFC5246 (TLS1.2), the evaluator shall carry out the following steps (note that for each of these tests, it is not necessary to perform the test case for each supported version of TLS):

a. The evaluator shall conduct a successful handshake and capture the TOE-generated session ID in the Server Hello message. The evaluator shall then initiate a new TLS connection and send the previously captured session ID to show that the TOE resumed the previous session by responding with ServerHello containing the same SessionID immediately followed by ChangeCipherSpec and Finished messages (as shown in figure 2 of RFC 4346 or RFC 5246).

b. The evaluator shall initiate a handshake and capture the TOE-generated session ID in the Server Hello message. The evaluator shall then, within the same handshake, generate or force an unencrypted fatal Alert message immediately before the client would otherwise send its ChangeCipherSpec message thereby disrupting the handshake. The evaluator shall then initiate a new Client Hello using the previously captured session ID, and verify that the server (1) implicitly rejects the session ID by sending a ServerHello containing a different SessionID and performing a full handshake (as shown in figure 1 of RFC 4346 or RFC 5246), or (2) terminates the connection in some way that prevents the flow of application data.

Remark: If multiple contexts are supported for session resumption, for each of the above test cases, the session ID may be obtained in one context for resumption in another context. There is no requirement that the session ID be obtained and replayed within the same context subject to the description provided in the TSS. All contexts that can reuse a session ID constructed in another context must be tested. It is not mandated that the session establishment and session resumption share context. For example, it is acceptable for a control channel to establish and application channel to resume the session.

Test 3 [conditional]: If the TOE supports session tickets according to RFC5077, the evaluator shall carry out the following steps (note that for each of these tests, it is not necessary to perform the test case for each supported version of TLS):

a. The evaluator shall permit a successful TLS handshake to occur in which a session ticket is exchanged with the non-TOE client. The evaluator shall then attempt to correctly reuse the previous session by sending the session ticket in the ClientHello. The evaluator shall confirm that the TOE responds with a ServerHello with an empty SessionTicket extension, NewSessionTicket, ChangeCipherSpec and Finished messages (as seen in figure 2 of RFC 5077).

b. The evaluator shall permit a successful TLS handshake to occur in which a session ticket is exchanged with the non-TOE client. The evaluator will then modify the session ticket and send it

as part of a new Client Hello message. The evaluator shall confirm that the TOE either (1) implicitly rejects the session ticket by performing a full handshake (as shown in figure 3 or 4 of RFC 5077), or (2) terminates the connection in some way that prevents the flow of application data.

Remark: If multiple contexts are supported for session resumption, for each of the above test cases, the session ticket may be obtained in one context for resumption in another context. There is no requirement that the session ticket be obtained and replayed within the same context subject to the description provided in the TSS. All contexts that can reuse a session ticket constructed in another context must be tested. It is not mandated that the session establishment and session resumption share context. For example, it is acceptable for a control channel to establish and application channel to resume the session.

5.2.6. FCS_SSHC_EXT.1 SSH Client

5.2.6.1. TSS

5.2.6.1.1. FCS_SSHC_EXT.1.2

The evaluator shall check to ensure that the TSS contains a list of the public key algorithms that are acceptable for use for user authentication and that this list is consistent with asymmetric key generation algorithms selected in FCS_CKM.1, hashing algorithms selected in FCS_COP.1/Hash, and signature generation algorithms selected in FCS_COP.1/SigGen. The evaluator shall confirm the TSS is unambiguous in declaring the TOE's ability to authenticate itself to a remote endpoint with a user-based public key.

If password-based authentication method has been selected in the FCS_SSHC_EXT.1.2, then the evaluator shall confirm it is also described in the TSS.

5.2.6.1.2. FCS_SSHC_EXT.1.3

The evaluator shall check that the TSS describes how "large packets" in terms of RFC 4253 are detected and handled.

5.2.6.1.3. FCS_SSHC_EXT.1.4

The evaluator shall check the description of the implementation of this protocol in the TSS to ensure that optional characteristics are specified, and the encryption algorithms supported are specified as well. The evaluator shall check the TSS to ensure that the encryption algorithms specified are identical to those listed for this component.

5.2.6.1.4. FCS_SSHC_EXT.1.5

The evaluator shall confirm the TSS describes how a host-key public key (i.e., SSH server's public key) is associated with the server identity.

The evaluator shall check the description of the implementation of this protocol in the TSS to ensure that optional characteristics are specified, and the host-key public key algorithms supported by the TOE are specified as well. The evaluator shall check the TSS to ensure that the host-key public key algorithms specified are identical to those listed for this component.

If x509v3-based public key authentication algorithms are claimed, the evaluator shall confirm that the TSS includes the description of how the TOE establishes the server's identity and how this identity is confirmed with the one that is presented in the provided certificate. For example, the TOE could verify that a server's configured IP address matches the one presented in the server's x.509v3 certificate.

5.2.6.1.5. FCS_SSHC_EXT.1.6

The evaluator shall check the TSS to ensure that it lists the supported data integrity algorithms, and that the list corresponds to the list in this component.

5.2.6.1.6. FCS_SSHC_EXT.1.7

The evaluator shall check the TSS to ensure that it lists the supported key exchange algorithms, and that the list corresponds to the list in this component.

5.2.6.1.7. FCS_SSHC_EXT.1.8

The evaluator shall check that the TSS specifies the following:

a. Both thresholds are checked by the TOE.
b. Rekeying is performed upon reaching the threshold that is hit first.

5.2.6.2. Guidance Documentation

5.2.6.2.1. FCS_SSHC_EXT.1.2

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 initiated by the TOE.

5.2.6.2.2. FCS_SSHC_EXT.1.4

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

5.2.6.2.3. FCS_SSHC_EXT.1.5

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

5.2.6.2.4. FCS_SSHC_EXT.1.6

The evaluator shall also check the guidance documentation to ensure that it contains instructions to the Security Administrator on how to ensure that only the allowed data integrity algorithms are used in SSH connections with the TOE (specifically, that the "none" MAC algorithm is not allowed).

5.2.6.2.5. FCS_SSHC_EXT.1.7

The evaluator shall also check the guidance documentation to ensure that it contains instructions to the Security Administrator on how to ensure that only the allowed key exchange algorithms are used in SSH connections with the TOE.

5.2.6.2.6. FCS_SSHC_EXT.1.8

If one or more thresholds that are checked by the TOE to fulfil the SFR are configurable, then the evaluator shall check that the guidance documentation describes how to configure those thresholds. Either the allowed values are specified in the guidance documentation and must not exceed the limits specified in the SFR (one hour of session time, one gigabyte of transmitted traffic) or the TOE must not accept values beyond the limits specified in the SFR. The evaluator shall check that the guidance documentation describes that the TOE reacts to the first threshold reached.

5.2.6.3. Tests

5.2.6.3.1. FCS_SSHC_EXT.1.2

Test objective: The purpose of these tests is to check the authentication of the client to the server using each claimed authentication method.

Test 1: For each claimed public-key authentication method, the evaluator shall configure the TOE to present a public key corresponding to that authentication method (e.g., 2048-bit RSA key when using ssh-rsa public key). The evaluator shall establish sufficient separate SSH connections with an appropriately configured remote non-TOE SSH server to demonstrate the use of all claimed public key algorithms. It is sufficient to observe the successful completion of the SSH Authentication Protocol to satisfy the intent of this test.

Test 2: [Conditional] If password-based authentication method has been selected in the FCS_SSHC_EXT.1.2, then following the guidance documentation the evaluator shall configure the TOE to perform password-based authentication with a remote SSH server to demonstrate that the TOE can successfully authenticate using a password as an authentication method.

5.2.6.3.2. FCS_SSHC_EXT.1.3

The evaluator shall demonstrate that if the TOE receives a packet larger than that specified in this component, that packet is dropped.

5.2.6.3.3. FCS_SSHC_EXT.1.4

The evaluator must ensure that only claimed ciphers and cryptographic primitives are used to establish an SSH connection. To verify this, the evaluator shall start session establishment for an SSH connection with a remote server (referred to as ‘remote endpoint’ below). 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 all the ciphers defined in the TSS for the TOE for SSH sessions, but no additional ones compared to the definition in the TSS. The evaluator shall perform one successful negotiation of an SSH session to verify that the TOE behaves as expected. It is sufficient to observe the successful negotiation of the session to satisfy the intent of the test. If the evaluator detects that not all ciphers defined in the TSS for SSH are supported by

the TOE and/or the TOE supports one or more additional ciphers not defined in the TSS for SSH, the test shall be regarded as failed.

5.2.6.3.4. FCS_SSHC_EXT.1.5

Test 1: The evaluator shall establish an SSH connection using each of the public key algorithms specified by the requirement to authenticate an SSH server to the TOE. It is sufficient to observe (on the wire) the successful negotiation of the algorithm to satisfy the intent of the test. Test objective: The purpose of this positive test is to check the authentication of the server by the client (when establishing the transport layer connection), and not for checking generation of the authentication message from the client (in the User Authentication Protocol). The evaluator shall therefore establish sufficient separate SSH connections (with an appropriately configured server) to cause the TOE to demonstrate use of all public key algorithms claimed in FCS_SSHC_EXT.1.5 in the ST.

Test 2: The evaluator shall configure an SSH server to only allow a public key algorithm that is not included in the ST selection. The evaluator shall attempt to establish an SSH connection from the TOE to the SSH server and observe that the connection is rejected.

5.2.6.3.5. FCS_SSHC_EXT.1.6

Test 1: [conditional, if an HMAC or AEAD_AES_*_GCM algorithm is selected in the ST] The evaluator shall establish an SSH connection using each of the algorithms, except “implicit”, specified by the requirement. It is sufficient to observe (on the wire) the successful negotiation of the algorithm to satisfy the intent of the test.

Note: To ensure the observed algorithm is used, the evaluator shall ensure a non-aes*-gcm@openssh.com encryption algorithm is negotiated while performing this test.

Test 2: [conditional, if an HMAC or AEAD_AES_*_GCM algorithm is selected in the ST] The evaluator shall configure an SSH server to only allow a MAC algorithm that is not included in the ST selection. The evaluator shall attempt to connect from the TOE to the SSH server and observe that the attempt fails.

Note: To ensure the proposed MAC algorithm is used, the evaluator shall ensure a non-aes*-gcm@openssh.com encryption algorithm is negotiated while performing this test.

5.2.6.3.6. FCS_SSHC_EXT.1.7

Test 1: The evaluator shall configure an SSH server to permit all allowed key exchange methods. The evaluator shall attempt to connect from the TOE to the SSH server using each allowed key exchange method and observe that each attempt succeeds.

5.2.6.3.7. FCS_SSHC_EXT.1.8

The evaluator needs to perform testing that rekeying is performed according to the description in the TSS. The evaluator shall test both, the time-based threshold and the traffic-based threshold.

For testing of the time-based threshold, the evaluator shall use the TOE to connect to an SSH server and keep the session open until the threshold is reached. The evaluator shall verify that the SSH session has been active longer than the threshold value and shall verify that the TOE initiated a rekey (the method of verification shall be reported by the evaluator).

Testing does not necessarily have to be performed with the threshold configured at the maximum allowed value of one hour of session time, but the value used for testing shall not exceed one hour. The evaluator needs to ensure that the rekeying has been initiated by the TOE and not by the SSH server the TOE is connected to.

For testing of the traffic-based threshold the evaluator shall use the TOE to connect to an SSH server and shall transmit data to and/or receive data from the TOE within the active SSH session until the threshold for data protected by either encryption key is reached. It is acceptable if the rekey occurs before the threshold is reached (e.g. because the traffic is counted according to one of the alternatives given in the Application Note for FCS_SSHC_EXT.1.8).

The evaluator shall verify that more data has been transmitted within the SSH session than the threshold allows and shall verify that the TOE initiated a rekey (the method of verification shall be reported by the evaluator).

Testing does not necessarily have to be performed with the threshold configured at the maximum allowed value of one gigabyte of transferred traffic, but the value used for testing shall not exceed one gigabyte. The evaluator needs to ensure that the rekeying has been initiated by the TOE and not by the SSH server the TOE is connected to.

If one or more thresholds that are checked by the TOE to fulfil the SFR are configurable, the evaluator needs to verify that the threshold(s) can be configured as described in the guidance documentation and the evaluator needs to test that modification of the thresholds is restricted to Security Administrators (as required by FMT_MOF.1).

In cases where data transfer threshold could not be reached due to hardware limitations it is acceptable to omit testing of this (SSH rekeying based on data transfer threshold) threshold if both the following conditions are met:

a. An argument is present in the TSS section describing this hardware-based limitation and
b. All hardware components that are the basis of such argument are definitively identified in the ST. For example, if specific Ethernet Controller or WiFi radio chip is the root cause of such limitation, these chips must be identified.

5.2.6.3.8. FCS_SSHC_EXT.1.9

Test 1: The evaluator shall delete all entries in the TOE's list of recognized SSH server host keys and, if selected, all entries in the TOE's list of trusted certification authorities. The evaluator shall initiate a connection from the TOE to an SSH server. The evaluator shall ensure that the TOE either rejects the connection or displays the SSH server's public key (either the key bytes themselves or a hash of the key using any allowed hash algorithm) and prompts the Security Administrator to accept or deny the key before continuing the connection.

Test 2: The evaluator shall add an entry associating a host name with a public key into the TOE's local database. The evaluator shall replace, on the corresponding SSH server, the server's host key with a different host key. If 'password-based' is selected for the TOE in FCS_SSHC_EXT.1.2, the evaluator shall initiate a connection from the TOE to the SSH server using password-based authentication, shall ensure that the TOE rejects the connection, and shall ensure that the password was not transmitted to the SSH server (for example, by instrumenting the SSH server with a

debugging capability to output received passwords). If 'password-based' is not selected for the TOE in FCS_SSHC_EXT.1.2, the evaluator shall initiate a connection from the TOE to the SSH server using public key-based authentication and shall ensure that the TOE rejects the connection.

5.2.7. FCS_SSHS_EXT.1 SSH Server

5.2.7.1. TSS

5.2.7.1.1. FCS_SSHS_EXT.1.2

The evaluator shall check to ensure that the TSS contains a description of the public key algorithms that are acceptable for use for authentication and that this list conforms to FCS_SSHS_EXT.1.5. and ensure that if password-based authentication methods have been selected in the ST then these are also described.

5.2.7.1.2. FCS_SSHS_EXT.1.3

The evaluator shall check that the TSS describes how “large packets” in terms of RFC 4253 are detected and handled.

5.2.7.1.3. FCS_SSHS_EXT.1.4

The evaluator shall check the description of the implementation of this protocol in the TSS to ensure that optional characteristics are specified, and the encryption algorithms supported are specified as well. The evaluator shall check the TSS to ensure that the encryption algorithms specified are identical to those listed for this component.

5.2.7.1.4. FCS_SSHS_EXT.1.5

The evaluator shall check the description of the implementation of this protocol in the TSS to ensure that optional characteristics are specified, and the public key algorithms supported are specified as well. The evaluator shall check the TSS to ensure that the public key algorithms specified are identical to those listed for this component.

The evaluator shall confirm that the TSS includes the description of how the TOE establishes a user identity when an SSH client presents a public key or X.509v3 certificate. For example, the TOE could verify that the SSH client’s presented public key matches one that is stored within the SSH server’s authorized_keys file.

5.2.7.1.5. FCS_SSHS_EXT.1.6

The evaluator shall check the TSS to ensure that it lists the supported data integrity algorithms, and that the list corresponds to the list in this component.

5.2.7.1.6. FCS_SSHS_EXT.1.7

The evaluator shall check the TSS to ensure that it lists the supported key exchange algorithms, and that the list corresponds to the list in this component.

5.2.7.1.7. FCS_SSHS_EXT.1.8

The evaluator shall check that the TSS specifies the following:

a. Both thresholds are checked by the TOE.

b. Rekeying is performed upon reaching the threshold that is hit first.

5.2.7.2. Guidance Documentation

5.2.7.2.1. FCS_SSHS_EXT.1.4

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

5.2.7.2.2. FCS_SSHS_EXT.1.5

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

5.2.7.2.3. FCS_SSHS_EXT.1.6

The evaluator shall check the guidance documentation to ensure that it contains instructions to the Security Administrator on how to ensure that only the allowed data integrity algorithms are used in SSH connections with the TOE (specifically, that the “none” MAC algorithm is not allowed).

5.2.7.2.4. FCS_SSHS_EXT.1.7

The evaluator shall check the guidance documentation to ensure that it contains instructions to the Security Administrator on how to ensure that only the allowed key exchange algorithms are used in SSH connections with the TOE.

5.2.7.2.5. FCS_SSHS_EXT.1.8

If one or more thresholds that are checked by the TOE to fulfil the SFR are configurable, then the evaluator shall check that the guidance documentation describes how to configure those thresholds. Either the allowed values are specified in the guidance documentation and must not exceed the limits specified in the SFR (one hour of session time, one gigabyte of transmitted traffic) or the TOE must not accept values beyond the limits specified in the SFR. The evaluator shall check that the guidance documentation describes that the TOE reacts to the first threshold reached.

5.2.7.3. Tests

5.2.7.3.1. FCS_SSHS_EXT.1.2

Test 1: If password-based authentication methods have been selected in the ST then using the guidance documentation, the evaluator shall configure the TOE to accept password-based authentication and demonstrate that user authentication succeeds when the correct password is provided by the user.

Test 2: If password-based authentication methods have been selected in the ST then the evaluator shall use an SSH client, enter an incorrect password to attempt to authenticate to the TOE, and demonstrate that the authentication fails. Note: Public key authentication is tested as part of testing for FCS_SSHS_EXT.1.5.

5.2.7.3.2. FCS_SSHS_EXT.1.3

The evaluator shall demonstrate that if the TOE receives a packet larger than that specified in this component, that packet is dropped.

5.2.7.3.3. FCS_SSHS_EXT.1.4

The evaluator must ensure that only claimed ciphers and cryptographic primitives are used to establish an SSH connection. To verify this, the evaluator shall start session establishment for an SSH connection from a remote client (referred to as 'remote endpoint' below). 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 all the ciphers defined in the TSS for the TOE for SSH sessions, but no additional ones compared to the definition in the TSS. The evaluator shall perform one successful negotiation of an SSH session to verify that the TOE behaves as expected. It is sufficient to observe the successful negotiation of the session to satisfy the intent of the test. If the evaluator detects that not all ciphers defined in the TSS for SSH are supported by the TOE and/or the TOE supports one or more additional ciphers not defined in the TSS for SSH, the test shall be regarded as failed.

5.2.7.3.4. FCS_SSHS_EXT.1.5

Test 1: The evaluator shall establish an SSH connection using each of the public key algorithms specified by the requirement to authenticate the TOE to an SSH client. It is sufficient to observe (on the wire) the successful negotiation of the algorithm to satisfy the intent of the test.

Test 2: The evaluator shall choose one public key algorithm supported by the TOE. The evaluator shall generate a new key pair for that algorithm without configuring the TOE to recognize the public key for authentication. The evaluator shall use an SSH client to attempt to connect to the TOE with the new key pair and demonstrate that authentication fails. Test objective: The purpose of this negative test is to verify that the server rejects authentication attempts of clients that present a public key that does not match public key(s) associated by the TOE with the identity of the client (i.e. the public keys are unknown to the server). To demonstrate correct functionality, it is sufficient to determine that an SSH connection was not established after using a valid username and an unknown key of supported type.

Test 3: The evaluator shall configure an SSH client to only allow a public key algorithm that is not included in the ST selection. The evaluator shall attempt to establish an SSH connection from the SSH client to the TOE and observe that the connection is rejected.

5.2.7.3.5. FCS_SSHS_EXT.1.6

Test 1: [conditional, if an HMAC or AEAD_AES_*_GCM algorithm is selected in the ST] The evaluator shall establish an SSH connection using each of the algorithms, except "implicit", specified by the requirement. It is sufficient to observe (on the wire) the successful negotiation of the algorithm to

satisfy the intent of the test.

Note: To ensure the observed algorithm is used, the evaluator shall ensure a non-aes*-gcm@openssh.com encryption algorithm is negotiated while performing this test.

Test 2: [conditional, if an HMAC or AEAD_AES_*_GCM algorithm is selected in the ST] The evaluator shall configure an SSH client to only allow a MAC algorithm that is not included in the ST selection. The evaluator shall attempt to connect from the SSH client to the TOE and observe that the attempt fails.

Note: To ensure the proposed MAC algorithm is used, the evaluator shall ensure a non-aes*-gcm@openssh.com encryption algorithm is negotiated while performing this test.

5.2.7.3.6. FCS_SSHS_EXT.1.7

Test 1: The evaluator shall configure an SSH client to only allow the diffie-hellman-group1-sha1 key exchange. The evaluator shall attempt to connect from the SSH client to the TOE and observe that the attempt fails.

Test 2: For each allowed key exchange method, the evaluator shall configure an SSH client to only allow that method for key exchange, attempt to connect from the client to the TOE, and observe that the attempt succeeds.

5.2.7.3.7. FCS_SSHS_EXT.1.8

The evaluator needs to perform testing that rekeying is performed according to the description in the TSS. The evaluator shall test both, the time-based threshold and the traffic-based threshold.

For testing of the time-based threshold, the evaluator shall use an SSH client to connect to the TOE and keep the session open until the threshold is reached. The evaluator shall verify that the SSH session has been active longer than the threshold value and shall verify that the TOE initiated a rekey (the method of verification shall be reported by the evaluator).

Testing does not necessarily have to be performed with the threshold configured at the maximum allowed value of one hour of session time, but the value used for testing shall not exceed one hour. The evaluator needs to ensure that the rekeying has been initiated by the TOE and not by the SSH client that is connected to the TOE.

For testing of the traffic-based threshold the evaluator shall use the TOE to connect to an SSH client and shall transmit data to and/or receive data from the TOE within the active SSH session until the threshold for data protected by either encryption key is reached. It is acceptable if the rekey occurs before the threshold is reached (e.g. because the traffic is counted according to one of the alternatives given in the Application Note for FCS_SSHS_EXT.1.8).

The evaluator shall verify that more data has been transmitted within the SSH session than the threshold allows and shall verify that the TOE initiated a rekey (the method of verification shall be reported by the evaluator).

Testing does not necessarily have to be performed with the threshold configured at the maximum allowed value of one gigabyte of transferred traffic, but the value used for testing shall not exceed one gigabyte. The evaluator needs to ensure that the rekeying has been initiated by the TOE and not

by the SSH client that is connected to the TOE.

If one or more thresholds that are checked by the TOE to fulfil the SFR are configurable, the evaluator needs to verify that the threshold(s) can be configured as described in the guidance documentation and the evaluator needs to test that modification of the thresholds is restricted to Security Administrators (as required by FMT_MOF.1/Functions).

In cases where data transfer threshold could not be reached due to hardware limitations it is acceptable to omit testing of this (SSH rekeying based on data transfer threshold) threshold if both the following conditions are met:

a. An argument is present in the TSS section describing this hardware-based limitation and
b. All hardware components that are the basis of such argument are definitively identified in the ST. For example, if specific Ethernet Controller or WiFi radio chip is the root cause of such limitation, these chips must be identified.

5.2.8. FCS_HTTPS_EXT.1 HTTPS selected

5.2.8.1. TSS

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

5.2.8.2. Guidance Documentation

The evaluator shall examine the guidance documentation to verify it instructs the Administrator how to configure TOE for use as an HTTPS client or HTTPS server.

5.2.8.3. Tests

Tests are performed in conjunction with the TLS evaluation activities.

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.

5.2.9. FCS_COP.1/KeyedHash Cryptographic Operation (Keyed Hash Algorithm)

5.2.9.1. TSS

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.

5.2.9.2. Guidance Documentation

The evaluator shall verify that the AGD guidance instructs the administrator how to configure the TOE to use the values used by the HMAC function: key length, hash function used, block size, and output MAC length used defined in the Security Target supported by the TOE for keyed hash function.

5.2.9.3. Tests

Note: The tests detailed below are not required to be performed for cryptographic functions implemented in the Root of Trust for Secure Boot (FPT_SBT_EXT.1).

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.

5.2.10. FIA_PSK_EXT.1 Pre-Shared Key Composition

5.2.10.1. TSS

The evaluator shall examine the TSS to ensure that it states that text-based pre-shared keys of 22 characters are supported, and that the TSS states the conditioning that takes place to transform the text-based pre-shared key from the key sequence entered by the user (e.g., ASCII representation) to the bit string used by IPsec, and that this conditioning is consistent with the first selection in the FIA_PSK_EXT.1.3 requirement. If the assignment is used to specify conditioning, the evaluator will confirm that the TSS describes this conditioning.

The evaluator shall also examine the TSS to ensure it describes the process by which the bit-based pre-shared keys are generated (if the TOE supports this functionality), and confirm that this process uses the RBG specified in FCS_RBG_EXT.1.

5.2.10.2. Guidance Documentation

The evaluator shall examine the operational guidance to determine that it provides guidance on the composition of strong text-based pre-shared keys, and (if the selection indicates keys of various lengths can be entered) that it provides information on the merits of shorter or longer pre-shared keys. The guidance must specify the allowable characters for pre-shared keys, and that list must be a super-set of the list contained in FIA_PSK_EXT.1.2.

If “bit-based pre-shared keys” is selected, the evaluator shall confirm the operational guidance contains instructions for either entering bit-based pre-shared keys for each protocol identified in the requirement, or generating a bit-based pre-shared key (or both).

5.2.10.3. Tests

The evaluator shall also perform the following tests:

1. The evaluator shall compose at least 15 pre-shared keys of 22 characters that cover all allowed characters in various combinations that conform to the operational guidance, and demonstrates that a successful protocol negotiation can be performed with each key.
2. [conditional]: If the TOE supports pre-shared keys of multiple lengths, the evaluator shall repeat Test 1 using the minimum length; the maximum length; and an invalid length. The minimum and maximum length tests should be successful, and the invalid length must be rejected by the TOE.
3. [conditional]: If the TOE supports bit-based pre-shared keys but does not generate such keys, the

evaluator shall obtain a bit-based pre-shared key of the appropriate length and enter it according to the instructions in the operational guidance. The evaluator shall then demonstrate that a successful protocol negotiation can be performed with the key.

4. [conditional]: If the TOE supports bit-based pre-shared keys and does generate such keys, the evaluator shall generate a bit-based pre-shared key of the appropriate length and use it according to the instructions in the operational guidance. The evaluator shall then demonstrate that a successful protocol negotiation can be performed with the key.

5.3. Passphrase-based Key Entry

5.3.1. FCS_PCC_EXT.1 Cryptographic Password Construct and Conditioning

5.3.1.1. TSS

The evaluator shall ensure the TSS describes the manner in which the TOE enforces the construction of passwords, including the length, and requirements on characters (number and type). The TSS also provides a description of how the password is conditioned and the evaluator ensures it satisfies the requirement.

5.3.1.2. KMD

The evaluator shall examine the KMD to ensure that the formation of the BEV and intermediary keys is described and that the key sizes match that selected by the ST Author.

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

5.3.1.3. Tests

The evaluator shall also perform the following tests:

Test 1: Ensure that the TOE supports passwords/passphrases of a minimum length of 64 characters.

Test 2: If the TOE supports a password/passphrase length up to a maximum number of characters, n (which would be greater than 64), then ensure that the TOE will not accept more than n characters.

Test 3: Ensure that the TOE supports passwords consisting of all characters assigned and supported by the ST.

5.3.2. FCS_KDF_EXT.1 Cryptographic Key Derivation

5.3.2.1. TSS

The evaluator shall verify the TSS includes a description of the key derivation function and shall

verify the key derivation uses an approved derivation mode and key expansion algorithm according to SP800-108, SP800-132, or ISO/IEC 11770-6:2016.

5.3.2.2. KMD

The evaluator shall examine the vendor's KMD to ensure that all keys used are derived using an approved method and a description of how and when the keys are derived.

5.3.2.3. Tests

The evaluator shall include test cases of FCS_KDF_EXT.1 to the test subset. Note that FCS_KDF_EXT.1 may be not mapped to the specific interface(s) after evaluator's analysis during ADV_FSP.1.

The evaluator shall produce test documentation for test cases of FCS_KDF_EXT.1. If there is no explicit external interface(s) mapped to FCS_KDF_EXT.1, the evaluator shall employ an alternative test approach (refer to CEM, section 15.2.2.).

The evaluator shall perform the following tests or witness respective tests executed by the developer if technically possible, otherwise an analysis of the implementation representation has to be performed.

Preconditions for testing

1. Specification of input parameter to the key derivation function to be tested
2. Specification of further required input parameters
3. Access to derived key(s)

The below tests are derived from Key Derivation using Pseudorandom Functions (SP 800-108) Validation System (KBKDFVS), Updated 4 January 2016, Section 6.2, from the National Institute of Standards and Technology.

The evaluator shall perform one or more of the following tests to verify the correctness of the key derivation function, depending on the mode(s) that are supported:

Counter Mode Tests

The evaluator shall determine the following characteristics of the key derivation function:

1. One or more pseudorandom functions (PRFs) that are included in the 'key derivation algorithm' selection in the SFR, and their output lengths in bits (h)
2. One or more of the values {8, 16, 24, 32} that equal the length of the binary representation of the counter (r), and the location of the counter relative to the fixed input data: before, after, or in the middle. If the counter is in the middle then the lengths of data before and after the counter must be determined
3. the 'key size' selections in the SFR, i.e the lengths (in bits) of the derived keying material (L)

For each supported combination of PRF, counter location, value of r, and value of L, the evaluator shall generate 20 pseudorandom key derivation key values (K_i).

For each value of K_I, the evaluator shall supply this data to the TOE in order to produce the keying material output K_O. The evaluator shall verify that the resulting output matches the results from submitting the same inputs to a known-good implementation of the key derivation function, having the same characteristics.

Feedback Mode Tests

The evaluator shall determine the following characteristics of the key derivation function:

1. One or more pseudorandom functions (PRFs) that are included in the 'key derivation algorithm' selection in the SFR, and their output lengths in bits (h)
2. If the implementation includes a counter then one or more of the values {8, 16, 24, 32} that equal the length of the binary representation of the counter (r), and the location of the counter relative to the fixed input data: before, after, or in the middle. If the counter is in the middle then the lengths of data before and after the counter must be determined
3. the 'key size' selections in the SFR, i.e the lengths (in bits) of the derived keying material (L)
4. the supported IV lengths.

For each supported combination of PRF, counter location (of a counter is used), value of r (if a counter is used), value of L, and IV length, the evaluator shall generate 20 pseudorandom key derivation key values (K_I).

For each value of K_I, the evaluator shall supply this data to the TOE in order to produce the keying material output K_O. The evaluator shall verify that the resulting output matches the results from submitting the same inputs to a known-good implementation of the key derivation function, having the same characteristics.

Double Pipeline Iteration Mode Tests

The evaluator shall determine the following characteristics of the key derivation function:

1. One or more pseudorandom functions (PRFs) that are included in the 'key derivation algorithm' selection in the SFR, and their output lengths in bits (h)
2. If the implementation includes a counter then one or more of the values {8, 16, 24, 32} that equal the length of the binary representation of the counter (r), and the location of the counter relative to the fixed input data: before, after, or in the middle. If the counter is in the middle then the lengths of data before and after the counter must be determined
3. the 'key size' selections in the SFR, i.e the lengths (in bits) of the derived keying material (L)

For each supported combination of PRF, counter location (if a counter is used), value of r (if a counter is used), and value of L, the evaluator shall generate 20 pseudorandom key derivation key values (K_I).

For each value of K_I, the evaluator shall supply this data to the TOE in order to produce the keying material output K_O. The evaluator shall verify that the resulting output matches the results from submitting the same inputs to a known-good implementation of the key derivation function, having the same characteristics.

5.3.3. FCS_COP.1/CMAC Cryptographic Operation (for cipher-based message authentication)

5.3.3.1. TSS

If HMAC was selected: 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.

If CMAC was selected: The evaluator shall examine the TSS to ensure that it specifies the following values used by the CMAC function: key length, block cipher used, block size (of the cipher), and output MAC length used.

5.3.3.2. Guidance Documentation

There are no AGD evaluation activities for this SFR.

5.3.3.3. KMD

There are no KMD evaluation activities for this SFR.

5.3.3.4. Tests

Note: The tests detailed below are not required to be performed for cryptographic functions implemented in the Root of Trust for Secure Boot (FPT_SBT_EXT.1).

If HMAC was selected: 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 using a known good implementation.

If CMAC was selected: For each of the supported parameter sets, the evaluator shall compose at least 15 sets of test data. Each set shall consist of a key and message data. The test data shall include messages of different lengths, some with partial blocks as the last block and some with full blocks as the last block. The test data keys shall include cases for which subkey K1 is generated both with and without using the irreducible polynomial R_b, as well as cases for which subkey K2 is generated from K1 both with and without using the irreducible polynomial R_b. (The subkey generation and polynomial R_b are as defined in SP800-38E.) The evaluator shall have the TSF generate CMAC tags for these sets of test data. The resulting MAC tags shall be compared to the result of generating CMAC tags with the same key using a known good implementation.

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

5.3.4.1. TSS

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

The evaluator shall ensure the TSS describes how nonces are created uniquely and how IVs and

tweaks are handled (based on the AES mode). The evaluator shall confirm that the nonces are unique and the IVs and tweaks meet the stated requirements.

5.4. Identification and Authentication (FIA)

5.4.1. Authentication using X.509 certificates (FIA_X509_EXT)

5.4.1.1. FIA_X509_EXT.1 X.509 Certificate Validation

5.4.1.1.1. TSS

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) 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).

The TSS shall describe when revocation checking is performed and on what certificates. If the revocation checking during authentication is handled differently depending on whether a full certificate chain or only a leaf certificate is being presented, any differences must be summarized in the TSS section and explained in the Guidance.

5.4.1.1.2. Guidance Documentation

The evaluator shall also ensure that the guidance documentation describes where the check of validity of the certificates takes place, describes 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) and describes how certificate revocation checking is performed and on which certificate.

5.4.1.1.3. Tests

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. 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 FIA_X509_EXT.1. These tests must be repeated for each distinct security function that utilizes X.509v3 certificates. For example, if the TOE implements certificate-based authentication with IPSEC and TLS, then it shall be tested with each of these protocols:

a. 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).

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.

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

c. 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.

d. 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.

e. 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.)

f. Test 6: The evaluator shall modify any byte in the certificate signatureValue field (see RFC5280 Sec. 4.1.1.3), which is normally the last field in the certificate, and demonstrate that the certificate fails to validate. (The signature on the certificate will not validate.)

g. 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.)

   The following tests are run when a minimum certificate path length of three certificates is implemented.

h. Test 8: (Conditional on support for EC certificates as indicated in FCS_COP.1/SigGen). The evaluator shall conduct the following tests:

   Test 8a: (Conditional on TOE ability to process CA certificates presented in certificate message) The test shall be designed in a way such that only the EC root certificate is designated as a trust anchor, and by setting up the trust store in a way that the EC 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 EC root CA certificate in the trust store). The evaluator shall present the TOE with a valid chain of EC certificates (terminating in a trusted CA certificate), where the elliptic curve parameters are specified as a named curve. The evaluator shall confirm that the TOE validates the certificate chain.

Test 8b: (Conditional on TOE ability to process CA certificates presented in certificate message) The test shall be designed in a way such that only the EC root certificate is designated as a trust anchor, and by setting up the trust store in a way that the EC 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 EC root CA certificate in the trust store). The evaluator shall present the TOE with a chain of EC certificates (terminating in a trusted CA certificate), where the intermediate certificate in the certificate chain uses an explicit format version of the Elliptic Curve parameters in the public key information field, and is signed by the trusted EC root CA, but having no other changes. The evaluator shall confirm the TOE treats the certificate as invalid.

Test 8c: The evaluator shall establish a subordinate CA certificate, where the elliptic curve parameters are specified as a named curve, that is signed by a trusted EC root CA. The evaluator shall attempt to load the certificate into the trust store and observe that it is accepted into the TOE's trust store. The evaluator shall then establish a subordinate CA certificate that uses an explicit format version of the elliptic curve parameters, and that is signed by a trusted EC root CA. The evaluator shall attempt to load the certificate into the trust store and observe that it is rejected, and not added to the TOE's trust store.

The evaluator shall perform the following tests for FIA_X509_EXT.1.2. The tests described must be performed in conjunction with the other certificate services assurance activities, including the functions in FIA_X509_EXT.2.1. The 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.

The goal of the following tests is to verify that the TOE accepts a certificate as a CA certificate only if it has been marked as a CA certificate by using basicConstraints with the CA flag set to True (and implicitly tests that the TOE correctly parses the basicConstraints extension as part of X509v3 certificate chain validation).

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).

a. 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).
b. 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).

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. But there is no need to repeat the tests for each separate TLS channel/path in FTP_ITC.1, FTP_TRP.1/Admin and FTP_TRP.1/NonAdmin (unless the channels/paths use separate implementations of TLS).

5.4.1.2. FIA_X509_EXT.2 X.509 Certificate Authentication

5.4.1.2.1. TSS

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

The evaluator shall examine the TSS to confirm that it describes the behaviour 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.

5.4.1.2.2. Guidance Documentation

The evaluator shall also ensure that the guidance documentation describes the configuration required in the operating environment so the TOE can use the certificates. The guidance documentation shall also include any required configuration on the TOE to use the certificates. The guidance document shall also describe the steps for the Security Administrator to follow if the connection cannot be established during the validity check of a certificate used in establishing a trusted channel.

5.4.1.2.3. Tests

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

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.

5.4.1.3. FIA_X509_EXT.3 X509 Certificate Requests

5.4.1.3.1. TSS

If the ST author selects "device-specific information", the evaluator shall verify that the TSS contains a description of the device-specific fields used in certificate requests.

5.4.1.3.2. Guidance Documentation

The evaluator shall check to ensure that the guidance documentation contains instructions on requesting certificates from a CA, including generation of a Certificate Request. If the ST author selects "Common Name", "Organization", "Organizational Unit", or "Country", the evaluator shall ensure that this guidance includes instructions for establishing these fields before creating the Certification Request.

5.4.1.3.3. Tests

The evaluator shall perform the following tests:

a. Test 1: The evaluator shall use the guidance documentation to cause the TOE to generate a Certification Request. The evaluator shall capture the generated message and ensure that it conforms to the format specified. The evaluator shall confirm that the Certification Request provides the public key and other required information, including any necessary user-input information.

b. Test 2: The evaluator shall demonstrate that validating a response message to a Certification Request without a valid certification path results in the function failing. The evaluator shall then load a certificate or certificates as trusted CAs needed to validate the certificate response message and demonstrate that the function succeeds.

Chapter 6. Evaluation Activities for SARs

The sections below specify EAs for the Security Assurance Requirements (SARs) included in the related cPPs. The EAs in Chapter 2, *Evaluation Activities for SFRs*, Chapter 3, *Evaluation Activities for Conditionally Mandatory Requirements*, Chapter 4, *Evaluation Activities for Optional Requirements*, and Chapter 5, *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.

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. Class ASE: Security Target

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 Chapter 2, *Evaluation Activities for SFRs* as well as the EAs for the conditionally-mandatory, optional, and selection-based SFRs claimed by the ST and specified in Chapter 3, *Evaluation Activities for Conditionally Mandatory Requirements*, Chapter 4, *Evaluation Activities for Optional Requirements*, and Chapter 5, *Evaluation Activities for Selection-Based Requirements*.

6.2. Class ADV: Development

6.2.1. Basic Functional Specification (ADV_FSP.1)

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 Chapter 2, *Evaluation Activities for SFRs*, Chapter 3, *Evaluation Activities for Conditionally Mandatory Requirements*, Chapter 4, *Evaluation Activities for Optional Requirements*, Chapter 5, *Evaluation Activities for Selection-Based Requirements*, and in EAs for AGD, ATE and AVA SARs in other parts of Chapter 6, *Evaluation Activities for SARs*.

The EAs are reworded (indicated with italicization) 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.

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.

Table 2. Mapping of ADV_FSP.1 CEM Work Units to Evaluation Activities

CEM ADV_FSP.1 Work Units	Evaluator 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.	Section 6.2.2, "ADV_FSP.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.	Section 6.2.3, "ADV_FSP.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 parameters associated with each SFR-enforcing and SFR supporting TSFI.	Section 6.2.4, "ADV_FSP.1-3 Evaluation Activity": *The evaluator shall check the interface documentation to ensure it 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 categorisation of interfaces as SFR-non-interfering to determine that it is accurate.	Paragraph 609 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.	Section 6.2.5, "ADV_FSP.1-5 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 Chapter 2, *Evaluation Activities for SFRs*, and, if applicable, Chapter 3, *Evaluation Activities for Conditionally Mandatory Requirements*, Chapter 4, *Evaluation Activities for Optional Requirements* and Chapter 5, *Evaluation Activities for Selection-Based Requirements*, 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.

CEM ADV_FSP.1 Work Units	Evaluator Activities
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 Chapter 2, *Evaluation Activities for SFRs*, and, if applicable, Chapter 3, *Evaluation Activities for Conditionally Mandatory Requirements*, Chapter 4, *Evaluation Activities for Optional Requirements* and Chapter 5, *Evaluation Activities for Selection-Based Requirements*, 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.

6.2.2. ADV_FSP.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.

In this context, TSFI are deemed security relevant if they are used by the administrator to configure the TOE, or to perform other administrative functions (e.g., audit review or performing updates). Additionally, those interfaces that are identified in the ST, or guidance documentation, as adhering to the security policies (as presented in the SFRs), are also considered security relevant. The intent, is that these interfaces will be adequately tested, and having an understanding of how these interfaces are used in the TOE is necessary to ensure proper test coverage is applied.

The set of TSFI that are provided as evaluation evidence are contained in the Administrative Guidance and User Guidance.

6.2.3. ADV_FSP.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.

In this context, TSFI are deemed security relevant if they are used by the administrator to configure the TOE, or to perform other administrative functions (e.g., audit review or performing updates). Additionally, those interfaces that are identified in the ST, or guidance documentation, as adhering to the security policies (as presented in the SFRs), are also considered security relevant. The intent, is that these interfaces will be adequately tested, and having an understanding of how these interfaces are used in the TOE is necessary to ensure proper test coverage is applied.

The set of TSFI that are provided as evaluation evidence are contained in the Administrative Guidance and User Guidance.

6.2.4. ADV_FSP.1-3 Evaluation Activity

The evaluator shall check the interface documentation to ensure it identifies and describes the parameters for each TSFI that is identified as being security relevant.

6.2.5. ADV_FSP.1-5 Evaluation Activity

The evaluator shall examine the interface documentation to develop a mapping of the interfaces to SFRs.

The evaluator uses the provided documentation and first identifies, and then examines a representative set of interfaces to perform the EAs presented in Chapter 2 and, if applicable, Chapter 3, *Evaluation Activities for Conditionally Mandatory Requirements*, Chapter 4, *Evaluation Activities for Optional Requirements* and Chapter 5, *Evaluation Activities for Selection-Based Requirements*, including the EAs associated with testing of the interfaces.

It should be noted that there may be some SFRs that do not have an interface that is explicitly "mapped" to invoke the desired functionality. For example, generating a random bit string, destroying a cryptographic key that is no longer needed, or the TSF failing to a secure state, are capabilities that may be specified in SFRs, but are not invoked by an interface.

However, if the evaluator is unable to perform some other required EA because there is insufficient design and interface information, then the evaluator is entitled to conclude that an adequate functional specification has not been provided, and hence that the verdict for the ADV_FSP.1 assurance component is a 'fail'.

6.3. Class AGD: Guidance Documentation

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)

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.

In addition, the evaluator performs the EAs specified below.

6.3.1.1. Evaluation Activity

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

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

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 that use of other cryptographic engines was not evaluated nor tested during the CC evaluation of the TOE.

6.3.1.4. Evaluation Activity

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.

6.3.1.5. Evaluation Activity

In addition the evaluator shall ensure that the following requirements are also met.

1. The documentation must describe the process for verifying updates to the TOE by verifying a digital signature. The evaluator shall verify that this process includes the following steps:
    a. Instructions for obtaining the update itself. This should include instructions for making the update accessible to the TOE (e.g., placement in a specific directory).
    b. Instructions for initiating the update process, as well as discerning whether the process was successful or unsuccessful. This includes instructions that describe at least one method of validating the hash/digital signature.
2. The TOE will likely contain security functionality that does not fall in the scope of evaluation under this cPP. The guidance documentation shall make it clear to an administrator which security functionality is covered by the EAs.

6.3.2. Preparative Procedures (AGD_PRE.1)

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.

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.

In addition, the evaluator performs the EAs specified below.

6.3.2.1. Evaluation Activity

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).

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

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

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

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.

6.3.2.5. Evaluation Activity

In addition the evaluator shall ensure that the following requirements are also met.

The preparative procedures must

1. Include instructions to provide a protected administrative capability; and
2. Identify TOE passwords that have default values associated with them and instructions shall be provided for how these can be changed.

6.4. Class ALC: Life-cycle Support

6.4.1. Labelling of the TOE (ALC_CMC.1)

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)

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

6.5. Class ATE: Tests

6.5.1. Independent Testing - Conformance (ATE_IND.1)

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.

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 Chapter 2, *Evaluation Activities for SFRs*, Chapter 3, *Evaluation Activities for Conditionally Mandatory Requirements*, Chapter 4, *Evaluation Activities for Optional Requirements*, and Chapter 5, *Evaluation Activities for Selection-Based Requirements*.

The evaluator should consult Appendix B, *Equivalency Considerations* when determining the appropriate strategy for testing multiple variations or models of the TOE that may be under evaluation.

6.6. Class AVA: Vulnerability Assessment

6.6.1. Vulnerability Survey (AVA_VAN.1)

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.

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.

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

CEM AVA_VAN.1 Work Units	Evaluator 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. *If the iTC specifies any tools to be used in performing this analysis in section A.1.4, the following text is also included in this cell: "The calibration of test resources specified in paragraph 1467 of the CEM applies to the tools listed in Appendix A, Section A.1.4."*

CEM AVA_VAN.1 Work Units	Evaluator Activities
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.	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 Appendix A, Section A.1	Replace CEM work unit with activities outlined in Appendix A, 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 Appendix A, section A.1, and documentation as specified in Appendix A, 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 Appendix A, section A.2.

CEM AVA_VAN.1 Work Units	Evaluator Activities
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 Appendix A, 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 Appendix A, Section A.3 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.

CEM AVA_VAN.1 Work Units	Evaluator Activities
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 Appendix A, 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 Appendix A, 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 Appendix A, Section A.2, paragraph 110.
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 Appendix A, Section A.3.

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)

In addition to the activities specified by the CEM in accordance with Table 3 above, the evaluator shall perform the following activities.

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.

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.

Chapter 7. Required Supplementary Information

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.

The supplementary information required by this SD are:

- Entropy Documentation, which is evaluated against the guidance specified in Appendix E of [HCDcPP].
- A Key Management Description (KMD), which is evaluated against guidance specified in Appendix F of [HCDcPP] and all relevant KMD Evaluation Activities in this SD.

Chapter 8. Evaluation Activity

The evaluator formulates hypotheses in accordance with process defined in Appendix A. 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.

Chapter 9. References

- [2600] IEEE Std. 2600™-2008 “IEEE Standard for Information Technology: Hardcopy Device and System Security”
- [2600.1] IEEE Std. 2600.1™-2009 “IEEE Standard for a Protection Profile in Operational Environment A”
- [610.12] IEEE Std 610.12-1990 “IEEE Standard Glossary of Software Engineering Terminology”
- [8802-6] ISO /IEC 8802-6:1994 “Information technology – Telecommunications and information exchange between systems – Local and metropolitan area networks – Specific requirements – Part 6”
- [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.
- [addenda] CC and CEM addenda, Exact Conformance, Selection-Based SFRs, Optional SFRs, Version 0.5, May 2017
- [HCDcPP] collaborative Protection Profile for Hardcopy Devices, 1.0e-DRAFT, 17 January 2024

Appendix A: Vulnerability Analysis

A.1. Sources of Vulnerability Information

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 derived from public sources as documented in Section A.1.1, “Type 1 Hypotheses - Public-Vulnerability-based”-this fixed set has been agreed to 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, “Type 1 Hypotheses - Public-Vulnerability-based” 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 contained 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, “Type 2 Hypotheses - iTC-sourced”;
3. A list of flaw hypotheses derived from information available to the evaluators; this includes the baseline evidence provided by the vendor described in this Supporting Document (documentation associated with EAs, documentation described in Section 6.6.1.1, “Evaluation Activity (Documentation)”, <the iTC can remove the reference to Section 6.6.1.1, “Evaluation Activity (Documentation)” if no additional documentation is defined> documentation described in Chapter 7, *Required Supplementary Information*), as well as other information (public and/or based on evaluator experience) as documented in Section A.1.3, “Type 3 Hypotheses - Evaluation-Team-Generated”; and
4. A list of flaw hypotheses that are generated through the use of iTC-defined tools (e.g., nmap, protocol testers) and their application is specified in Section A.1.4, “Type 4 Hypotheses - Tool-Generated”.

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

The following list of public sources of vulnerability information was selected by the iTC:

a. Search Common Vulnerabilities and Exposures: http://cve.mitre.org/cve/
b. Search the National Vulnerability Database: https://nvd.nist.gov/
c. Search US-CERT: http://www.kb.cert.org/vuls/html/search

The list of sources above was searched with the search terms selected by evaluation laboratories.

In order to supplement this list, the evaluators shall also perform a search on the sources listed above 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.

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

The following list of flaw hypothesis generated by the iTC for this technology must be considered by the evaluation team as flaw hypotheses in performing the vulnerability assessment.

There are currently no iTC-sourced flaw hypotheses. A future revision of the HCD cPP may update this section for relevant findings made by evaluation laboratories.

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

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.

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

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

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.

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.

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).

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.

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

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.

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).

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.

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.

Any testing performed by the evaluator shall be documented in the test report as outlined in Section A.3, “Reporting” below.

As indicated in Section A.3, “Reporting”, 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, “Sources of Vulnerability Information”) flaw hypotheses only. The fact that the iTC generates these candidate hypotheses indicates these must be addressed.

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, "Reporting". 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

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.

The public-facing report contains:

- The terms and sources used when the procedures for searching public sources were followed according to instructions in the Supporting Document per Section A.1.1, "Type 1 Hypotheses - Public-Vulnerability-based";
- A statement that the evaluators have examined the Type 1 flaw hypotheses specified in this Supporting Document in Section A.1.1, "Type 1 Hypotheses - Public-Vulnerability-based" (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, "Type 2 Hypotheses - iTC-sourced".
- A statement that the evaluation team developed Types 3 and 4 flaw hypotheses in accordance with Sections Section A.1.3, "Type 3 Hypotheses - Evaluation-Team-Generated", Section A.1.4, "Type 4 Hypotheses - Tool-Generated", and Section A.2, "Process for Evaluator Vulnerability Analysis", 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.

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

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_VAN1-10); 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).

Appendix B: Equivalency Considerations

B.1. Introduction

This appendix provides a foundation for evaluators to determine whether a vendor's request for equivalency of products is allowed.

For the purpose of this evaluation, equivalency can be broken into two categories:

- **Variations in models**: Separate TOE models/variations may include differences that could necessitate separate testing across each model. If there are no variations in any of the categories listed below, the models may be considered equivalent.
- **Variations in TOE dependencies on the environment (e.g., OS/platform the product is tested on)**: The method a TOE provides functionality (or the functionality itself) may vary depending upon the environment on which it is installed. If there is no difference in the TOE-provided functionality or in the manner in which the TOE provides the functionality, the models may be considered equivalent.

Determination of equivalency for each of the above specified categories can result in several different testing outcomes.

If a set of TOE are determined to be equivalent, testing may be performed on a single variation of the TOE. However, if the TOE variations have security-relevant functional differences, each of the TOE models that exhibits either functional or structural differences must be separately tested. Generally speaking, only the difference between each variation of TOE must be separately tested. Other equivalent functionality may be tested on a representative model and not across multiple platforms.

If it is determined that a TOE operates the same regardless of the environment, testing may be performed on a single instance for all equivalent configurations. However, if the TOE is determined to provide environment-specific functionality, testing must take place in each environment for which a difference in functionality exists. Similar to the above scenario, only the functionality affected by environment differences must be retested.

If a vendor disagrees with the evaluator's assessment of equivalency, the Scheme arbitrates between the two parties whether equivalency exists.

B.2. Evaluator guidance for determining equivalence

B.2.1. Strategy

When performing the equivalency analysis, the evaluator should consider each factor independently. A factor may be any number of things at various levels of abstraction, ranging from the processor a device uses, to the underlying operating system and hardware platform a software application relies upon. Examples may be the various chip sets employed by the product, the type of network interface (different device drivers), storage media (solid state drive, spinning disk, EEPROM). It is important to consider how the difference in these factors may influence the TOE's

ability to enforce the SFRs. Each analysis of an individual factor will result in one of two outcomes:

- For the particular factor, all variations of the TOE on all supported platforms are equivalent. In this case, testing may be performed on a single model in a single test environment and cover all supported models and environments.
- For the particular factor, a subset of the product has been identified to require separate testing to ensure that it operates identically to all other equivalent TOEs. The analysis would identify the specific combinations of models/testing environments that needed to be tested.

Complete CC testing of the product would encompass the totality of each individual analysis performed for each of the identified factors.

B.2.2. Guidance for hardcopy devices

The following table provides a description of how an evaluator should consider each of the factors that affect equivalency between TOE model variations and across operating environments. Additionally, the table also identifies scenarios that will result in additional separate testing across models.

Table 4. Evaluation Equivalency Analysis

Factor	Same/Not Same	Evaluator guidance
Platform/Hardware Dependencies	Independent	If there are no identified platform/hardware dependencies, the evaluator shall consider testing on multiple hardware platforms to be equivalent.
	Dependencies	If there are specified differences between platforms/hardware, the evaluator must identify if the differences affect the cPP-specified security functionality or if they apply to non-cPP-specified functionality. If functionality specified in the cPP is dependent upon platform/hardware provided services, the product must be tested on each of the different platforms to be considered validated on that particular hardware combination. In these cases, the evaluator has the option of only retesting the functionality dependent upon the platform/hardware provided functionality. If the differences only affect non-cPP-specified functionality, the variations may still be considered equivalent. For each difference the evaluator must provide an explanation of why the difference does or does not affect cPP-specified functionality.

Factor	Same/Not Same	Evaluator guidance
Differences in TOE Software Binaries	Identical	If the model binaries are identical, the model variations shall be considered equivalent.
	Different	If there are differences between model software binaries, a determination must be made if the differences affect cPP-specified security functionality. If cPP-specified functionality is affected, the models are not considered equivalent and must be tested separately. The evaluator has the option of only retesting the functionality that was affected by the software differences. If the differences only affect non-PP specified functionality, the models may still be considered equivalent. For each difference the evaluator must provide an explanation of why the difference does or does not affect cPP specified functionality.

Factor	Same/Not Same	Evaluator guidance
Differences in Libraries Used to Provide TOE Functionality	Same	If there are no differences between the libraries used in various TOE models, the model variations shall be considered equivalent.
	Different	If the separate libraries are used between model variations, a determination of whether the functionality provided by the library affects cPPspecified functionality must be made. If cPPspecified functionality is affected, the models are not considered equivalent and must be tested separately. The evaluator has the option of only retesting the functionality that was affected by the differences in the included libraries. If the different libraries only affect non-PP specified functionality, the models may still be considered equivalent. For each different library, the evaluator must provide an explanation of why the different libraries do or do not affect cPP specified functionality.

Factor	Same/Not Same	Evaluator guidance
TOE Management Interface Differences	Consistent	If there are no differences in the management interfaces between various TOE models, the model variations shall be considered equivalent.
	Differences	If the product provides separate interfaces based on the model variation, a determination must be made of whether cPP-specified functionality can be configured by the different interfaces. If the interface differences affect cPP-specified functionality, the variations are not considered equivalent and must be separately tested. The evaluator has the option of only retesting the functionality that can be configured by the different interfaces (and the configuration of said functionality). If the different management interfaces only affect non-PP specified functionality, the models may still be considered equivalent. For each management interface difference, the evaluator must provide an explanation of why the different management interfaces do or do not affect cPP specified functionality.

Factor	Same/Not Same	Evaluator guidance
TOE Functional Differences	Identical	If the functionality provided by different TOE model variation is identical, the models variations shall be considered equivalent.
	Different	If the functionality provided by different TOE model variations differ, a determination must be made if the functional differences affect cPPspecified functionality. If cPP-specific functionality differs between models, the models are not considered equivalent and must be tested separately. In these cases, the evaluator has the option of only retesting the functionality that differs model-to-model. If the functional differences only affect non-cPP specified functionality, the model variations may still be considered equivalent. For each difference the evaluator must provide an explanation of why the difference does or does not affect cPP specified functionality.

B.2.3. Test presentation/Truth in advertising

In addition to determining what to test, the evaluation results and resulting validation report must identify the actual module and testing environment combinations that have been tested. The analysis used to determine the testing subset may be considered proprietary and will only optionally be publicly included.

Appendix C: Glossary

For definitions of standard CC terminology see [CC1].

Administrator

A User who has been specifically granted the authority to manage some portion or all of the TOE and whose actions may affect the security policies of the TOE. Administrators may possess special privileges that provide capabilities to override portions of security policies. [2600.1]

Confidential (TSF) Data

Assets for which either disclosure or alteration by a User who is not an Administrator or the owner of the data would have an effect on the operational security of the TOE. [2600.1]

DEK

A key used to encrypt data at rest.

External Authentication

Identification and authentication mechanism that uses services of External IT Entities to authenticate TOE Users.

Hardcopy Device

A system producing or utilizing a physical embodiment of an electronic document or image. These systems include printers, scanners, fax machines, digital copiers, MFPs (multifunction peripherals), MFDs (multifunction devices), “all-in-ones” and other similar products.[2600]

Intermediate Key

A key used in a point between the initial user authorization and the DEK. [CPP_FDE_EE_V2.0]

Internal Authentication

Identification and authentication function that is wholly contained within the TOE.

Job

A document processing task submitted to the hardcopy device. A single processing task may process one or more documents. [2600.1]

Nonvolatile Storage Device

A device that provides computer storage of data that is not cleared when the power is turned off.

Operational Environment

Environment in which the TOE is operated.[CC]

Protection Profile

Implementation-independent statement of security needs for a TOE type. [CC]

Read

To access data from a storage device or data medium. (Note that in this case, the data medium may be a printed output, and therefore, release of a print job is a “read” operation) [610.12]

Security Assurance Requirement

A description of how assurance is to be gained that the TOE meets the SFRs. [CC]

Security Functional Requirement

A translation of the Security Objectives for the TOE into a standardized language. [CC]

Security Objective

Statement of an intent to counter identified Threats and/or satisfy identified organization security policies and/or Assumptions. [CC]

Security Target

Implementation-dependent statement of security needs for a specific identified TOE. [CC]

Submask

A submask is a bit string that can be generated and stored in a numbers of ways, such as passphrases, tokens, etc. [CPP_FDE_EE_V2.0]

Target of Evaluation

Set of software, firmware and/or hardware possibly accompanied by guidance. [CC]

TOE Security Functionality

Combined functionality of all hardware, software, and firmware of a TOE that must be relied upon for the correct enforcement of the SFRs. [CC]

TSF Data

Data for the operation of the TOE upon which the enforcement of the SFR relies. [CC]

Unauthorized Access

Access to a resource that a User is not permitted to access.

User

Human or IT entity possibly interacting with the TOE from outside of the TOE boundary. [CC]

User Data

Data for the User that does not affect the operation of the TSF. [CC]

User Document Data

The Asset that consists of the information contained in a User's Document. This includes the original Document itself in either hardcopy or electronic form, image data, or residually stored data created by the hardcopy device while processing an original Document and printed hardcopy output. [2600.1]

Appendix D: Acronyms

Table 5. Acronyms

Acronym	Meaning
AAD	Additional Authenticated Data
AES-CBC	Advanced Encryption Standard (AES) Cipher Algorithm in Cipher Block Chaining (CBC) Mode
AES-CCM	Advanced Encryption Standard (AES) Cipher Algorithm in Counter with CBC-MAC (CCM) Mode
AES-GCM	Advanced Encryption Standard (AES) Cipher Algorithm in Galois/Counter Mode (GCM)
XTS-AES	Advanced Encryption Standard (AES) Cipher Algorithm in XEX-based tweaked-codebook mode with ciphertext stealing (XTS)
AES	Advanced Encryption Standard
AESAVS	The Advanced Encryption Standard Algorithm Validation Suite (AESAVS)
AGD	Administartor Guidance Documentation
ATA	Advanced Technology Attachment
ASCII	American Standard Code for Information Interchange
BEV	Border Encryption Value
CC	Common Criteria
CCM	Counter with CBC-MAC
CCMVS	The CCM Validation System
CEM	Common Evaluation Methodology
CMAC	Cipher-based Message Authentication Code
CMACVS	The CMAC Validation System (CMACVS)
CSP	Critical Security Parameter
CVE	Common Vulnerabilities and Exposures
cPP	collaborative Protection Profile
DEK	Data Encryption Key
DHE	Diffie-Hellman Ephemeral
DRBG	Deterministic Random Bit Generator
DSA2VS	The FIPS 186-4 Digital Signature Algorithm Validation System (DSA2VS)
EA	Evaluation Activity
ECDHE	Elliptic Curve Diffie-Hellman Ephemeral
ECDSA2VS	The FIPS 186-4 Elliptic Curve Digital Signature Algorithm Validation System (ECDSA2VS)

Acronym	Meaning
EEPROM	Electrically Erasable Programmable Read Only Memory
ETR	Evaluation Technical Report
ESP	Encapsulating Security Payload
FDE	Full-Disk Encryption
FIPS	Federal Information Processing Standards
GCM	Galois/Counter Mode
GCMVS	The Galois/Counter Mode (GCM) and GMAC Validation System (GCMVS) with the Addition of XPN Validation Testing
GMAC	Galois Message Authentication Code
HCD	Hardcopy Device
HMAC	Keyed-Hashing for Message Authentication
HMACVS	The Keyed-Hash Message Authentication Code Validation System (HMACVS)
IPsec	Internet Protocol Security
IT	Information Technology
iTC	international Technical Community
IV	Initialization Vector
I&A	Identification and Authentication
HTTPS	Hypertext Transfer Protocol Secure
IKE	Internet Key Exchange
KAT	Known Answer Test
KEM	Key Encapsulation Mechanism
KMD	Key Management Description
KTS	Key Transport Scheme
KWK	KEK Wrapping Key
KWS	Key Wrapping Scheme
LDAP	Lightweight Directory Access Protocol
MAC	Message Authentication Code
NAT	Network Address Translation
NIST	National Institute of Standards and Technology
NTP	Network Time Protocol
OAEP	Optimal Asymmetric Encryption Padding
OCSP	Online Certificate Status Protocol
OE	Operational Environment
PC	Personal Computer

Acronym	Meaning
PSTN	Public Switched Telephone Network
RAID	Redundant Array of Independent Disks
RBG	Random Bit Generator
RFC	Request for Comments
RSA	Rivest–Shamir–Adleman
RSA2VS	The 186-4 RSA Validation System (RSA2VS)
SAR	Security Assurance Requirement
SD	Supporting Document
SED	Self-Encrypting Drive
SFP	Security Function Policy
SFR	Security Functional Requirement
SHA	Secure Hash Algorithm
SHAVS	The Secure Hash Algorithm Validation System (SHAVS)
SOC	System on a Chip
SPD	Security Problem Definition
SSH	Secure Shell
ST	Security Target
TCP	Transmission Control Protocol
TLS	Transport Layer Security
TOE	Target of Evaluation
TPM	Trusted Platform Module
TSF	TOE Security Functionality
TSFI	TOE Security Functional Interface
TSS	TOE Summary Specification
UDP	User Datagram Protocol
VPN	Virtual Private Network
XPN	eXtended Packet Number
XTS	XEX-based tweaked-codebook mode with ciphertext stealing