SAP SE
SAP CommonCryptoLib Crypto Kernel
FIPS 140-3 Non-Proprietary Security Policy

Document version: 1.4
Date: 2025-10-21

Table of Contents

1 General 7
1.1 Overview 7
1.2 Security Levels 7
Cryptographic Module Specification 8
2.1 Description 8
2.2 Tested and Vendor Affirmed Module Version and Identification 9
2.3 Excluded Components 13
2.4 Modes of Operation 14
2.5 Algorithms 15
2.6 Security Function Implementations 23
2.7 Algorithm Specific Information 30
2.8 RBG and Entropy 32
2.9 Key Generation 34
2.10 Key Establishment 34
2.11 Industry Protocols 36
3 Cryptographic Module Interfaces 37
3.1 Ports and Interfaces 37
3.2 Trusted Channel Specification 37
3.3 Control Interface Not Inhibited 37
3.4 Additional Information 37
4 Roles, Services, and Authentication 38
4.1 Authentication Methods 38
4.2 Roles 38
4.3 Approved Services 38

4.4 Non-Approved Services 50
4.5 External Software/Firmware Loaded 54
4.6 Bypass Actions and Status 54
4.7 Cryptographic Output Actions and Status 55
5 Software/Firmware Security 56
5.1 Integrity Techniques 56
5.2 Initiate on Demand 56
6 Operational Environment 57
6.1 Operational Environment Type and Requirements 57
6.2 Configuration Settings and Restrictions 57
7 Physical Security 58
8 Non-Invasive Security 59
9 Sensitive Security Parameters Management 60
9.1 Storage Areas 60
9.2 SSP Input-Output Methods 60
9.3 SSP Zeroization Methods 61
9.4 SSPs 62
9.5 Transitions 72
10 Self-Tests 73
10.1 Pre-Operational Self-Tests 73
10.2 Conditional Self-Tests 73
10.3 Periodic Self-Test Information 78
10.4 Error States 81
10.5 Operator Initiation of Self-Tests 82
11 Life-Cycle Assurance 83

11.1 Installation, Initialization, and Startup Procedures 83
11.2 Administrator Guidance 85
11.3 Non-Administrator Guidance 85
11.4 Design and Rules 85
11.5 Maintenance Requirements 86
11.6 End of Life 86
12 Mitigation of Other Attacks 87
12.1 Attack List 87
12.2 Mitigation Effectiveness 87
12.3 Guidance and Constraints 87

List of Tables

Table 1: Security Levels 7
Table 2: Tested Module Identification – Software, Firmware, Hybrid (Executable Code Sets) 10
Table 3: Tested Operational Environments - Software, Firmware, Hybrid 12
Table 4: Optionally Available PAAs per Algorithm 13
Table 5: Modes List and Description 14
Table 6: Approved Algorithms - General 18
Table 7: Approved Algorithms - Legacy 18
Table 8: Approved Algorithms - CVL 19
Table 9: Vendor-Affirmed Algorithms 20
Table 10: Non-Approved, Not Allowed Algorithms 22
Table 11: Security Function Implementations 30
Table 12: Entropy Certificates 32
Table 13: Entropy Sources 32
Table 14: Obtained Assurances for the Implemented Approved KAS-SSC and KTS 35
Table 15: Ports and Interfaces 37
Table 16: Roles 38
Table 17: Approved Services 49
Table 18: Non-Approved Services 54
Table 19: Storage Areas 60
Table 20: SSP Input-Output Methods 60
Table 21: SSP Zeroization Methods 61
Table 22: SSP Table 1 68
Table 23: SSP Table 2 71
Table 24: Pre-Operational Self-Tests 73
Table 25: Conditional Self-Tests 78
Table 26: Pre-Operational Periodic Information 79
Table 27: Conditional Periodic Information 81
Table 28: Error States 81
Table 29: Error Causes and Expected Return Codes 82
Table 30: Module File Names and Checksums 83
Table 31: File Access Permissions 84

List of Figures

Figure 1: Block Diagram 9
Figure 2: Module in Context of its Operational Environment 9

1 General

1.1 Overview

This document is the non-proprietary FIPS 140-3 Security Policy for the cryptographic module “SAP CommonCryptoLib Crypto Kernel” (hereafter denoted as “module”) in its version 8.6.1 developed by SAP SE.

1.2 Security Levels

Section	Title	Security Level
1	General	1
2	Cryptographic module specification	1
3	Cryptographic module interfaces	1
4	Roles, services, and authentication	1
5	Software/Firmware security	1
6	Operational environment	1
7	Physical security	N/A
8	Non-invasive security	N/A
9	Sensitive security parameter management	1
10	Self-tests	1
11	Life-cycle assurance	1
12	Mitigation of other attacks	1
	Overall Level	1

Table 1: Security Levels

Cryptographic Module Specification

2.1 Description

Purpose and Use:

The module is a shared software library that implements various cryptographic functions such as encryption/decryption, signature generation/verification, key establishment, key generation, and random number generation. The module also implements an entropy source. It provides C/C++ APIs for key management and operation of the implemented cryptographic functions. The module itself is subdivided as shown in Figure 1. The arrows in this figure indicate the interactions between the different module components.

The module is used by a single operator, which is the linked application using the library by calling its functions and methods. Typically, this application is the “SAP CommonCryptoLib” (not in scope of the validation), which itself is a library providing cryptographic protocols and services to an application using it.

Module Type: Software

Module Embodiment: MultiChipStand

Cryptographic Boundary:

The cryptographic boundary of the module comprises the shared library (including its executable code and data as well as its runtime representation) and a file containing the reference value for the software/firmware integrity test (see Section 5 Software/Firmware Security and Section 11.1 Installation, Initialization, and Startup Procedures for details).

In Figure 2, the contents of the cryptographic boundary are indicated by the blue boxes. The “API” box shown in Figure 2 corresponds to the blue box shown in the block diagram in Figure 1. The linked application calling the module’s services and residing in the same shared volatile memory region is outside the cryptographic boundary.

Tested Operational Environment’s Physical Perimeter (TOEPP):

The TOEPP of the module is made up by the workstation, laptop, or server hardware the module is running on. Figure 2 shows the module and its interactions in context of the modifiable operational environment. The module interacts with the operating system within the TOEPP as well as the application it is linked to. At runtime, the module resides in the volatile memory provided by the TOEPP.

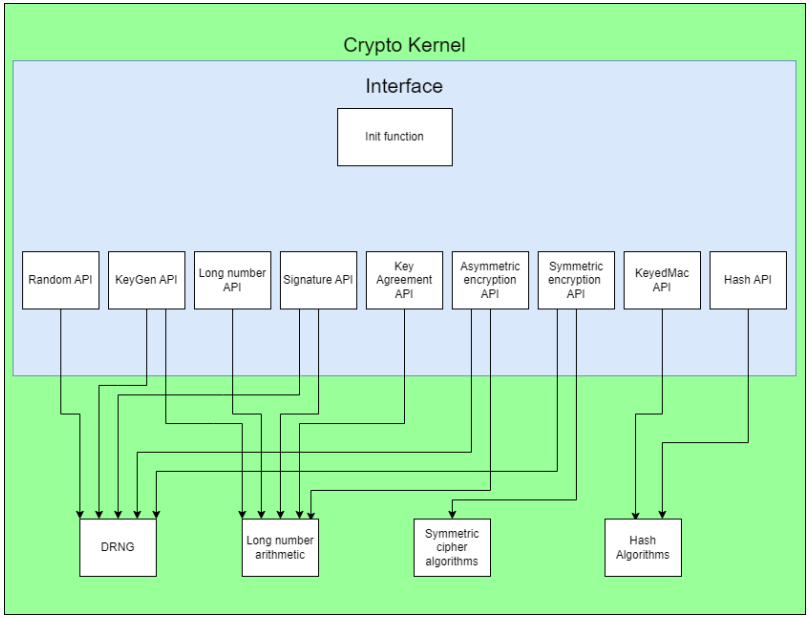

Figure 1: Block Diagram

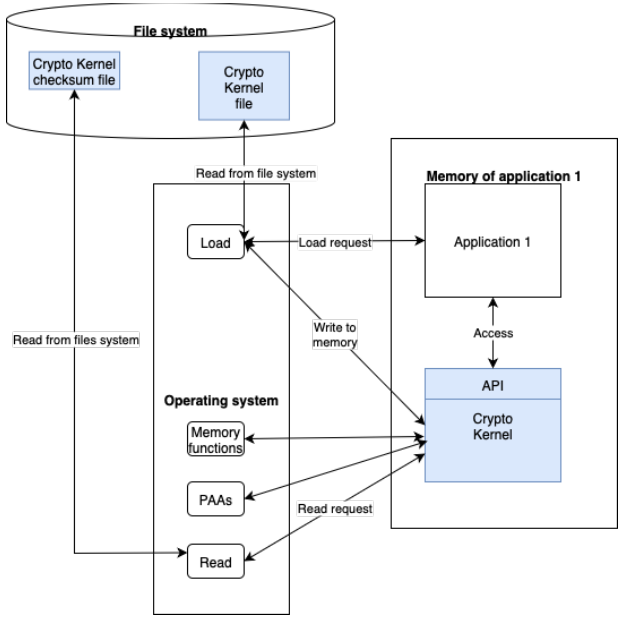

Figure 2: Module in Context of its Operational Environment

2.2 Tested and Vendor Affirmed Module Version and Identification

Tested Module Identification – Software, Firmware, Hybrid (Executable Code Sets):

Package or File Name	Software/ Firmware Version	Features	Integrity Test
aix-6.1-ppc-64	8.6.1 (aix-6.1-ppc-64)		HMAC-SHA2-256
aix-7.2-ppc-64	8.6.1 (aix-7.2-ppc-64)		HMAC-SHA2-256
hpux-b.11.31-ia-64	8.6.1 (hpux-b.11.31-ia-64)		HMAC-SHA2-256
linux-gcc-11.2-armv8-64	8.6.1 (linux-gcc-11.2-armv8-64)		HMAC-SHA2-256
linux-gcc-4.3-ia-64	8.6.1 (linux-gcc-4.3-ia-64)		HMAC-SHA2-256
linux-gcc-4.3-s390x-64	8.6.1 (linux-gcc-4.3-s390x-64)		HMAC-SHA2-256
linux-gcc-4.3-x86-64	8.6.1 (linux-gcc-4.3-x86-64)		HMAC-SHA2-256

Package or File Name	Software/ Firmware Version	Features	Integrity Test
linux-gcc-4.8-ppcle-64	8.6.1 (linux-gcc-4.8-ppcle-64)		HMAC-SHA2-256
linux-musl-1.2.4-x86-64	8.6.1 (linux-musl-1.2.4-x86-64)		HMAC-SHA2-256
macosx-arm-64	8.6.1 (macosx-arm-64)		HMAC-SHA2-256
macosx-x86-64	8.6.1 (macosx-x86-64)		HMAC-SHA2-256
sunos-5.10-sparc-64	8.6.1 (sunos-5.10-sparc-64)		HMAC-SHA2-256
sunos-5.10-x86-64	8.6.1 (sunos-5.10-x86-64)		HMAC-SHA2-256
windows-x86-64	8.6.1 (windows-x86-64)		HMAC-SHA2-256

Table 2: Tested Module Identification – Software, Firmware, Hybrid (Executable Code Sets)

The module is a shared library, i.e., it consists of software only. It provides an API in terms of C/C++ functions for operation of the implemented cryptographic functions and is delivered already compiled.

The module comprises two files:

- A shared library file that implements the module's cryptographic functionality.
- A text file that contains a hexadecimal representation of the SHA2-256 hash of, and the HMAC-SHA2-256 value computed over, the library file. This file is required for the integrity test performed during module's initialization.

Section 11.1 Installation, Initialization, and Startup Procedures provides a mapping between the unique package names listed in the above table for the module's executable code sets and the actual file names.

The version and name of the library can be queried separately using the `sec_crypto_get_feature_info` API (see also the service "show versioning information" in Section 4.3 Approved Services). The validated module has the version v8.6.1 and is called "SAP CommonCryptoLib Crypto Kernel". Note that the package names provided in brackets for the version numbers are only listed here for reference in the following tables. They are not part of the versioning information output by the module.

Tested Operational Environments - Software, Firmware, Hybrid:

Operating System	Hardware Platform	Processors	PAA/PAI	Hypervisor or Host OS	Version(s)
SLES 15 SP4	Amazon EC2 c7g.16xlarge	AWS Graviton3	Yes	AWS Nitro System	8.6.1 (linux-gcc-11.2-armv8-64)

Operating System	Hardware Platform	Processors	PAA/PAI	Hypervisor or Host OS	Version(s)
Apple macOS 14	MacBook Pro (2019)	Intel Core i7-9750H	Yes		8.6.1 (macosx-x86-64)
Apple macOS 12	Mac mini (2020)	Apple M1	Yes		8.6.1 (macosx-arm-64)
SunOS 5.10	Sun-4u - Fujitsu M4000	Fujitsu SPARC64-VI	No		8.6.1 (sunos-5.10-sparc-64)
HP-UX 11.31 (IA64)	Hewlett-Packard rx2800 i4	Intel Itanium Processor 9540	No		8.6.1 (hpux-b.11.31-ia-64)
SLES 15 SP2	IBM z13	IBM S390	No	IBM z/VM 7.2.0	8.6.1 (linux-gcc-4.3-s390x-64)
IBM AIX 7.2	IBM Power System S824	IBM POWER9	Yes	IBM PowerVM 3.1.4.21	8.6.1 (aix-7.2-ppc-64)
IBM AIX 6.1	IBM Power System S824	IBM POWER8	No	IBM PowerVM 3.1.4.21	8.6.1 (aix-6.1-ppc-64)
SLES 15 SP4	Ampere D12A-M1-AA	Arm Neoverse N1	Yes	QEMU (KVM) 6.2 on SLES 15 SP4	8.6.1 (linux-gcc-11.2-armv8-64)
SLES 11 SP1	Hewlett-Packard rx2660	Intel Itanium Processor 9120N	No		8.6.1 (linux-gcc-4.3-ia-64)
SLES 12 SP5	IBM Power System E980	IBM POWER8	Yes	IBM PowerVM 3.1.4.21	8.6.1 (linux-gcc-4.8-ppcle-64)
SunOS 5.10	i86pc - Fujitsu Primergy RX600 S6	Intel Xeon E7-4807	Yes		8.6.1 (sunos-5.10-x86-64)

Operating System	Hardware Platform	Processors	PAA/PAI	Hypervisor or Host OS	Version(s)
Alpine Linux 3.18.2	LENOVO_MT_20QU_BU_Think_FM_ThinkPad P1 Gen 2	Intel Core i9-9880H	Yes	VMware Workstation 17.5.0 on Windows 11 Enterprise	8.6.1 (linux-musl-1.2.4-x86-64)
SLES 15 SP5	Dell EMC PowerEdge R840	Intel Xeon Platinum 8260M	Yes	VMware ESXi 7.0.3	8.6.1 (linux-gcc-4.3-x86-64)
Microsoft Windows Server 2022 Standard	DELL EMC PowerEdge R840	Intel Xeon Platinum 8260M	Yes	VMware ESXi 7.0.3	8.6.1 (windows-x86-64)

Table 3: Tested Operational Environments - Software, Firmware, Hybrid

All operational environments listed above with “Yes” in the “PAA/PAI” column can optionally use what is considered as a Processor Algorithm Acceleration (PAA) for certain approved algorithm implementations. For details on the PAAs supported by each binary, please refer to Table 4 and its footnotes as well as the CAVP certificate referenced in Section 2.5 Algorithms. The PAAs are also used when the listed algorithms are embedded into other higher cryptographic algorithms (e.g., SHA-256 used as part of an HMAC).

Package	Algorithm				
	AES (all modes)	AES-GCM[1]	SHA-1	SHA2-224/256	SHA2-384/512
aix-6.1-ppc-64	P8	P8		P8	P8
aix-7.2-ppc-64	P8	P8		P8	P8
hpux-b.11.31-ia-64					
linux-gcc-11.2-armv8-64	AES		SHA1	SHA2	SHA512[2]
linux-gcc-4.3-ia-64					
linux-gcc-4.3-s390x-64					

[1] The PAAs listed in this column are used for the counter logic of AES-GCM. They are used in addition to the PAAs listed in the “AES (all modes)” column.

[2] Not supported by the tested operational environment using the Arm Neoverse-N1 CPU.

Package	Algorithm				
	AES (all modes)	AES-GCM[1]	SHA-1	SHA2-224/256	SHA2-384/512
linux-gcc-4.3-x86-64	AES-NI, SSSE3[3]	CLMUL			
linux-gcc-4.8-ppcle-64	P8	P8		P8	P8
linux-musl-1.2.4-x86-64	AES-NI, SSSE3[3]	CLMUL			
macosx-arm-64	AES		SHA1	SHA2	SHA512
macosx-x86-64	AES-NI, SSSE3[3]	CLMUL			
sunos-5.10-sparc-64					
sunos-5.10-x86-64	AES-NI, SSSE3[3]	CLMUL			
windows-x86-64	AES-NI, SSSE3[3]	CLMUL			

Table 4: Optionally Available PAAs per Algorithm

In compliance with IG 2.3.C, the module implements every algorithm utilizing a PAA also entirely in software. Different combinations of enabled and disabled PAAs were used during testing to cover all code paths of the implemented algorithms.

Before module initialization, the `crypt_disable_cpu_features` API can be used to configure the activated PAAs. Reconfiguration of the activated PAAs is only possible after re-initializing the module (see Section 11.1 Installation, Initialization, and Startup Procedures for more details).

Vendor-Affirmed Operational Environments - Software, Firmware, Hybrid:

No vendor-affirmed operational environments are claimed. Nevertheless, the module may be ported to other operational environments that have the necessary capabilities (operating system, system libraries, sufficient hardware, etc.) per the CMVP porting rules specified in the FIPS 140-3 Management Manual. However, in this case the CMVP makes no statement as to the correct operation of the module or the security strengths of the generated keys when so ported if the specific operational environment is not listed on the validation certificate. In addition, when running a module on such an untested platform the “No assurance of the minimum strength of generated SSPs (e.g., keys)” caveat applies per IG 9.3.A.

2.3 Excluded Components

There are no excluded components within the cryptographic boundary.

[3] The use of AES-NI and SSS3 for AES is mutually exclusive.

2.4 Modes of Operation

Modes List and Description:

Mode Name	Description	Type	Status Indicator
Approved mode	The module provides approved services as part of its regular operation.	Approved	Provided by each service as a return code. Services requiring an approved security service indicator per IG 2.4.C return the value RC_FIPS_APPROVED ("1").
Non-approved mode	The module provides non-approved services as part of its regular operation.	Non-Approved	Provided by each service as a return code. Non-approved services either return the value "0" or do not have a return code.

Table 5: Modes List and Description

As the module supports both an approved mode and non-approved modes, the caveat “when operated in approved mode” is applicable.

The module further supports a non-compliant test mode that is off by default. This mode allows for additional controls for functional testing that violate the FIPS 140-3 requirements. As the test mode is non-compliant, the caveat “when installed, initialized, and configured as specified in Section 11.1 of the Security Policy” is additionally applicable. When this mode is activated, the module is not considered to be FIPS 140-3 validated.

Mode Change Instructions and Status:

With the module’s default configuration, both approved and non-approved services are available at the same time. There is no transitioning procedure to switch between different modes. The return codes of the called security services indicate whether they were executed in the approved mode or the non-approved mode depending on the provided input parameters. For more details, please see Section 4.3 Approved Services.

The module’s non-compliant test mode can be activated by passing the “TESTMODE” value to the `crypt_control` API before module initialization. When this mode is activated, all the module’s security services, including those that are usually approved, are non-approved services. This is reflected by the implemented approved service indicator. The test mode can be deactivated by reloading the module or by passing the value “PRODUCTIONMODE” to the `crypt_control` API.

Degraded Mode Description:

The module does not support a degraded mode of operation.

2.5 Algorithms

Approved Algorithms:

General

Algorithm	CAVP Cert	Properties	Reference
AES-CBC	A5497	Direction - Decrypt, Encrypt Key Length - 128, 192, 256	SP 800-38A
AES-CBC-CS3	A5497	Direction - decrypt, encrypt Key Length - 128, 192, 256	SP 800-38A
AES-CFB128	A5497	Direction - Decrypt, Encrypt Key Length - 128, 192, 256	SP 800-38A
AES-CFB8	A5497	Direction - Decrypt, Encrypt Key Length - 128, 192, 256	SP 800-38A
AES-CTR	A5497	Direction - Decrypt, Encrypt Key Length - 128, 192, 256	SP 800-38A
AES-ECB	A5497	Direction - Decrypt, Encrypt Key Length - 128, 192, 256	SP 800-38A
AES-GCM	A5497	Direction - Decrypt, Encrypt IV Generation - Internal IV Generation Mode - 8.2.1, 8.2.2 Key Length - 128, 192, 256	SP 800-38D
AES-OFB	A5497	Direction - Decrypt, Encrypt Key Length - 128, 192, 256	SP 800-38A
Counter DRBG	A5497	Prediction Resistance - No Mode - AES-256 Derivation Function Enabled - Yes	SP 800-90A Rev. 1
DSA KeyGen (FIPS186-4)	A5497	L - 2048, 3072 N - 224, 256	FIPS 186-4
DSA PQGGen (FIPS186-4)	A5497	L - 2048, 3072 N - 224, 256 Hash Algorithm - SHA2-224, SHA2-256, SHA2-384, SHA2-512	FIPS 186-4
ECDSA KeyGen (FIPS186-5)	A5497	Curve - P-224, P-256, P-384, P-521 Secret Generation Mode - testing candidates	FIPS 186-5

Algorithm	CAVP Cert	Properties	Reference
ECDSA KeyVer (FIPS186-5)	A5497	Curve - P-224, P-256, P-384, P-521	FIPS 186-5
ECDSA SigGen (FIPS186-5)	A5497	Curve - P-224, P-256, P-384, P-521 Hash Algorithm - SHA2-224, SHA2-256, SHA2-384, SHA2-512, SHA3-224, SHA3-256, SHA3-384, SHA3-512 Component - No, Yes	FIPS 186-5
ECDSA SigVer (FIPS186-5)	A5497	Curve - P-224, P-256, P-384, P-521 Hash Algorithm - SHA2-224, SHA2-256, SHA2-384, SHA2-512, SHA3-224, SHA3-256, SHA3-384, SHA3-512	FIPS 186-5
EDDSA KeyGen	A5497	Curve - ED-25519, ED-448	FIPS 186-5
EDDSA KeyVer	A5497	Curve - ED-25519, ED-448	FIPS 186-5
EDDSA SigGen	A5497	Curve - ED-25519, ED-448	FIPS 186-5
EDDSA SigVer	A5497	Curve - ED-25519, ED-448	FIPS 186-5
HMAC-SHA-1	A5497	Key Length - Key Length: 8-524288 Increment 8	FIPS 198-1
HMAC-SHA2-224	A5497	Key Length - Key Length: 8-524288 Increment 8	FIPS 198-1
HMAC-SHA2-256	A5497	Key Length - Key Length: 8-524288 Increment 8	FIPS 198-1
HMAC-SHA2-384	A5497	Key Length - Key Length: 8-524288 Increment 8	FIPS 198-1
HMAC-SHA2-512	A5497	Key Length - Key Length: 8-524288 Increment 8	FIPS 198-1
HMAC-SHA3-224	A5497	Key Length - Key Length: 8-524288 Increment 8	FIPS 198-1
HMAC-SHA3-256	A5497	Key Length - Key Length: 8-524288 Increment 8	FIPS 198-1
HMAC-SHA3-384	A5497	Key Length - Key Length: 8-524288 Increment 8	FIPS 198-1
HMAC-SHA3-512	A5497	Key Length - Key Length: 8-524288 Increment 8	FIPS 198-1
KAS-ECC-SSC Sp800-56Ar3	A5497	Domain Parameter Generation Methods - P-224, P-256, P-384, P-521 Scheme - ephemeralUnified - KAS Role - initiator, responder staticUnified - KAS Role - initiator, responder	SP 800-56A Rev. 3
KAS-FFC-SSC Sp800-56Ar3	A5497	Domain Parameter Generation Methods - FB, FC, ffdhe2048, ffdhe3072, ffdhe4096, ffdhe6144, ffdhe8192, MODP-2048, MODP-3072, MODP-4096, MODP-6144, MODP-8192 Scheme -	SP 800-56A Rev. 3

Algorithm	CAVP Cert	Properties	Reference
		dhEphem - KAS Role - initiator, responder dhStatic - KAS Role - initiator, responder	
KTS-IFC	A5497	Modulo - 2048, 3072, 4096, 6144, 8192 Key Generation Methods - rsakpg1-crt Scheme - KTS-OAEP-basic - KAS Role - initiator, responder Key Transport Method - Key Length - 1024	SP 800-56B Rev. 2
RSA KeyGen (FIPS186-5)	A5497	Key Generation Mode - probable Modulo - 2048, 3072, 4096, 6144, 8192 Primality Tests - 2pow100 Private Key Format - crt	FIPS 186-5
RSA SigGen (FIPS186-5)	A5497	Modulo - 2048, 3072, 4096 Signature Type - pkcs1v1.5, pss	FIPS 186-5
RSA SigVer (FIPS186-5)	A5497	Modulo - 2048, 3072, 4096 Signature Type - pkcs1v1.5, pss	FIPS 186-5
Safe Primes Key Generation	A5497	Safe Prime Groups - ffdhe2048, ffdhe3072, ffdhe4096, ffdhe6144, ffdhe8192, MODP-2048, MODP-3072, MODP-4096, MODP-6144, MODP-8192	SP 800-56A Rev. 3
Safe Primes Key Verification	A5497	Safe Prime Groups - ffdhe2048, ffdhe3072, ffdhe4096, ffdhe6144, ffdhe8192, MODP-2048, MODP-3072, MODP-4096, MODP-6144, MODP-8192	SP 800-56A Rev. 3
SHA-1	A5497	Message Length - Message Length: 0-65536 Increment 8 Large Message Sizes - 1, 2, 4, 8	FIPS 180-4
SHA2-224	A5497	Message Length - Message Length: 0-65536 Increment 8 Large Message Sizes - 1, 2, 4, 8	FIPS 180-4
SHA2-256	A5497	Message Length - Message Length: 0-65536 Increment 8 Large Message Sizes - 1, 2, 4, 8	FIPS 180-4
SHA2-384	A5497	Message Length - Message Length: 0-65536 Increment 8 Large Message Sizes - 1, 2, 4, 8	FIPS 180-4

Algorithm	CAVP Cert	Properties	Reference
SHA2-512	A5497	Message Length - Message Length: 0-65536 Increment 8 Large Message Sizes - 1, 2, 4, 8	FIPS 180-4
SHA3-224	A5497	Message Length - Message Length: 0-65536 Increment 8 Large Message Sizes - 1, 2, 4, 8	FIPS 202
SHA3-256	A5497	Message Length - Message Length: 0-65536 Increment 8 Large Message Sizes - 1, 2, 4, 8	FIPS 202
SHA3-384	A5497	Message Length - Message Length: 0-65536 Increment 8 Large Message Sizes - 1, 2, 4, 8	FIPS 202
SHA3-512	A5497	Message Length - Message Length: 0-65536 Increment 8 Large Message Sizes - 1, 2, 4, 8	FIPS 202
SHAKE-128	A5497	Output Length - Output Length: 16-65536 Increment 8	FIPS 202
SHAKE-256	A5497	Output Length - Output Length: 16-65536 Increment 8	FIPS 202

Table 6: Approved Algorithms - General

Legacy

Algorithm	CAVP Cert	Properties	Reference
DSA PQGVer (FIPS186-4)	A5497	L - 1024, 2048, 3072 N - 160, 224, 256 Hash Algorithm - SHA-1, SHA2-224, SHA2-256, SHA2-384, SHA2-512	FIPS 186-4
DSA SigVer (FIPS186-4)	A5497	L - 1024, 2048, 3072 N - 160, 224, 256 Hash Algorithm - SHA-1, SHA2-224, SHA2-256, SHA2-384, SHA2-512	FIPS 186-4
ECDSA KeyVer (FIPS186-4)	A5497	Curve - P-192	FIPS 186-4
ECDSA SigVer (FIPS186-4)	A5497	Component - No Curve - P-192 Hash Algorithm - SHA-1, SHA2-224, SHA2-256, SHA2-384, SHA2-512, SHA3-224, SHA3-256, SHA3-384, SHA3-512	FIPS 186-4
RSA SigVer (FIPS186-4)	A5497	Signature Type - PKCS 1.5, PKCSPSS Modulo - 1024, 2048, 3072, 4096	FIPS 186-4

Table 7: Approved Algorithms - Legacy

CVL

Algorithm	CAVP Cert	Properties	Reference
KAS-ECC CDH-Component SP800-56Ar3 (CVL)	A5497	Function - Full Public Key Validation, Key Pair Generation Curve - P-224, P-256, P-384, P-521	SP 800-56A Rev. 3
RSA Decryption Primitive Sp800-56Br2 (CVL)	A5497	Modulo - 2048, 3072, 4096	SP 800-56B Rev. 2
RSA Signature Primitive (CVL)	A5497	Modulo - 2048, 3072, 4096	FIPS 186-4

Table 8: Approved Algorithms - CVL

Before approved signature verification is performed, the module performs a partial public key validation as required by

- FIPS 186-4 Section 3.3 (DSA) and
- FIPS 186-5 Section 3.3 (ECDSA, EdDSA, and RSA)

in accordance with

- SP 800-56Ar3 Section 5.6.2.3.1 and FIPS 186-4 Sections A.1.1.3 / A.2.2 (DSA),
- SP 800-56Ar3 Section 5.6.2.3.4 and SP 800-186 Section D.1.1.1 (ECDSA),
- SP 800-186 Section D.1.3.1 (EdDSA), and
- SP 800-89 Section 5.3.3 (RSA).

The module further performs the following explicit private key validations before approved signature generation:

- SP 800-56Ar3 Section 5.6.2.1.2 (ECDSA).

Assurance of the domain parameter validity for ECDSA and EdDSA is ensured by only supporting the approved curves specified in the CAVP certificate referenced in the above table (also see Section 2.6 Security Function Implementations). When using the FIPS 186-type FFC domain parameters for DSA, the module performs an explicit domain parameter validation for the unverifiable generation method of *g* as specified in SP 800-89 Section 4.1 and FIPS 186-4 Section A.1.1.3 and Section A.2.2.

Vendor-Affirmed Algorithms:

Name	Properties	Implementation	Reference
AES CKG	Key type: Symmetric	N/A	SP 800-133r2 Sections 4 / 6.1 (no post-processing or value V are used)

Name	Properties	Implementation	Reference
Signature CKG	Key type: Asymmetric	N/A	SP 800-133r2 Sections 4 / 5.1 (no post-processing or value V are used)
Key establishment CKG	Key type: Asymmetric	N/A	SP 800-133r2 Sections 4 / 5.2 (no post-processing or value V are used)
DSA signature verification with SHA-3	Key size: L: 1024/N: 160, L: 2048/N: 224, L: 2048/N: 256, L: 3072/N: 256 Hash functions: SHA3-224, SHA3-256, SHA3-384, SHA3-512	SAP CommonCryptoLib Crypto Kernel	FIPS 186-4 Section 4, IG C.C

Table 9: Vendor-Affirmed Algorithms

For the approved key generation algorithms used together with the CKG claimed according to IG D.H, please see the approved algorithms table above and Section 2.6 Security Function Implementations.

Non-Approved, Allowed Algorithms:

N/A for this module.

Non-Approved, Allowed Algorithms with No Security Claimed:

N/A for this module.

Non-Approved, Not Allowed Algorithms:

Name	Use and Function
MD2, MD4, MD5	Hash generation
RIPEMD-128, RIPEMD-160	Hash generation
CRC32	Checksum generation
IDEA, RC2, RC5-32, ARIA128, ARIA192, ARIA256, SEED	Block cipher encryption / decryption, key generation
DES (non-compliant)	Block cipher encryption / decryption, key generation
2-key / 3-key TDES (non-compliant)	Block cipher encryption / decryption and key generation

Name	Use and Function
AES with the mode of operations * ciphertext stealing (CTS) ECB or * OFB with a non-standard number of feedback bits. (non-compliant)	Block cipher encryption / decryption
AES-GCM with * user-provided IV for encryption (outside of the use within TLS 1.2 / 1.3) * decryption without prior tag check, or * tags with a length of < 96 bits. (non-compliant)	Authenticated encryption / decryption
RC4	Stream cipher encryption / decryption, key generation
HMAC generation with * key length less than 112 bits, * IPAD and/or OPAD configured to values not specified in FIPS 198-1, * non-approved hash functions, or * SHAKE128 or SHAKE256. (non-compliant)	HMAC generation
DSA signature generation using key sizes of L: 2048 / N: 224, L: 2048 / N: 256, and L: 3072 / N: 256 and hash functions SHA2-224, SHA2-256, SHA2-384, SHA2-512, SHA3-224, SHA3-256, SHA3-384, and SHA3-512. (non-compliant)	Signature generation
DSA signature verification with * groups not approved for signature verification, * hash functions not approved for signature generation respectively verification, or * pre-hashed messages. (non-compliant)	Signature verification
DSA key pair and domain parameter generation * with groups not approved for KAS-FFC or * for use outside of KAS-FFC. (non-compliant)	Key pair and domain parameter generation
RSA signature generation / verification with * modulus length not approved for signature generation respectively verification, * user-provided salt (only PSS generation), * padding not approved for signature generation,	Signature generation, signature verification

Name	Use and Function
* hash functions not approved for signature generation respectively verification, or * pre-hashed messages (only verification). (non-compliant)	
RSA key pair generation with modulus length not approved for key generation. (non-compliant)	Key pair generation
ECDSA signature generation / verification with * curves not approved for signature generation respectively verification, * hash functions not approved for signature generation respectively verification, * using SHAKE-128 or SHAKE-256, or * pre-hashed messages (only verification). (non-compliant)	Signature generation
ECDSA key pair generation with curves not approved for key generation. (non-compliant)	Key pair generation
KAS-ECC-SSC with curves not approved for KAS-ECC. (non-compliant)	Key agreement, key pair generation
KAS-FFC-SSC with groups not approved for KAS-FFC. (non-compliant)	Key agreement, key pair generation
KTS-IFC (OAEP) with * modulus length not approved for KTS-IFC, * hash functions not approved for use with OAEP, or * user-provided seed (only encapsulation). (non-compliant)	Key transport, key pair generation
RSA key transport with * no padding (only encapsulation) or * padding not approved for use with RSA (i.e., non-OAEP padding). (non-compliant)	Key transport
ElGamal	Key encapsulation, key generation
Counter DRBG using AES-128, AES-192, or AES-256 with derivation function when seeded entirely by the calling application. (non-compliant)	Random number generation

Table 10: Non-Approved, Not Allowed Algorithms

Note that “non-approved” in the above non-approved, not allowed algorithms table refers to algorithms or parameter sets (e.g., modulus sizes or curves) not listed as approved in either the approved algorithms table or the vendor-affirmed algorithms table. Please further note the algorithm specific information provided in Section 2.7 Algorithm Specific Information.

Note that some of the functions listed in the above table are non-compliant implementations of what appear to be approved algorithms. For a description of the security strength as a function of the key length respectively the algorithm parameters of the listed non-approved security functions, please see SP 800-57 Part 1 Revision 5 Section 5.6.1.

2.6 Security Function Implementations

Name	Type	Description	Properties	Algorithms
HMAC generation	MAC		Truncation: not supported Key length: >= 112 bits	HMAC-SHA-1: (A5497) HMAC-SHA2-224: (A5497) HMAC-SHA2-256: (A5497) HMAC-SHA2-384: (A5497) HMAC-SHA2-512: (A5497) HMAC-SHA3-224: (A5497) HMAC-SHA3-256: (A5497) HMAC-SHA3-384: (A5497) HMAC-SHA3-512: (A5497) SHA-1: (A5497) SHA2-224: (A5497) SHA2-256: (A5497) SHA2-384: (A5497) SHA2-512: (A5497) SHA3-224: (A5497) SHA3-256: (A5497)

Name	Type	Description	Properties	Algorithms
				SHA3-384: (A5497) SHA3-512: (A5497)
Hash generation	SHA			SHA-1: (A5497) SHA2-224: (A5497) SHA2-256: (A5497) SHA2-384: (A5497) SHA2-512: (A5497) SHA3-224: (A5497) SHA3-256: (A5497) SHA3-384: (A5497) SHA3-512: (A5497)
XOF generation	XOF			SHAKE-128: (A5497) SHAKE-256: (A5497)
Random number generation	DRBG ENT-Cond ENT-ESV			Counter DRBG: (A5497) AES-CTR: (A5497) Key length: 256 bits SHA2-512: (A5497)
AES encryption / decryption	BC-UnAuth			AES-CBC: (A5497) AES-CBC-CS3: (A5497) AES-CTR: (A5497) AES-ECB: (A5497) AES-OFB: (A5497)
AES GCM encryption / decryption (random IV)	BC-Auth		Tag length: >= 96 bits IV length (for encryption): >= 96 bits	AES-GCM: (A5497)
AES GCM encryption / decryption (TLS 1.2)	BC-Auth		Standards: RFC 5246 and RFC 5288	AES-GCM: (A5497)
AES GCM encryption / decryption (TLS 1.3)	BC-Auth		Standard: RFC 8446	AES-GCM: (A5497)

Name	Type	Description	Properties	Algorithms
AES CFB encryption / decryption	BC-UnAuth		s: 8, 16, ..., 128 (only s = 8 and 128 are CAVP-tested)	AES-CFB8: (A5497) AES-CFB128: (A5497)
Ed25519 signature generation	DigSig-SigGen			EDDSA SigGen: (A5497) SHA2-512: (A5497)
Ed25519 signature verification	DigSig-SigVer			EDDSA SigVer: (A5497) SHA2-512: (A5497)
Ed448 signature generation	DigSig-SigGen			EDDSA SigGen: (A5497) SHAKE-256: (A5497)
Ed448 signature verification	DigSig-SigVer			EDDSA SigVer: (A5497) SHAKE-256: (A5497)
RSA PKCS1-v1.5 signature generation	DigSig-SigGen		Modulus length: >= 2048 bits (only modulus length of 2048, 3072, and 4096 are CAVP-tested)	RSA SigGen (FIPS186-5): (A5497) SHA2-224: (A5497) SHA2-256: (A5497) SHA2-384: (A5497) SHA2-512: (A5497) SHA3-224: (A5497) SHA3-256: (A5497) SHA3-384: (A5497) SHA3-512: (A5497)
RSA PKCS1-v1.5 signature verification	DigSig-SigVer		Modulus length: 1024 bits and >= 2048 bits (only modulus length of 1024, 2048, 3072, and 4096 are CAVP-tested)	RSA SigVer (FIPS186-4): (A5497) RSA SigVer (FIPS186-5): (A5497) SHA-1: (A5497) SHA2-224: (A5497) SHA2-256: (A5497) SHA2-384: (A5497) SHA2-512: (A5497)

Name	Type	Description	Properties	Algorithms
				SHA3-224: (A5497) SHA3-256: (A5497) SHA3-384: (A5497) SHA3-512: (A5497)
RSA PSS signature generation	DigSig-SigGen		Modulus length: >= 2048 bits (only modulus length of 2048, 3072, and 4096 are CAVP-tested)	RSA SigGen (FIPS186-5): (A5497) SHA2-224: (A5497) SHA2-256: (A5497) SHA2-384: (A5497) SHA2-512: (A5497) SHA3-224: (A5497) SHA3-256: (A5497) SHA3-384: (A5497) SHA3-512: (A5497) SHAKE-128: (A5497) SHAKE-256: (A5497)
RSA PSS signature verification	DigSig-SigVer		Modulus length: 1024 bits and >= 2048 bits (only modulus length of 1024, 2048, 3072, and 4096 are CAVP-tested)	RSA SigVer (FIPS186-4): (A5497) RSA SigVer (FIPS186-5): (A5497) SHA-1: (A5497) SHA2-224: (A5497) SHA2-256: (A5497) SHA2-384: (A5497) SHA2-512: (A5497) SHA3-224: (A5497) SHA3-256: (A5497) SHA3-384: (A5497) SHA3-512: (A5497) SHAKE-128: (A5497) SHAKE-256: (A5497)
RSA signature generation with pre-computed hash (CVL)	DigSig-SigGen		Modulus length: >= 2048 bits (only modulus length of 2048, 3072,	RSA Signature Primitive: (A5497)

Name	Type	Description	Properties	Algorithms
			and 4096 are CAVP-tested)	
ECDSA signature generation	DigSig-SigGen		Curves: P-224, P-256, P-384, P-521	ECDSA SigGen (FIPS186-5): (A5497) SHA2-224: (A5497) SHA2-256: (A5497) SHA2-384: (A5497) SHA2-512: (A5497) SHA3-224: (A5497) SHA3-256: (A5497) SHA3-384: (A5497) SHA3-512: (A5497)
ECDSA signature verification	DigSig-SigVer		Curves: P-192, P-224, P-256, P-384, P-521	ECDSA SigVer (FIPS186-4): (A5497) ECDSA SigVer (FIPS186-5): (A5497) SHA-1: (A5497) SHA2-224: (A5497) SHA2-256: (A5497) SHA2-384: (A5497) SHA2-512: (A5497) SHA3-224: (A5497) SHA3-256: (A5497) SHA3-384: (A5497) SHA3-512: (A5497)
ECDSA signature generation with pre-computed hash (CVL)	DigSig-SigGen		Curves: P-224, P-256, P-384, P-521	ECDSA SigGen (FIPS186-5): (A5497)
DSA signature verification	DigSig-SigVer		Groups: L: 1024/N: 160, L: 2048/N: 224, L: 2048/N: 256, L: 3072/N: 256	DSA SigVer (FIPS186-4): (A5497) SHA-1: (A5497) SHA2-224: (A5497) SHA2-256: (A5497) SHA2-384: (A5497)

Name	Type	Description	Properties	Algorithms
				SHA2-512: (A5497) SHA3-224: (A5497) SHA3-256: (A5497) SHA3-384: (A5497) SHA3-512: (A5497) DSA signature verification with SHA-3: () DSA PQGVer (FIPS186-4): (A5497)
EdDSA public key validation	AsymKeyPair-PubKeyVal		Curves: Edwards25519, Edwards448	EDDSA KeyVer: (A5497)
ECDSA public key validation	AsymKeyPair-PubKeyVal		Curves: P-192, P-224, P-256, P-384, P-521	ECDSA KeyVer (FIPS186-4): (A5497) ECDSA KeyVer (FIPS186-5): (A5497)
DSA domain parameter validation	AsymKeyPair-DomPar		Groups: L: 1024/N: 160, L: 2048/N: 224 (FB), L: 2048/N: 256 (FC), L: 3072/N: 256	DSA PQGVer (FIPS186-4): (A5497) SHA-1: (A5497) SHA2-224: (A5497) SHA2-256: (A5497) SHA2-384: (A5497) SHA2-512: (A5497)
EdDSA key pair generation	AsymKeyPair-KeyGen CKG		Curves: Edwards25519, Edwards448	EDDSA KeyGen: (A5497) Signature CKG: ()
ECDSA key pair generation	AsymKeyPair-KeyGen CKG		Curves: P-224, P-256, P-384, P-521	ECDSA KeyGen (FIPS186-5): (A5497) Signature CKG: ()
RSA key pair generation	AsymKeyPair-KeyGen CKG		Modulus length: 2048 to 8192 bits (only modulus length of 2048, 3072, 4096, and 8192 are CAVP-tested)	RSA KeyGen (FIPS186-5): (A5497) Signature CKG: ()

Name	Type	Description	Properties	Algorithms
AES key generation	CKG		Key length: 128, 192, 256 bits	AES CKG: ()
KTS-IFC	KTS-Encap		Standard: SP 800-56Brev2 IG D.G: Approved RSA-based key transport scheme Key confirmation: No Caveat: Key establishment methodology provides between 112 and 256 bits of security strength Modulus length: 2048 bits (only modulus length of 2048, 3072, and 4096 are CAVP-tested)	KTS-IFC: (A5497) Key establishment CKG: () RSA KeyGen (FIPS186-5): (A5497) RSA Decryption Primitive Sp800-56Br2: (A5497)
KAS-SSC FFC	AsymKeyPair-DomPar CKG KAS-KeyGen KAS-SSC		IG: IG D.F Scenario 2, path (1) Caveat: Key establishment methodology provides between 112 and 192 bits of security strength Groups: FB, FC, MODP 2048, MODP 3072, MODP 4096, MODP 6144, MODP 8192, ffdhe2048, ffdhe3072, ffdhe4096, ffdhe6144, ffdhe8192	KAS-FFC-SSC Sp800-56Ar3: (A5497) DSA KeyGen (FIPS186-4): (A5497) DSA PQGGen (FIPS186-4): (A5497) SHA2-224: (A5497) SHA2-256: (A5497) SHA2-384: (A5497) SHA2-512: (A5497) DSA PQGVer (FIPS186-4): (A5497) Key establishment CKG: () Safe Primes Key Generation: (A5497)

Name	Type	Description	Properties	Algorithms
				Safe Primes Key Verification: (A5497)
KAS-SSC ECC	CKG KAS-KeyGen KAS-SSC		IG: IG D.F Scenario 2, path (1) Caveat: Key establishment methodology provides between 112 and 256 bits of security strength	KAS-ECC-SSC Sp800-56Ar3: (A5497) KAS-ECC CDH-Component SP800-56Ar3: (A5497) EDDSA KeyGen: (A5497) Key establishment CKG: () ECDSA KeyVer (FIPS186-5): (A5497)

Table 11: Security Function Implementations

Unless identified otherwise in the above table, all tested capabilities (e.g., key sizes, curves, modes) of the listed algorithms are used by the security function implementations. For details, please refer to Section 2.5 Algorithms.

Note that the “random number generation” SFI also makes use of the module’s validated entropy source (cf. 2.8 RBG and Entropy).

2.7 Algorithm Specific Information

When using the approved security functions (see Section 2.6 Security Function Implementations), the Crypto Officer **shall** observe the following algorithm-specific requirements:

KAS-SSC FFC / ECC: The module does not establish SSPs using an approved key agreement scheme (KAS). However, it does offer some or all of the underlying KAS cryptographic functionality to be used by an external operator/application as part of an approved KAS.

KTS-IFC: The module does not establish SSPs using an approved key transport scheme (KTS). However, it does offer approved authenticated algorithms that can be used by an external operator/application as part of an approved KTS.

AES-CTR Encryption: Externally loaded counter values **shall** have the properties required by SP 800-38A Section 6.5.

AES-GCM: The module provides APIs to use AES-GCM encryption as specified in SP 800-38D as an approved service according to the following scenarios of IG C.H:

- 1a (GCM cipher suites in TLS 1.2),
- 2 (for encryption: random IV generated by the internal Counter DRBG instance with a length of at least 96 bits), and
- 5 (GCM cipher suites in TLS 1.3).

The use of AES-GCM encryption with a user-provided IV is not approved. The APIs provided for scenarios 1a and 5 **shall** only be used in context of the TLS 1.2 and TLS 1.3 protocol, respectively.

The module itself does not implement the full TLS protocol but just the AES-GCM mode of operation and the IV / nonce management logic according to RFC 5246 and RFC 5288 for TLS 1.2, and RFC 8446 for TLS 1.3. For this reason, the following statement of IG D.C case 3 is applicable: no parts of these protocols, other than the approved AES-GCM algorithm, have been tested by the CAVP and CMVP.

The module ensures that an error is provided to the linked application before the IV exhausts the maximum number of values for a given key in all three scenarios. A new key **shall** be established in this case. The module itself is not capable of restoring the IV state after a power loss, therefore, in case the module's power is lost and then restored, a new (session) key for use with the AES-GCM encryption **shall** be established by the Crypto Officer.

Component Algorithms (CVLs): For the following algorithms (cf. Section 2.5 Algorithms) usage restrictions apply:

- ECDSA signature generation without the computation of a hash: This component **shall** only be used within the context of a FIPS 186-5 signature generation.
- RSA signature generation without the computation of a hash: This component **shall** only be used within the context of a FIPS 186-5 signature generation.
- RSA decryption primitive: This component **shall** only be used within the context of a SP 800-56Br2 KTS.
- KAS-ECC CDH-Component: This component **shall** only be used within the context of a SP 800-56Ar3 KAS.

Legacy Algorithms: The following algorithms are only approved for legacy use (cf. Section 2.5 Algorithms):

- FIPS 186-4 DSA signature verification and domain parameter verification,
- FIPS 186-4 ECDSA signature verification with curve P-192 or using SHA-1 and ECDSA public key validation using curve P-192,
- FIPS 186-4 RSA signature verification with a modulus size of 1024 bits or using SHA-1.

These legacy algorithms **shall** only be used on data that was generated prior to the Legacy Date specified in FIPS 140-3 IG C.M.

SHA-1: SHA-1 **shall** only be used for digital signature generation where specifically allowed by NIST protocol-specific guidance. For all other applications, SHA-1 **shall not** be used for digital signature generation. When used for digital signature verification, SHA-1 **shall** only be used in legacy applications. For non-digital-signature applications, SHA-1 **shall** only be used in applications that do not require collision resistance.

2.8 RBG and Entropy

Cert Number	Vendor Name
E172	SAP

Table 12: Entropy Certificates

Name	Type	Operational Environment	Sample Size	Entropy per Sample	Conditioning Component
SAP CommonCryptoLib Entropy Collector 1.0.0	Non-Physical	All operating environments listed in Section 2.2 Tested and Vendor Affirmed Module Version and Identification.	512 bits	Full entropy	SHA2-512 (see Section 2.5 Algorithms for the CAVP Cert.)

Table 13: Entropy Sources

The entropy source is within the cryptographical boundary of the module. The module's entropy source provides 512 bits of min-entropy per conditioned 512-bit output.

The module implements an internal Counter DRBG instance using AES-256 with derivation function, but without support for reseeding and predication resistance (i.e., the Counter DRBG is only seeded once during module initialization). This DBRG instance is managed internally by the module and used for all random number generation in the approved security functions, for SSP generation, and SSP establishment. It is seeded entirely by the validated entropy source during module initialization and thus provides a security strength of 256 bits. After the allowed maximum number of random bits was requested from the Counter DRBG, further output is blocked, and the module **shall** be reinitialized to request additional outputs.

The Counter DRBG's additional input and personalization string used for its initialization are derived from additional outputs of the entropy source and, if available, data obtained from the operating system ("dev/urandom" on operating systems that provide this file) as well as data provided by the linked application during module initialization. Note that this additional data is not considered in the security strength estimate of the approved Counter DRBG.

In compliance with IG 2.4.A, the internal Counter DRBG is used by approved and non-approved services for the following purposes by the module:

In Approved Services:
- Blinding of RSA decryption and signature generation as well as ECDSA signature generation operations (see Section 12 Mitigation of Other Attacks).
- ECDSA signature generation (per-message secret number).
- Generation of random bit strings with variable length (as requested by the operator).
- IV generation for the CBC, CBC-CS3, CFB, OFB, CTR, and GCM modes of operation.
- Key and domain parameter generation for the approved symmetric and asymmetric cryptographic functions (see Section 2.5 Algorithms and Section 2.9 Key Generation).
- Pair-Wise Consistency Tests (PCTs) for approved ECDSA, KAS-SSC ECC, KTS-IFC, and RSA.
- Primality testing for RSA assurance checks (see Section 2.5 Algorithms and Section 2.10 Key Establishment).
- RSA-OAEP encryption (seed).
- RSA-PSS signature generation (salt).

In Non-Approved Services:
- DSA signature generation (per-message secret number).
- Generation of random numbers / primes and primality testing within the implemented long number arithmetic functions.
- Key generation for the non-approved symmetric and asymmetric cryptographic functions (see Section 2.5 Algorithms and Section 2.9 Key Generation).
- Pair-Wise Consistency Tests (PCTs) for DSA and non-approved ECDSA, KAS-SSC ECC, KTS-IFC, and RSA key pairs.
- RSA PKCS1-v1.5 encryption (PS).

The state information of the internal Counter DRBG instance as well as its entropy inputs are considered as CSPs (also see Section 9.4 SSPs). The additional data used for deriving personalization string and additional input are not considered as SSPs as they cannot degrade the module's security.

The operator of the module is also able to instantiate their own non-approved Counter DRBG instance using AES-128, AES-192, or AES-256. This instance must be seeded entirely by the operator. It uses the derivation function and supports reseeding but does not support prediction resistance.

2.9 Key Generation

The module's key generation methods as well as the related vendor-affirmed CKG entries per IG D.H are specified in Section 2.5 Algorithms and Section 2.6 Security Function Implementations. The approved services for key generation cover

- symmetric key generation for AES,
- asymmetric key pair generation for ECDSA, EdDSA, KAS-SSC ECC, KTS-IFC, and RSA as well as
- asymmetric key pair and domain parameter generation for KAS-SSC FFC.

For the pair-wise consistency tests performed after key pair generation please see Section 11.2 Administrator Guidance.

All random numbers used during the key generation processes are taken from the approved internal Counter DRBG instance (see Section 2.8 RBG and Entropy). As the claimed security strength of this Counter DRBG is 256 bits and at the time of publication of this SP the claimed security strength of any approved algorithm may not be greater than 256 bits, no entropy caveat is applicable for key generation.

All intermediate key generation values processed by the module's approved services are considered as CSPs (also see Section 9.4 SSPs). They are not output.

2.10 Key Establishment

The key establishment schemes in terms of the implemented Key Agreement Schemes Shared Secret Computation (KAS-SSC) and Key Transport Schemes (KTS) as specified in SP 800-56Ar3 and SP 800-56Br2 implemented by the module are listed in Section 2.5 Algorithms and Section 2.6 Security Function Implementations. The module only supports the shared secret computation. It does not implement key derivation or key confirmation. It can act both in the initiator and responder roles.

As required by IG D.F, for the implemented approved schemes, the module obtains all assurances required by the respective standards for which the module has the necessary inputs either using explicit assurance checks or by generating all values as specified in the respective algorithm standard. The coverage of the required assurances, other than the pair-wise consistency tests that are always performed when a key pair (i.e., both a private and a public key) is generated or imported (see Section 11.2 Administrator Guidance for details), are explained in Table 14. When not stated otherwise, the assurances are obtained directly by the dedicated KAS-SSC and KTS APIs. Some assurances must be manually obtained by the operator of the module as the module does not have the necessary inputs to perform the validations by itself. For the implemented KAS-SSC, the owner's private key and received public key are validated at the latest before generating the shared secret. It is also ensured that the owner's static or ephemeral private / public key is validated before it is first exported.

<table>
<tr><th>Scheme</th><th>Owner’s private key</th><th>Owner’s public key</th><th>Received public key</th><th>Domain parameters</th></tr>
<tr><td colspan="5">SP 800-56Ar3</td></tr>
<tr><td>KAS-FFC-SSC
(C(2e, 0s, FFC-DH))</td><td rowspan="2">SP 800-56Ar3,
5.6.2.1.2</td><td rowspan="2">SP 800-56Ar3,
5.6.2.3.1</td><td>SP 800-56Ar3,
5.6.2.2.2</td><td rowspan="2">N/A for named approved groups.

For FIPS 186-type FFC domain parameters, explicit domain parameter validation (for the unverifiable generation method of g) as specified in SP 800-89 Section 4.1 respectively FIPS 186-4 Section A.1.1.3 / A.2.2 is performed if the used seed is provided.</td></tr>
<tr><td>KAS-FFC-SSC
(C(0e, 2s, FFC DH))</td><td>SP 800-56Ar3,
5.6.2.2.1</td></tr>
<tr><td>KAS-ECC-SSC
(C(2e, 0s, ECC CDH))</td><td rowspan="3">SP 800-56Ar3,
5.6.2.1.2</td><td>SP 800-56Ar3,
5.6.2.3.3</td><td rowspan="3">SP 800-56Ar3,
5.6.2.3.3</td><td rowspan="3">N/A for approved named curves.</td></tr>
<tr><td>KAS-ECC-SSC
(C(0e, 2s, ECC CDH))</td></tr>
<tr><td>KAS-ECC
(ECC CDH primitive / component)</td><td>Shall be obtained manually (see below) as the owner’s public key is not input into this API.</td></tr>
<tr><td colspan="5">SP 800-56Br2</td></tr>
<tr><td>KTS-IFC
(KTS-OAEP-basic)</td><td colspan="2" rowspan="2">Shall be obtained manually (see below) before calling the KTS APIs, which only take the owner’s private key and received ciphertext as inputs.</td><td>SP 800-56Br2,
6.4.2.2 / SP 800-89, 5.3.3</td><td rowspan="2">N/A for RSA.</td></tr>
<tr><td>RSA Decryption Primitive</td><td>N/A as this Scheme does not allow encrypting.</td></tr>
</table>

Table 14: Obtained Assurances for the Implemented Approved KAS-SSC and KTS

The module further provides the following standalone functions, which can be used independently of the dedicated KAS-SSC and KTS APIs addressed in Table 14, to manually obtain assurances:

- `sec_crypto_keypair_sp800_56b_validate_RSA`: rsakpv1-crt key pair validation as specified in SP 800-56Br2 Section 6.4.1.2.3 for RSA key pairs.
- `sec_crypto_ecc_sp800_56a_checkKeyFull`: Full public key validation as specified in SP 800-56Ar3 Section 5.6.2.3.3 respectively SP 800-186 Section D.1.1.2 for ECDSA / KAS-SSC ECC key pairs respectively SP 800-186 Section D.1.3.2 for EdDSA key pairs.
- `sec_crypto_ffc_parameterValidate`: Validation of the probably primes *p* and *q* for KAS-SSC FFC (DSA) as specified in FIPS 186-4 Section A.1.1.3.

As the module only implements the shared secret computation part of the KAS (i.e., no key derivation or use of a nonce), the difference between using the KAS-SSC with static and ephemeral key pairs lies in the assurances obtained and the use of additional module-internal checks that prevent using the freshly generated ephemeral keys for multiple key establishments.

Key generation is addressed in Section 2.9 Key Generation. SSP entry in form of plaintext SSP input from and output to the application linked to the module are addressed in Section 9.2 SSP Input-Output Methods. Key derivation is not supported by the module, and it does not contain any pre-loaded SSPs.

2.11 Industry Protocols

The module itself does not implement any industry protocols. However, note the information provided in Section 2.7 Algorithm Specific Information for the use of AES-GCM encryption in context of the TLS 1.2 and 1.3 protocols.

3 Cryptographic Module Interfaces

3.1 Ports and Interfaces

Physical Port	Logical Interface(s)	Data That Passes
N/A	Data Input	API input parameters
N/A	Data Output	API output parameters
N/A	Control Input	API calls
N/A	Status Output	API output parameters for status and API return values

Table 15: Ports and Interfaces

As the module is a software library, its logical interfaces are realized in terms of a set of APIs. The above table maps the FIPS 140-3 logical interfaces to the distinct parts of these APIs.

All functionality of the module is made available to the calling application (i.e., the operator of the module) in terms of exported functions (APIs). Some of these functions are also used internally, e.g., the self-test service makes use of some of those functions when performing cryptographic algorithm self-tests. For a full reference of all exported APIs, please see the guidance documents referenced in Section 11.2 Administrator Guidance.

Because the module is a software library, it does not have any physical ports or manual controls of its own and does not support any external input or output devices. It also does not have a maintenance access interface.

3.2 Trusted Channel Specification

The module does not implement a trusted channel.

3.3 Control Interface Not Inhibited

Not applicable as the module does not implement a control output interface.

3.4 Additional Information

The module uses technical means to inhibit the data output interface during pre-operational self-tests, during zeroization of non-temporary SSPs that are under control of the module, and when in an error state.

4 Roles, Services, and Authentication

4.1 Authentication Methods

The module does not implement operator authentication.

4.2 Roles

Name	Type	Operator Type	Authentication Methods
Crypto Officer	Role	CO	None

Table 16: Roles

The module only supports the Crypto Officer role. As the module does not implement any operator identification or authentication mechanisms, the Crypto Officer role is implicitly assumed by the operator when calling any of the module's APIs.

Note that the module does not allow the operator to perform maintenance services and thus does not support a maintenance role.

4.3 Approved Services

Name	Description	Indicator	Inputs	Outputs	Security Functions	SSP Access
Initialize and self-test	Configures and initializes the module as well as performs the pre-operational self-tests and the CASTs.	Return value 1	Additional input for DRBG, memory management callback pointers, PAA and test mode configuration, path to the shared library (for non-tested OEs)	List of available API functions if all self-tests passed or test result in form of error code if any test failed	HMAC generation Hash generation XOF generation Random number generation AES encryption / decryption AES GCM encryption / decryption	Crypto Officer - Counter DBRG seed: entropy source output: G - Counter DRBG key: G - Counter DRBG V: G

Name	Description	Indicator	Inputs	Outputs	Security Functions	SSP Access
					(random IV) Ed25519 signature generation Ed25519 signature verification Ed448 signature generation Ed448 signature verification RSA PKCS1-v1.5 signature verification RSA signature generation with pre-computed hash (CVL) ECDSA signature verification ECDSA signature generation with pre-computed hash (CVL) DSA	

Name	Description	Indicator	Inputs	Outputs	Security Functions	SSP Access
					signature verification EdDSA public key validation ECDSA public key validation DSA domain parameter validation KTS-IFC KAS-SSC FFC KAS-SSC ECC	
Finalize and zeroize	Finalizes the module. Internal resources are zeroized / released.	Return value 1	-	-	None	Crypto Officer - Counter DRBG key: Z - Counter DRBG V: Z
Show versioning information	Outputs the module name and version.	None	-	Module name and version	None	Crypto Officer
Show status	Informs about the module status, (e.g., the status of each service call).	None	-	Status information	None	Crypto Officer
Block cipher encryption	Encrypt data using a symmetric block cipher.	Return value 1	Algorithm, mode of operation, plaintext, key, IV (optional), additional authenticated data (if any), authentication	Ciphertext, authentication tag (if any)	Random number generation AES encryption / decryption	Crypto Officer - AES keys: W,E

Name	Description	Indicator	Inputs	Outputs	Security Functions	SSP Access
			tag (if any), padding scheme		AES GCM encryption / decryption (random IV) AES GCM encryption / decryption (TLS 1.2) AES GCM encryption / decryption (TLS 1.3) AES CFB encryption / decryption	
Block cipher decryption	Decrypt data using a symmetric block cipher.	Return value 1	Algorithm, mode of operation, ciphertext, key, IV, additional authenticated data (if any), authentication tag (if any), padding scheme	Plaintext, tag check result	AES encryption / decryption AES GCM encryption / decryption (random IV) AES GCM encryption / decryption (TLS 1.2) AES GCM encryption / decryption (TLS 1.3) AES CFB encryption / decryption	Crypto Officer - AES keys: W,E

Name	Description	Indicator	Inputs	Outputs	Security Functions	SSP Access
Key transport	Asymmetric key material encapsulation and decapsulation.	Return value 1	Scheme, public key / private key (optional), keying material / ciphertext	Ciphertext / keying material	KTS-IFC	Crypto Officer - KTS-IFC private key: G,R,W,E - KTS-IFC public key: G,R,W,E - Received KTS-IFC public key: W,E - Intermediate key generation values: G,Z - Keying material: R,W,E - Counter DRBG key: E - Counter DRBG V: E
Signature generation	Digital signature generation.	Return value 1	Algorithm, domain parameters, private key, hash algorithm, (hash of) message, padding scheme	Signature	Random number generation Ed25519 signature generation Ed448 signature generation RSA PKCS1-v1.5 signature	Crypto Officer - EdDSA private keys: W,E - ECDSA private keys: W,E - RSA private keys: W,E - Counter DRBG key: E

Name	Description	Indicator	Inputs	Outputs	Security Functions	SSP Access
					generation RSA PSS signature generation RSA signature generation with pre-computed hash (CVL) ECDSA signature generation ECDSA signature generation with pre-computed hash (CVL)	- Counter DRBG V: E
Signature verification	Digital signature verification.	Return value 1	Algorithm, domain parameters, public key, hash algorithm, (hash of) message, padding scheme, signature	Verification result	Ed25519 signature verification Ed448 signature verification RSA PKCS1-v1.5 signature verification RSA PSS signature verification ECDSA	Crypto Officer - EdDSA public keys: W,E - ECDSA public keys: W,E - DSA public keys: W,E - RSA public keys: W,E - Counter DRBG key: E - Counter

Name	Description	Indicator	Inputs	Outputs	Security Functions	SSP Access
					signature verification DSA signature verification EdDSA public key validation ECDSA public key validation DSA domain parameter validation	DRBG V: E - Intermediate key generation values: G,Z
Shared secret computation	ECC and FFC key agreement.	Return value 1	Scheme, domain parameters, own private key (optional), received public key	Own public key, shared secret	Random number generation KAS-SSC FFC KAS-SSC ECC	Crypto Officer - KAS-ECC private key: G,R,W,E - KAS-ECC public key: G,R,W,E - Received KAS-ECC public key: W,E - KAS-FFC private key: G,R,W,E - KAS-FFC public key: G,R,W,E - Received KAS-FFC public key:

Name	Description	Indicator	Inputs	Outputs	Security Functions	SSP Access
						W,E - Counter DRBG key: E - Counter DRBG V: E - Intermediate key generation values: G,Z - Shared secret: G,R - KAS-ECC domain parameters: R,W,E - KAS-FFC domain parameters: G,R,W,E
Assurance checks	Perform assurance checks as specified in Section 2.10 Key Establishment.	Return value 1	Domain parameters, public key, private key	Check result	Random number generation EdDSA public key validation ECDSA public key validation DSA domain parameter validation	Crypto Officer - KTS-IFC private key: W,E - KTS-IFC public key: W,E - Received KAS-ECC public key: W,E - Received KAS-FFC public key: W,E

Name	Description	Indicator	Inputs	Outputs	Security Functions	SSP Access
						- Counter DRBG key: E - Counter DRBG V: E - KAS-ECC domain parameters: W,E - KAS-FFC domain parameters: W,E
HMAC generation	HMAC generation.	Return value 1	Hash algorithm, key, IPAD and OPAD values, data	HMAC value	HMAC generation	Crypto Officer - HMAC keys: W,E
Hash generation	Hash generation.	Return value 1	Hash algorithm, data	Hash value	Hash generation	Crypto Officer
XOF generation	XOF generation.	Return value 1	XOF algorithm, data, output length	XOF output	XOF generation	Crypto Officer
Random number generation	Random number generation.	Return value 1	Number of bytes to generate	Random data	Random number generation	Crypto Officer - Counter DRBG key: E - Counter DRBG V: E
Signature key pair generation	Signature key pair generation	Return value 1	Key type, key size, domain parameters, hash algorithm (for DSA and EdDSA)	Generated key pair	Random number generation EdDSA key pair generation ECDSA key pair generation	Crypto Officer - EdDSA private keys: G,R - EdDSA public keys: G,R - ECDSA private keys:

Name	Description	Indicator	Inputs	Outputs	Security Functions	SSP Access
					RSA key pair generation	G,R - ECDSA public keys: G,R - RSA private keys: G,R - RSA public keys: G,R - Intermediate key generation values: G,Z
Symmetric key generation	Symmetric key generation.	Return value 1	Key type, key size	Generated key	Random number generation AES key generation	Crypto Officer - AES keys: G,R - Counter DRBG key: E - Counter DRBG V: E
Export / import cryptographic object	Export / import of cryptographic context objects.	Return value 1	Context object / blob	Blob / context object	None	Crypto Officer - AES keys: R,W - HMAC keys: R,W - EdDSA private keys: R,W - EdDSA public keys: R,W - ECDSA private keys: R,W - ECDSA

Name	Description	Indicator	Inputs	Outputs	Security Functions	SSP Access
						public keys: R,W - DSA public keys: R,W - RSA private keys: R,W - RSA public keys: R,W - KAS-ECC private key: R,W - KAS-ECC public key: R,W - KAS-FFC private key: R,W - KAS-FFC public key: R,W - Received KAS-ECC public key: R,W - Received KAS-FFC public key: R,W - KTS-IFC private key: R,W - KTS-IFC public key:

Name	Description	Indicator	Inputs	Outputs	Security Functions	SSP Access
						R,W - Received KTS-IFC public key: R,W

Table 17: Approved Services

For the table above, the following notation is used to indicate the type of SSP access:

- **G = Generate**: The module generates or derives the SSP.
- **R = Read**: The SSP is read from the module (e.g., the SSP is output).
- **W = Write**: The SSP is updated, imported, or written to the module.
- **E = Execute**: The module uses the SSP in performing a cryptographic operation.
- **Z = Zeroize**: The module zeroizes the SSP.

The module operator can identify whether a security service was performed as an approved or non-approved service using the implemented approved service indicator. The implemented service indicator is compliant with IG 2.4.C example scenario 1. This indicator is provided by every API that can determine from the passed parameters whether the service is approved or not. The expected return value for a successfully performed call of an approved security service is `RC_FIPS_APPROVED` (value “1”). Other successful approved and non-approved operations either return the value “0” or do not have a return code. Failed operations return a value less than “0”.

Some APIs may be executed as either an approved or non-approved security service depending on the provided parameters. For example, for some algorithms there are restrictions concerning the key length or the underlying elliptic curves that are considered approved. The approved state of services further depends on the module’s mode of operation. For more information on this, please refer to Section 2.4 Modes of Operation describing the available modes of operation, and Section 2.5 Algorithms listing implemented approved and non-approved algorithms. Note that when key pairs are imported for digital signature generation / verification, the approved service indicator for both operations is set based on the properties of the private key.

Other than for the self-tests of non-approved algorithms during module initialization or when manually invoked by the operator, the module does not utilize any non-approved algorithms as part of an approved service.

4.4 Non-Approved Services

Name	Description	Algorithms	Role
Block cipher encryption	Encrypt data using a symmetric block cipher.	IDEA, RC2, RC5-32, ARIA128, ARIA192, ARIA256, SEED AES with the mode of operations * ciphertext stealing (CTS) ECB or * OFB with a non-standard number of feedback bits. (non-compliant) AES-GCM with * user-provided IV for encryption (outside of the use within TLS 1.2 / 1.3) * decryption without prior tag check, or * tags with a length of < 96 bits. (non-compliant)	Crypto Officer
Block cipher decryption	Decrypt data using a symmetric block cipher.	IDEA, RC2, RC5-32, ARIA128, ARIA192, ARIA256, SEED AES with the mode of operations * ciphertext stealing (CTS) ECB or * OFB with a non-standard number of feedback bits. (non-compliant)	Crypto Officer
Key transport	Asymmetric key material encapsulation and decapsulation.	ElGamal KTS-IFC (OAEP) with * modulus length not approved for KTS-IFC, * hash functions not approved for use with OAEP, or * user-provided seed (only encapsulation). (non-compliant) RSA key transport with * no padding (only encapsulation) or * padding not approved for use with RSA (i.e., non-OAEP padding). (non-compliant)	Crypto Officer
Signature generation	Digital signature generation.	DSA signature generation using key sizes of L: 2048 / N: 224, L: 2048 / N: 256, and L: 3072 / N: 256 and hash functions SHA2-224, SHA2-256, SHA2-384, SHA2-512, SHA3-224, SHA3-256, SHA3-384, and SHA3-512. (non-compliant)	Crypto Officer

Name	Description	Algorithms	Role
		RSA signature generation / verification with * modulus length not approved for signature generation respectively verification, * user-provided salt (only PSS generation), * padding not approved for signature generation, * hash functions not approved for signature generation respectively verification, or * pre-hashed messages (only verification). (non-compliant) ECDSA signature generation / verification with * curves not approved for signature generation respectively verification, * hash functions not approved for signature generation respectively verification, * using SHAKE-128 or SHAKE-256, or * pre-hashed messages (only verification). (non-compliant)	
Signature verification	Digital signature verification.	DSA signature verification with * groups not approved for signature verification, * hash functions not approved for signature generation respectively verification, or * pre-hashed messages. (non-compliant) RSA signature generation / verification with * modulus length not approved for signature generation respectively verification, * user-provided salt (only PSS generation), * padding not approved for signature generation, * hash functions not approved for signature generation respectively verification, or * pre-hashed messages (only verification). (non-compliant) ECDSA signature generation / verification with * curves not approved for signature generation respectively	Crypto Officer

Name	Description	Algorithms	Role
		verification, * hash functions not approved for signature generation respectively verification, * using SHAKE-128 or SHAKE-256, or * pre-hashed messages (only verification). (non-compliant)	
Shared secret computation	ECC and FFC key agreement.	DSA key pair and domain parameter generation * with groups not approved for KAS-FFC or * for use outside of KAS-FFC. (non-compliant) RSA key pair generation with modulus length not approved for key generation. (non-compliant) ECDSA key pair generation with curves not approved for key generation. (non-compliant) KAS-ECC-SSC with curves not approved for KAS-ECC. (non-compliant) KAS-FFC-SSC with groups not approved for KAS-FFC. (non-compliant)	Crypto Officer
HMAC generation	HMAC generation.	HMAC generation with * key length less than 112 bits, * IPAD and/or OPAD configured to values not specified in FIPS 198-1, * non-approved hash functions, or * SHAKE128 or SHAKE256. (non-compliant)	Crypto Officer
Hash generation	Hash generation.	MD2, MD4, MD5 RIPEMD-128, RIPEMD-160 CRC32	Crypto Officer
Random number generation	Random number generation.	Counter DRBG using AES-128, AES-192, or AES-256 with derivation function when seeded entirely by the calling application. (non-compliant)	Crypto Officer

Name	Description	Algorithms	Role
Signature key pair generation	Signature key pair generation.	DSA key pair and domain parameter generation * with groups not approved for KAS-FFC or * for use outside of KAS-FFC. (non-compliant) RSA key pair generation with modulus length not approved for key generation. (non-compliant) ECDSA key pair generation with curves not approved for key generation. (non-compliant)	Crypto Officer
Symmetric key generation	Symmetric key generation.	IDEA, RC2, RC5-32, ARIA128, ARIA192, ARIA256, SEED RC4 ElGamal 2-key / 3-key TDES (non-compliant) DES (non-compliant)	Crypto Officer
Export / import cryptographic object	Export / import of cryptographic context objects of non-approved algorithms.	MD2, MD4, MD5 RIPEMD-128, RIPEMD-160 CRC32 IDEA, RC2, RC5-32, ARIA128, ARIA192, ARIA256, SEED DES (non-compliant) 2-key / 3-key TDES (non-compliant) AES with the mode of operations * ciphertext stealing (CTS) ECB or * OFB with a non-standard number of feedback bits. (non-compliant) AES-GCM with * user-provided IV for encryption (outside of the use within TLS 1.2 / 1.3) * decryption without prior tag check, or * tags with a length of < 96 bits.	Crypto Officer

Name	Description	Algorithms	Role
		(non-compliant) RC4 HMAC generation with * key length less than 112 bits, * IPAD and/or OPAD configured to values not specified in FIPS 198-1, * non-approved hash functions, or * SHAKE128 or SHAKE256. (non-compliant) KAS-ECC-SSC with curves not approved for KAS-ECC. (non-compliant) KAS-FFC-SSC with groups not approved for KAS-FFC. (non-compliant) Counter DRBG using AES-128, AES-192, or AES-256 with derivation function when seeded entirely by the calling application. (non-compliant)	
Stream cipher encryption / decryption	Encrypt / decrypt data using a symmetric stream cipher.	RC4	Crypto Officer

Table 18: Non-Approved Services

Note that the approved internal Counter DRBG instance is used by some of the non-approved services per the allowance in IG 2.4.A. For a detailed description of the purposes for which this DRBG instance is used, please refer to Section 2.8 RBG and Entropy.

4.5 External Software/Firmware Loaded

The module does not support software/firmware loading.

4.6 Bypass Actions and Status

The module does not implement a bypass capability.

4.7 Cryptographic Output Actions and Status

The module does not implement a self-initiated output capability.

5 Software/Firmware Security

5.1 Integrity Techniques

The module uses an HMAC-SHA2-256 (see Section 2.5 Algorithms for the CAVP certificate number) as the approved integrity technique for the software/firmware integrity test. The HMAC is computed over the entire shared library file during module initialization. The reference value for the integrity test is stored in a separate file, with the exact delivery format depending on the used platform (see Section 11.1 Installation, Initialization, and Startup Procedures for more details):

- On Windows platforms, the module comprises the dynamic link library file slcryptokernel.dll accompanied by the file `slcryptokernel.dll.sha256` containing the reference value for the software/firmware integrity test.

- On Linux and UNIX based platforms, the module comprises the shared library file `libslcryptokernel.<extension>` accompanied by the file `libslcryptokernel.<extension>.sha256` containing the reference value for the software/firmware integrity test. Where “<extension>” stands for the OS-specific extension for shared libraries. These extensions are “dylib” for macOS and “so” for the other Linux/Unix operating systems.

5.2 Initiate on Demand

The integrity test can be executed on demand by re-initializing the module. This process is described in more detail in Section 10.5 Operator Initiation of Self-Tests.

6 Operational Environment

6.1 Operational Environment Type and Requirements

Type of Operational Environment: Modifiable

How Requirements are Satisfied:

This module is expected to be run in operational environments where each user application runs in a virtually-separated, independent process with its own address space. By default, all tested operating systems (Windows as well as the Unix derivates) provide such a separation so that no other, unauthorized process can access or modify SSPs while the cryptographic module is in use. The virtual memory provided by the operating systems also ensures that every instance of the module has exclusive control over its own SSPs.

The module does not spawn any processes or threads. It uses only memory within the virtual address space of the process and does not communicate with any other process in any way (e.g., using inter-process communication such as pipes). The application linked to the module and running in the same virtual memory area is expected to only interact with the module through the defined interfaces.

6.2 Configuration Settings and Restrictions

The operator of the module **shall not** configure the operating systems in a way that disables the process separation mechanisms referenced in Section 6.1 Operational Environment Type and Requirements.

7 Physical Security

The module comprises only software. It runs on operational environments with a multiple-chip standalone embodiment. It has no physical protection mechanisms. Therefore, the physical security requirements are not applicable.

8 Non-Invasive Security

The module does not implement any non-invasive security measures that are referenced in SP 800-140F.

9 Sensitive Security Parameters Management

9.1 Storage Areas

Storage Area Name	Description	Persistence Type
RAM	Volatile system memory shared with the linked application.	Dynamic

Table 19: Storage Areas

The state of the internal Counter DRBG instance is stored at most until the end of the module's runtime in volatile memory allocated by the module itself. All other SSPs used by the module are stored in the volatile memory shared with the linked application. SSPs passed to the module via memory pointers by the linked application are only accessed and used within a single API call. The module does not keep references to these SSPs after the API call is done.

To allow chaining of related APIs (e.g., for encrypting multiple blocks of data) the state of certain security functions is stored in context objects that include any required SSPs (either by reference or directly as byte arrays). These objects are passed back and forth between the module and the linked application for each API call. Where necessary to allow for procedural zeroization of such objects, the module provides information about the location and size of allocated memory to the linked application (see also Section 9.3 SSP Zeroization Methods).

Note that the linked application mentioned above runs inside the module's Tested Operational Environment's Physical Perimeter (TOEPP) but outside its cryptographic boundary. It resides in the same volatile memory area as the module.

9.2 SSP Input-Output Methods

Name	From	To	Format Type	Distribution Type	Entry Type	SFI or Algorithm
API Input	Linked application in the TOEPP	RAM	Plaintext	Manual	Electronic	
API output	RAM	Linked application in the TOEPP	Plaintext	Manual	Electronic	

Table 20: SSP Input-Output Methods

SSPs are passed between the module and the linked application running via API input and output parameters in plaintext. The parameters contain pointers to memory locations inside the shared volatile memory. As required, the module does not output any SSPs to locations outside the TOEPP.

For other key establishment methods, please refer to Section 2.10 Key Establishment.

9.3 SSP Zeroization Methods

Zeroization Method	Description	Rationale	Operator Initiation
Module: Automatic	SSPs temporarily allocated by the module or derived from operator-supplied inputs for use within a single API call are automatically zeroized by the module before the respective function returns.	To prevent the unintended reuse of SSPs stored in volatile memory are explicitly overwritten with zeros before the memory is deallocated. This zeroing process typically takes place at the end of a function call, ensuring that no residual data remains in memory after the function returns.	Not required as this zeroization is triggered automatically by the module itself.
Module: Finalization	Zeroization of SSPs that are stored under control of the module for longer than a single API call (internal Counter DRBG instance).	To prevent the unintended reuse of SSPs, those stored in volatile memory are explicitly overwritten with zeros before the memory is deallocated. This zeroing process occurs during the module_final() API function call or when the module is unloaded, ensuring that no residual data remains in memory after finalization or unloading.	Finalizing / unloading the module.
Procedural	SSPs that are not under control of the module can be procedurally zeroized by the operator.	The operator can perform procedural zeroization SSPs that are not managed by the cryptographic module by overwriting the specific regions of volatile memory where these parameters reside with zeros.	Procedural zeroization is independent of the module's control and must be triggered by the operator.

Table 21: SSP Zeroization Methods

As detailed in Section 9.1 Storage Areas, the module does not store any references to SSPs other than for the internal Counter DRBG instance for longer than a single API call. The module does not provide user-callable means to zeroize SSPs that are under control of the module only for individual API calls because these SSPs are otherwise under full control of the linked application.

SSPs that are:

- Generated internally by the module (e.g., during intermediate steps of cryptographic operations), or
- Derived from operator-supplied inputs within a single API call

are automatically overwritten with zeroes before control is returned to the calling application. This ensures that no residual sensitive data remains in memory once the cryptographic operation is complete. This process is performed internally by the module and does not require caller intervention.

The internal Counter DRBG instance can be explicitly zeroized by invoking the `module_final()` API function. This function securely terminates the module and overwrites the DRBG state with zeroes, ensuring complete zeroization.

Additionally, when the module is unloaded—either by the application or due to process termination—the module's destructor is automatically invoked. This destructor calls the same internal zeroization logic, ensuring that the DRBG state is securely overwritten with zeroes before the module is fully deallocated. A successful return from the `module_final()` function, or the confirmed unloading of the module, indicates that all persistent SSPs maintained by the module have been securely zeroized.

The operator can procedurally zeroize SSPs not under control of the module by overwriting the volatile memory where they are located. Successful zeroization can be ensured by re-reading the zeroized memory and checking whether it was completely overwritten before it is deallocated.

9.4 SSPs

Name	Description	Size - Strength	Type - Category	Generated By	Established By	Used By
AES keys	Keys used for AES encryption / decryption	128, 192, or 256 bits - 128, 192, or 256 bits	AES key - CSP	AES key generation		AES encryption / decryption AES GCM encryption / decryption (random IV) AES GCM encryption / decryption (TLS 1.2) AES GCM encryption / decryption (TLS

Name	Description	Size - Strength	Type - Category	Generated By	Established By	Used By
						1.3) AES CFB encryption / decryption
HMAC keys	Keys used for HMAC generation	>= 112 bits - >= 112 bits	HMAC key - CSP			HMAC generation
EdDSA private keys	Keys used for EdDSA signature generation	32 or 57 bytes - 128 or 224 bits	EdDSA private key - CSP	EdDSA key pair generation		Ed25519 signature generation Ed448 signature generation
EdDSA public keys	Keys used for EdDSA signature verification	32 or 57 bytes - 128 or 224 bits	EdDSA public key - PSP	EdDSA key pair generation		Ed25519 signature verification Ed448 signature verification EdDSA public key validation
ECDSA private keys	Keys used for ECDSA signature generation	Up to 521 bits - 112 to 256 bits	ECDSA private key - CSP	ECDSA key pair generation		ECDSA signature generation
ECDSA public keys	Keys used for ECDSA signature verification	Up to 521 bits - <= 80 (P-192) or 112 to 256 bits (other curves)	ECDSA public key - PSP	ECDSA key pair generation		ECDSA signature verification ECDSA public key validation
DSA public keys	Keys used for DSA signature verification	Up to 3072 bits - <= 80 (L: 1024/N: 160) or 112 to 128 bits (other groups)	DSA public key - PSP			DSA signature verification DSA domain parameter validation

Name	Description	Size - Strength	Type - Category	Generated By	Established By	Used By
RSA private keys	Keys used for RSA signature generation	Variable - 112 to 256 bits	RSA private key - CSP	RSA key pair generation		RSA PKCS1-v1.5 signature generation RSA PSS signature generation RSA signature generation with pre-computed hash (CVL)
RSA public keys	Keys used for RSA signature verification	Variable - <= 80 (1024 bits modulus) or 112 to 256 bits (longer modulus)	RSA public key - PSP	RSA key pair generation		RSA PKCS1-v1.5 signature verification RSA PSS signature verification
KAS-ECC private key	Own private keys used for KAS-ECC	Up to 521 bits - 112 to 256 bits	ECDSA private key - CSP	KAS-SSC ECC		KAS-SSC ECC
KAS-ECC public key	Own public keys used for KAS-ECC	Up to 521 bits - 112 to 256 bits	ECDSA public key - PSP	KAS-SSC ECC		ECDSA public key validation KAS-SSC ECC
KAS-FFC private key	Own private keys used for KAS-FFC	Up to 3072 bits - 112 to 128 bits	DSA private key - CSP	KAS-SSC FFC		KAS-SSC FFC
KAS-FFC public key	Own public keys used for KAS-FFC	Up to 3072 bits - 112 to 128 bits	DSA public key - PSP	KAS-SSC FFC		KAS-SSC FFC
Received KAS-ECC public key	Received public keys used for KAS-ECC	Up to 521 bits - 112 to 256 bits	ECDSA public key - PSP			KAS-SSC ECC
Received KAS-FFC public key	Received public keys used for KAS-FFC	Up to 3072 bits - 112 to 128 bits	DSA public key - PSP			KAS-SSC FFC

Name	Description	Size - Strength	Type - Category	Generated By	Established By	Used By
KTS-IFC private key	Own private keys used for KTS-IFC	Variable - 112 to 256 bits	RSA private key - CSP	KTS-IFC		KTS-IFC
KTS-IFC public key	Own public keys used for KTS-IFC	Variable - 112 to 256 bits	RSA public key - PSP	KTS-IFC		KTS-IFC
Received KTS-IFC public key	Received public keys used for KTS-IFC	Variable - 112 to 256 bits	RSA public key - PSP			KTS-IFC
Shared secret	Output of KAS-ECC and KAS-FFC	Variable - 112 to 256 bits	Shared secret - CSP		KAS-SSC FFC KAS-SSC ECC	
Counter DRBG key	Key of the internal state of the AES-256 Counter DRBG instance	256 bits - 256 bits	Counter DRBG key - CSP	Random number generation		Random number generation AES encryption / decryption AES GCM encryption / decryption (random IV) AES CFB encryption / decryption RSA PKCS1-v1.5 signature generation RSA PSS signature generation ECDSA signature generation DSA domain parameter

Name	Description	Size - Strength	Type - Category	Generated By	Established By	Used By
						validation EdDSA key pair generation ECDSA key pair generation RSA key pair generation AES key generation KAS-SSC FFC KTS-IFC KAS-SSC ECC
Counter DRBG V	Value V of the internal state of the AES-256 Counter DRBG instance	128 bits - 128 bits	Counter DRBG value V - CSP	Random number generation		Random number generation AES encryption / decryption AES GCM encryption / decryption (random IV) AES CFB encryption / decryption RSA PKCS1-v1.5 signature generation RSA PSS signature generation ECDSA signature generation DSA domain

Name	Description	Size - Strength	Type - Category	Generated By	Established By	Used By
						parameter validation EdDSA key pair generation ECDSA key pair generation RSA key pair generation AES key generation KAS-SSC FFC KTS-IFC KAS-SSC ECC
Counter DBRG seed: entropy source output	Output of the entropy source	512 bits - 512 bits	Bit string - CSP	Random number generation		Random number generation
Intermediate key generation values	Temporary values used during key generation	Variable - N/A	Values used for DSA, ECDSA, EdDSA, RSA key generation - CSP	EdDSA key pair generation ECDSA key pair generation RSA key pair generation KAS-SSC FFC KTS-IFC KAS-SSC ECC		
Keying material	Input / output of KTS-IFC	Variable - N/A	Keying material - CSP		KTS-IFC	KTS-IFC

Name	Description	Size - Strength	Type - Category	Generated By	Established By	Used By
KAS-ECC domain parameters	Domain parameters used by the KAS-SSC ECC	Variable - N/A	Domain parameters - PSP			KAS-SSC ECC
KAS-FFC domain parameters	Domain parameters used by the KAS-SSC FFC	Variable - N/A	SP 800-56Ar3 FFC domain parameters - PSP	KAS-SSC FFC		KAS-SSC FFC

Table 22: SSP Table 1

Name	Input - Output	Storage	Storage Duration	Zeroization	Related SSPs
AES keys	API Input API output	RAM:Plaintext	For a single API call	Module: Automatic Procedural	
HMAC keys	API Input API output	RAM:Plaintext	For a single API call	Procedural	
EdDSA private keys	API Input API output	RAM:Plaintext	For a single API call	Module: Automatic Procedural	EdDSA public keys:Paired With
EdDSA public keys	API Input API output	RAM:Plaintext	For a single API call	Module: Automatic Procedural	EdDSA private keys:Paired With
ECDSA private keys	API Input API output	RAM:Plaintext	For a single API call	Procedural	ECDSA public keys:Paired With
ECDSA public keys	API Input	RAM:Plaintext	For a single API call	Procedural	ECDSA private keys:Paired With

Name	Input - Output	Storage	Storage Duration	Zeroization	Related SSPs
	API output				
DSA public keys	API Input API output	RAM:Plaintext	For a single API call	Procedural	
RSA private keys	API Input API output	RAM:Plaintext	For a single API call	Procedural	RSA public keys:Paired With
RSA public keys	API Input API output	RAM:Plaintext	For a single API call	Procedural	RSA private keys:Paired With
KAS-ECC private key	API Input API output	RAM:Plaintext	For a single API call	Procedural	KAS-ECC public key:Paired With Received KAS-ECC public key:Used With
KAS-ECC public key	API Input API output	RAM:Plaintext	For a single API call	Procedural	KAS-ECC private key:Paired With
KAS-FFC private key	API Input API output	RAM:Plaintext	For a single API call	Procedural	KAS-FFC public key:Paired With Received KAS-FFC public key:Used With
KAS-FFC public key	API Input API output	RAM:Plaintext	For a single API call	Procedural	KAS-FFC private key:Paired With
Received KAS-ECC public key	API Input	RAM:Plaintext	For a single API call	Procedural	KAS-ECC private key:Used With

Name	Input - Output	Storage	Storage Duration	Zeroization	Related SSPs
	API output				
Received KAS-FFC public key	API Input API output	RAM:Plaintext	For a single API call	Procedural	KAS-FFC private key:Used With
KTS-IFC private key	API Input API output	RAM:Plaintext	For a single API call	Procedural	KTS-IFC public key:Paired With KTS-IFC public key:Used With
KTS-IFC public key	API Input API output	RAM:Plaintext	For a single API call	Procedural	KTS-IFC private key:Paired With
Received KTS-IFC public key	API Input API output	RAM:Plaintext	For a single API call	Procedural	KTS-IFC private key:Used With
Shared secret	API output	RAM:Plaintext	For a single API call	Procedural	KAS-ECC private key:Derived From Received KAS-ECC public key:Derived From KAS-FFC private key:Derived From Received KAS-FFC public key:Derived From
Counter DRBG key		RAM:Plaintext	Until finalization of the module	Module: Finalization	Counter DBRG seed: entropy source output:Derived From Counter DRBG V:Paired With
Counter DRBG V		RAM:Plaintext	Until finalization of the module	Module: Finalization	Counter DRBG key:Paired With

Name	Input - Output	Storage	Storage Duration	Zeroization	Related SSPs
Counter DBRG seed: entropy source output		RAM:Plaintext	For a single API call	Module: Automatic	
Intermediate key generation values		RAM:Plaintext	For a single API call	Module: Automatic	Counter DRBG key:Derived From EdDSA private keys:Generates ECDSA public keys:Generates RSA private keys:Generates RSA public keys:Generates ECDSA private keys:Generates EdDSA public keys:Generates KAS-ECC private key:Generates KAS-ECC public key:Generates KAS-FFC private key:Generates KAS-FFC public key:Generates KTS-IFC private key:Generates KTS-IFC public key:Generates Counter DRBG V:Derived From
Keying material	API Input API output	RAM:Plaintext	For a single API call	Procedural	KTS-IFC private key:Decrypts Received KTS-IFC public key:Encrypts
KAS-ECC domain parameters	API Input API output	RAM:Plaintext	For a single API call	Procedural	KAS-ECC private key:Used With KAS-ECC public key:Used With Received KAS-ECC public key:Used With
KAS-FFC domain parameters	API Input API output	RAM:Plaintext	For a single API call	Procedural	KAS-FFC private key:Used With KAS-FFC public key:Used With Received KAS-FFC public key:Used With

Table 23: SSP Table 2

For the use of the above SSPs by the module's approved services, please refer to Section 4.3 Approved Services. The module's non-approved services do not access any of the module's SSPs other than the Counter DRBG state, which is permitted per IG 2.4.A. For details on the use of the DRBG by the approved and non-approved services, please refer to Section 2.8 RBG and Entropy.

Note that the HMAC key used for the integrity test is not considered as an SSP. Keys used by non-approved services are also not considered as SSPs.

9.5 Transitions

Information on the transitions for the CMVP-approved algorithms and security functions are provided in SP 800-57 Part 1 Revision 5 and SP 800-131A Revision 2 as well as on the NIST website. At the time of publication of this Security Policy, the following transitions are identified, which take effect in 2031:

- Transition of the minimum security strength from 112 to 128 bits.
- Deprecation of SHA-1.

10 Self-Tests

10.1 Pre-Operational Self-Tests

Algorithm or Test	Test Properties	Test Method	Test Type	Indicator	Details
HMAC-SHA2-256 (A5497)	Key length: 32 bytes	The integrity of the module binary is tested by computing an HMAC over the entire shared library file. The correct reference value is stored in a separate file with the file extension "sha256".	SW/FW Integrity	Return code < 0 in case of a failure.	The HMAC-SHA2-256 is self-tested using a KAT before this integrity test is performed

Table 24: Pre-Operational Self-Tests

The module only implements one pre-operational self-test. This test is performed under control of the module when the function `crypt_init` is called by the operator after the module was loaded. This function either returns `RC_FIPS_APPROVED` (value “1”) when all self-tests passed or an error code when something went wrong during initialization. Only when the return value is `RC_FIPS_APPROVED`, the module returns pointers to the exported APIs required to execute the module’s other security services.

In addition to the pre-operational self-test described above, the module also automatically executes the Cryptographic Algorithm Self-Tests (CASTs) described in Section 10.2 Conditional Self-Tests during its initialization. In case of a failure in at least one of the self-tests executed during start-up, the module enters an error state (see Section 10.4 Error States for details).

Please note that the module neither implements a pre-operational bypass test, as it does not implement a bypass functionality, nor a pre-operational critical function test, as all functions critical to its secure operation are already covered by other self-tests.

10.2 Conditional Self-Tests

Algorithm or Test	Test Properties	Test Method	Test Type	Indicator	Details	Conditions
HMAC-SHA2-256 (A5497)	Key length: 20 bytes	KAT	CAST	Return code < 0 in case of a failure.	HMAC generation	Module initialization
SHAKE-128 (A5497)	N/A	KAT	CAST	Return code < 0 in	XOF generation	Module initialization

Algorithm or Test	Test Properties	Test Method	Test Type	Indicator	Details	Conditions
				case of a failure.		
SHAKE-256 (A5497)	N/A	KAT	CAST	Return code < 0 in case of a failure.	XOF generation	Module initialization
SHA-1 (A5497)	N/A	KAT	CAST	Return code < 0 in case of a failure.	Hash generation	Module initialization
SHA2-224 (A5497)	N/A	KAT	CAST	Return code < 0 in case of a failure.	Hash generation	Module initialization
SHA2-256 (A5497)	N/A	KAT	CAST	Return code < 0 in case of a failure.	Hash generation	Module initialization
SHA2-384 (A5497)	N/A	KAT	CAST	Return code < 0 in case of a failure.	Hash generation	Module initialization
SHA2-512 (A5497)	N/A	KAT	CAST	Return code < 0 in case of a failure.	Hash generation	Module initialization
SHA3-224 (A5497)	N/A	KAT	CAST	Return code < 0 in case of a failure.	Hash generation	Module initialization
SHA3-256 (A5497)	N/A	KAT	CAST	Return code < 0 in	Hash generation	Module initialization

Algorithm or Test	Test Properties	Test Method	Test Type	Indicator	Details	Conditions
				case of a failure.		
SHA3-384 (A5497)	N/A	KAT	CAST	Return code < 0 in case of a failure	Hash generation	Module initialization
SHA3-512 (A5497)	N/A	KAT	CAST	Return code < 0 in case of a failure	Hash generation	Module initialization
DSA SigVer (FIPS186-4) (A5497)	Group: FB (L: 2048/N: 224)	KAT	CAST	Return code < 0 in case of a failure	Signature verification using a fixed 32-byte hash	Module initialization
ECDSA SigGen (FIPS186-5) (A5497)	Curve: P-256	KAT	CAST	Return code < 0 in case of a failure	Signature generation using a fixed 32-byte hash	Module initialization
ECDSA SigVer (FIPS186-5) (A5497)	Curve: P-256	KAT	CAST	Return code < 0 in case of a failure	Signature verification using a fixed 32-byte hash	Module initialization
EDDSA SigGen (A5497) with Edwards25519	Curve: Edwards25519	KAT	CAST	Return code < 0 in case of a failure	Signature generation	Module initialization
EDDSA SigVer (A5497) with Edwards25519	Curve: Edwards25519	KAT	CAST	Return code < 0 in case of a failure	Signature verification	Module initialization

Algorithm or Test	Test Properties	Test Method	Test Type	Indicator	Details	Conditions
EDDSA SigGen (A5497) with Edwards448	Curve: Edwards448	KAT	CAST	Return code < 0 in case of a failure	Signature generation	Module initialization
EDDSA SigVer (A5497) with Edwards448	Curve: Edwards448	KAT	CAST	Return code < 0 in case of a failure	Signature verification	Module initialization
RSA SigGen (FIPS186-5) (A5497)	Modulus size: 2048 bits, Padding: PKCS1-v1.5	KAT	CAST	Return code < 0 in case of a failure	Signature generation using a fixed 32-byte hash	Module initialization
RSA SigVer (FIPS186-5) (A5497)	Modulus size: 2048 bits, Padding: PKCS1-v1.5	KAT	CAST	Return code < 0 in case of a failure	Signature verification using a fixed 32-byte hash	Module initialization
KAS-ECC-SSC Sp800-56Ar3 (A5497)	Curve: P-256	KAT	CAST	Return code < 0 in case of a failure	Shared secret computation with two fixed key pairs	Module initialization
KAS-FFC-SSC Sp800-56Ar3 (A5497)	Group: FB (L: 2048/N: 224) and ffdhe2048	KAT	CAST	Return code < 0 in case of a failure	Shared secret computation with fixed keys (with g^x > p)	Module initialization
AES-ECB (A5497)	Key length: 128, 192, and 256 bits	KAT	CAST	Return code < 0 in case of a failure	Encryption and decryption	Module initialization
AES-GCM (A5497)	Key length: 128 bits	KAT	CAST	Return code < 0 in case of a failure	Encryption and decryption	Module initialization

Algorithm or Test	Test Properties	Test Method	Test Type	Indicator	Details	Conditions
Counter DRBG (A5497)	Key length: 256 bits	KAT	CAST	Return code < 0 in case of a failure	Instantiation with fixed entropy input, generation of random bytes with additional input (generated bytes are tested against known answer), reseed with fix entropy input and again generation of random bytes, now without additional input (generated bytes are tested against known answer)	Module initialization
DSA PQGGen (FIPS186-4) (A5497)	Group: FB (L: 2048/N: 224)	KAT	CAST	Return code < 0 in case of a failure	Generation of p, q, and g with fixed randomness	Module initialization
DSA PQGVer (FIPS186-4) (A5497)	Group: FB (L: 2048/N: 224)	KAT	CAST	Return code < 0 in case of a failure	Domain parameter validation of fixed, correct parameters	Module initialization
Entropy source: start-up / continuous health tests	N/A	Fault-detection test	CAST	Return code < 0 in case of a failure	APT and RCT as specified in SP 800-90B as well as additional variations thereof performed on 6000 symbols.	Module initialization
RSA SigGen and SigVer (FIPS186-5) (A5497)	Padding: PKCS1-v1.5	PCT	PCT	Return code < 0 in case of a failure	RSA signature generation and verification of a random 32-byte pseudo-hash	After RSA and KTS-IFC key pair generation and RSA key pair import
RSA key-pair consistency (SP 800-56Br2)	N/A	PCT	PCT	Return code < 0 in case of a failure	Key-pair consistency check as specified in Section 6.4.1.2.3 of SP 800-56Br2	After RSA and KTS-IFC key pair generation
ECDSA SigGen and SigVer	N/A	PCT	PCT	Return code < 0 in	ECDSA signature generation and verification of a random 32-byte pseudo-hash	After ECDSA key pair

Algorithm or Test	Test Properties	Test Method	Test Type	Indicator	Details	Conditions
(FIPS186-5) (A5497)				case of a failure		generation and import
KAS-ECC pair-wise consistency (SP 800-56Ar3)	N/A	PCT	PCT	Return code < 0 in case of a failure	Key-pair consistency check as specified in Section 5.6.2.1.4 of SP 800-56Ar3	After ECDSA and KAS-ECC key pair generation and import
EdDSA SigGen and SigVer (FIPS186-5) (A5497)	N/A	PCT	PCT	Return code < 0 in case of a failure	EdDSA signature generation and verification of a fixed 7-byte message	After EdDSA key pair generation and import
KAS-FFC pair-wise consistency (SP 800-56Ar3)	N/A	PCT	PCT	Return code < 0 in case of a failure	Key-pair consistency check as specified in Section 5.6.2.1.4 of SP 800-56Ar3	After KAS-FFC key pair generation and import

Table 25: Conditional Self-Tests

Since the module does not implement the corresponding functionality, it does not implement a conditional software/firmware load test, manual entry test, bypass test, or critical function test.

As explained in Section 2.2 Tested and Vendor Affirmed Module Version and Identification, the module can be configured to use different PAAs for certain algorithms. These PAAs must be configured before the module's self-tests are performed during module initialization. During module initialization, the module performs the conditional self-tests using the activated PAAs. While the module is in its initialized state, re-configuration of the activated PAAs is blocked. Changing the activated PAAs thus requires a re-initialization of the module (see Section 11.2 Administrator Guidance for details).

Note that the module also implements self-tests for some non-approved algorithms (e.g., MD5 and RC4), but these self-tests are not listed here.

10.3 Periodic Self-Test Information

Algorithm or Test	Test Method	Test Type	Period	Periodic Method
HMAC-SHA2-256 (A5497)	The integrity of the module binary is tested	SW/FW Integrity	On demand	Module reinitialization.

Algorithm or Test	Test Method	Test Type	Period	Periodic Method
	by computing an HMAC over the entire shared library file. The correct reference value is stored in a separate file with the file extension "sha256".			

Table 26: Pre-Operational Periodic Information

Algorithm or Test	Test Method	Test Type	Period	Periodic Method
HMAC-SHA2-256 (A5497)	KAT	CAST	On demand	Module reinitialization.
SHAKE-128 (A5497)	KAT	CAST	On demand	Module reinitialization.
SHAKE-256 (A5497)	KAT	CAST	On demand	Module reinitialization.
SHA-1 (A5497)	KAT	CAST	On demand	Module reinitialization.
SHA2-224 (A5497)	KAT	CAST	On demand	Module reinitialization.
SHA2-256 (A5497)	KAT	CAST	On demand	Module reinitialization.
SHA2-384 (A5497)	KAT	CAST	On demand	Module reinitialization.
SHA2-512 (A5497)	KAT	CAST	On demand	Module reinitialization.
SHA3-224 (A5497)	KAT	CAST	On demand	Module reinitialization.
SHA3-256 (A5497)	KAT	CAST	On demand	Module reinitialization.
SHA3-384 (A5497)	KAT	CAST	On demand	Module reinitialization.
SHA3-512 (A5497)	KAT	CAST	On demand	Module reinitialization.
DSA SigVer (FIPS186-4) (A5497)	KAT	CAST	On demand	Module reinitialization.
ECDSA SigGen (FIPS186-5) (A5497)	KAT	CAST	On demand	Module reinitialization
ECDSA SigVer (FIPS186-5) (A5497)	KAT	CAST	On demand	Module reinitialization
EDDSA SigGen (A5497) with Edwards25519	KAT	CAST	On demand	Module reinitialization

Algorithm or Test	Test Method	Test Type	Period	Periodic Method
EDDSA SigVer (A5497) with Edwards25519	KAT	CAST	On demand	Module reinitialization
EDDSA SigGen (A5497) with Edwards448	KAT	CAST	On demand	Module reinitialization
EDDSA SigVer (A5497) with Edwards448	KAT	CAST	On demand	Module reinitialization
RSA SigGen (FIPS186-5) (A5497)	KAT	CAST	On demand	Module reinitialization
RSA SigVer (FIPS186-5) (A5497)	KAT	CAST	On demand	Module reinitialization
KAS-ECC-SSC Sp800-56Ar3 (A5497)	KAT	CAST	On demand	Module reinitialization
KAS-FFC-SSC Sp800-56Ar3 (A5497)	KAT	CAST	On demand	Module reinitialization
AES-ECB (A5497)	KAT	CAST	On demand	Module reinitialization
AES-GCM (A5497)	KAT	CAST	On demand	Module reinitialization
Counter DRBG (A5497)	KAT	CAST	On demand	Module reinitialization
DSA PQGGen (FIPS186-4) (A5497)	KAT	CAST	On demand	Module reinitialization
DSA PQGVer (FIPS186-4) (A5497)	KAT	CAST	On demand	Module reinitialization
Entropy source: start-up / continuous health tests	Fault-detection test	CAST	On demand	Module reinitialization
RSA SigGen and SigVer (FIPS186-5) (A5497)	PCT	PCT	On demand	RSA and KTS-IFC key pair generation and import
RSA key-pair consistency (SP 800-56Br2)	PCT	PCT	On demand	RSA and KTS-IFC key pair generation
ECDSA SigGen and SigVer (FIPS186-5) (A5497)	PCT	PCT	On demand	ECDSA key pair generation and import

Algorithm or Test	Test Method	Test Type	Period	Periodic Method
KAS-ECC pair-wise consistency (SP 800-56Ar3)	PCT	PCT	On demand	ECDSA and KAS-ECC key pair generation and import
EdDSA SigGen and SigVer (FIPS186-5) (A5497)	PCT	PCT	On demand	EdDSA key pair generation and import
KAS-FFC pair-wise consistency (SP 800-56Ar3)	PCT	PCT	On demand	KAS-FFC key pair generation and import

Table 27: Conditional Periodic Information

The module does not perform any periodic self-testing on its own. When desired, the operator can manually perform periodic self-testing using the means described in Section 10.5 Operator Initiation of Self-Tests.

10.4 Error States

Name	Description	Conditions	Recovery Method	Indicator
Initialization error	The module remains in its uninitialized state. No cryptographic services are available.	Module initialization failed due to self-test failure.	Manually by calling the crypt_init function.	Return code < 0 during module initialization.
Operational error	A conditional self-test performed after module initialized failed. Output of the affected data is inhibited.	PCT failure.	The module automatically recovers from this error state.	Return code < 0 returned by the function call that invoked the failed self-test.

Table 28: Error States

The module implements two different error states. In case of a failure during the self-tests performed during module start-up (i.e., the pre-operational integrity test and the CASTs), the module enters an error state that requires operator intervention to recover from. An indication of this error state is provided by the return code of the `crypt_init` function, which must be called during module initialization. If the module initialization fails due to failed self-tests or any other problem, the module does not return pointers to its other APIs, which are required to execute the module's cryptographic security services. It also behaves as if it is still in its uninitialized state, thereby preventing the use of any previously returned function pointers. Exiting this error state is possible by re-initializing the module (see Section 10.5 Operator Initiation of Self-Tests for details).

In case of a failure during a conditional self-test that is not automatically performed during start-up (see Section 10.2 Conditional Self-Tests), the module briefly enters a different, transient error state. An error indicator is provided by the return value of the function that invoked the self-test. Output of the generated or imported key pair that caused the conditional self-test error is inhibited. Afterwards, the module automatically recovers from this error state. The concerned conditional self-test is repeated the next time the corresponding operation is performed (e.g., the pair-wise consistency tests are performed for each generated key pair).

Table 29 lists the expected error codes (error indicators) related to failures of performed self-tests. More detailed information about which self-tests performed during module initialization failed can be obtained using the `sec_crypto_get_feature_info` API.

Error cause	Returned error code
Integrity test failure	`ERR_CRYPT__CHECKSUM`
Error reading shared library or checksum file	`ERR_CRYPT__CHECKSUMIO`
CAST failure	`ERR_CRYPT__ALGTEST`
Entropy source self-test failure	`ERR_CRYPT__RND`
PCT failure	`ERR_CRYPT__VALIDATION_FAILED`

Table 29: Error Causes and Expected Return Codes

Please note that the module does not enter an error state when a failure unrelated to the conditional self-tests is detected after successful module initialization (e.g., because an invalid padding was detected, memory could not be allocated, or an assurance check failed). Instead, the function return values indicate to the operator the type of occurred failure.

10.5 Operator Initiation of Self-Tests

All self-tests listed in Section 10.3 Periodic Self-Test Information with “module initialization” listed as their execution condition can be executed on-demand using the ‘initialize and self-test’ service described in Section 4.3 Approved Services. The exact procedure to use this service depends on the current state of the module:

- When the module is in its initialized state, the module must first be finalized using the `module_final` function. Then, the module must be re-initialized by calling `crypt_init` function.
- When the module is not yet initialized or is in its error state, only the `crypt_init` function must be called.

11 Life-Cycle Assurance

11.1 Installation, Initialization, and Startup Procedures

Two files are provided for the installation of the module as explained in Section 2.2 Tested and Vendor Affirmed Module Version and Identification. The file names depend on the operating system for which the module is provided. The file names for the tested operational environments are listed in Table 30.

Package	Shared library file	HMAC file	SHA2-256 checksum
aix-6.1-ppc-64	libslcryptokernel.so	libslcryptokernel.so.sha256	D5E4BC00EB784AE9F0179361B7189B4959 33ACFFEF50D1789A659E6F5827A364
aix-7.2-ppc-64	libslcryptokernel.so	libslcryptokernel.so.sha256	1A5D901BEA61ED1D0BFC138A5BF9698C4 C076A88AADAD60AB94506A21062F5C2
hpux-b.11.31-ia-64	libslcryptokernel.so	libslcryptokernel.so.sha256	4B55441FA1F732499A20F979A00FEF335C 7E252804DFED956356959DFF1F157A
linux-gcc-11.2-armv8-64	libslcryptokernel.so	libslcryptokernel.so.sha256	5106BBAA77B89DB6B320CCF46706FF92C D343DD55388AB1D58637D225F6E580D
linux-gcc-4.3-ia-64	libslcryptokernel.so	libslcryptokernel.so.sha256	3AFF87073CFC5397525577A3141F60D8A9 C08458D5791CB1E01FD73C715040E0
linux-gcc-4.3-s390x-64	libslcryptokernel.so	libslcryptokernel.so.sha256	B649BD3CED0C78473F27A102C48A957AD A35399CDDCF239B31D1AEB89F93CB65
linux-gcc-4.3-x86-64	libslcryptokernel.so	libslcryptokernel.so.sha256	FAAEA6FE2320E29990DDB52DE893E33D8 5186E5F3C8DE255510355648E22D5FF
linux-gcc-4.8-ppcle-64	libslcryptokernel.so	libslcryptokernel.so.sha256	8DA6D5355BB2A0DF40D105662696AF434 A537D2E3E651556F0D3689F80CE1B29
linux-musl-1.2.4-x86-64	libslcryptokernel.so	libslcryptokernel.so.sha256	881BF8113DB20671975EF4EBF30957F8D9 916526D88A6BC1388AA6F3CD2F36E1
macosx-arm-64	libslcryptokernel.dylib	libslcryptokernel.dylib.sha256	DA40327726E62140992D4FDE9D3A9CA55 7020750F81F45D68E23B9BBC9D8DF7A
macosx-x86-64	libslcryptokernel.dylib	libslcryptokernel.dylib.sha25	FEBD7C3E85E0F0817D6B903B35E93060E 231368784028CAE2DE8C672076DFDF1
sunos-5.10-sparc-64	libslcryptokernel.so	libslcryptokernel.so.sha256	39653C77CFF17DD9EFE61CF6F31F7FB1B AF580DF4D6585CECB68F65832278EDF
sunos-5.10-x86-64	libslcryptokernel.so	libslcryptokernel.so.sha256	F9260FB594EE2E62247518CFCC4EDA4A4 E803B5681D7C77042E68FCE47E60392
windows-x86-64	slcryptokernel.dll	slcryptokernel.dll.sha256	42BD86658272851A24510F14316491B0805 6B421012DF0C13E08A9F39E5AB92A

Table 30: Module File Names and Checksums

The module is delivered as part of multiple applications developed by SAP SE. Each application's installation package contains the FIPS 140-3 certified module. The Crypto Officer should use the SHA2-256 checksum provided in Table 30 to check that the desired version of the module is installed.

The Crypto Officer should also make sure that the files are installed with the access permissions shown in Table 31. The following terms are used in this table:

- Administrator: A Crypto Officer with administrative rights of the system on which the module is to be installed. This may also refer to any program executed using the administrator's account on the system.
- Non-Administrator: A Crypto Officer without administrative rights on the system on which the module is to be installed. This may also refer to any program executed without administrative rights on the system.

File	Non-Administrator	Administrator
Shared library	Read, execute	Read, write, execute
HMAC file	Read	Read, write

Table 31: File Access Permissions

Afterwards, the module can be linked to the operator's target application at link time or runtime as a shared library. After the module is linked and then loaded by the operating system, the Crypto Officer **shall** call the `crypt_init` function to initialize the module under consideration of the restrictions given in Section 2.4 Modes of Operation. Thereby, they can provide additional data to be used in the derivation of the additional input and personalization string used for the instantiation of the module's internal Counter DRBG instance (see Section 2.8 RBG and Entropy).

When the `crypt_init` function is called, the module performs the self-tests as indicated in Section 10 Self-Tests to verify the correct operation of the module's functionality and its integrity. The Crypto Officer **shall** make sure that the HMAC file, which contains the HMAC-SHA2-256 reference value for the integrity test, has not been modified. After the function terminates, the Crypto Officer **shall** check its return value. If this value does not equal "1", then the module's initialization has failed, and the module's cryptographic functionality cannot be used. If initialization fails, the Crypto Officer should further examine the return value to determine the cause of failure and attempt to rectify the problem. If initialization succeeds, the Crypto Officer will receive a list of function pointers that allow using the module's approved cryptographic methods.

After the module's successful initialization, the Crypto Officer may, at any time, finalize the module by calling `module_final` and initialize it by invoking `crypt_init` again. As this constitutes a re-initialization of the module, they **shall** again follow the initialization procedure outlined above. After the module has been finalized, the function pointers returned by the `crypt_init` function are disabled thereby preventing the further use of the module's cryptographic security services.

Detailed explanations of the functions `crypt_init` and `module_final` including their parameters and return values can be found in the guidance documentation (see Section 11.2 Administrator Guidance for details).

11.2 Administrator Guidance

The administrator and non-administrator guidance is provided in the following documents that are provided by SAP SE together with the module:

- “FIPS PUB 140-3, Installation Guide of SAP CommonCryptoLib Crypto Kernel v8.6.1” (v1.2, dated 2024-07-26),
- “FIPS PUB 140-3, Cryptographic Ports and Interfaces of SAP CommonCryptoLib Crypto Kernel v8.6.1” (v1.7, dated 2024-08-13).

These documents provide a full reference of all Application Programming Interfaces (APIs) that can be used to invoke the module’s services. Important excerpts of these documents are also reproduced in this Security Policy (see Section 11.1 Installation, Initialization, and Startup Procedures).

More information on the use of the implemented entropy source is provided in the document “FIPS PUB 140-3, SP 800-90B Non-Proprietary Public Use Document, SAP CommonCryptoLib Entropy Collector 1.0.0” (v1.3, dated 2024-08-07).

Besides the information provided in these guidance documents and the previous sections of this Security Policy, the Crypto Officer **shall** also consider the following statements when intending to operate the module in an approved way:

- The module **shall** be operated in its approved mode described in Section 2.4 Modes of Operation. It **shall not** be used in its non-compliant state.
- The requirements for the implemented cryptographic algorithms citied in Section 2.7 Algorithm Specific Information **shall** be followed.
- The application linked to the module **shall** only interact with the module through the defined interfaces and not modify any objects allocated by the module while they are in use or if a pointer to the objects is to be passed to the module for another API call.
- The steps for procedural zeroization given in Section 9.3 SSP Zeroization Methods **shall** be observed.

11.3 Non-Administrator Guidance

The same set of guidance is applicable to administrator and non-administrator users of the module. Therefore, please refer to Section 11.2 Administrator Guidance for the non-administrator guidance.

11.4 Design and Rules

Please refer to Section 11.1 Installation, Initialization, and Startup Procedures and Section 11.2 Administrator Guidance for a description of the module’s rules of operation.

11.5 Maintenance Requirements

The module does not require any maintenance.

11.6 End of Life

The module does not store any SSPs persistently. Secure sanitation of the module can thus be performed by using the ‘finalize and zeroize service to zeroize all SSPs that are under control of the module and by procedurally zeroizing of all other SSPs that are under control of the operator. For details, please see Section 4.3 Approved Services and Section 9.3 SSP Zeroization Methods.

12 Mitigation of Other Attacks

12.1 Attack List

The module implements the following measures to mitigate attacks other than those already addressed by functionality required by FIPS 140-3 Security Level 1:

- The module implements blinding for RSA private key operations (decryption and signature generation) and ECDSA signature generation to mitigate timing attacks and other side-channel attacks. The blinding factor is randomly generated.

12.2 Mitigation Effectiveness

The use of a random blinding factor that is unknown to an attacker makes it more difficult to perform successful timing attacks on RSA private key and ECDSA signature generation operations, as the execution time is not only dependent on the private key value. However, it is important to note that blinding does not completely eliminate these attacks.

12.3 Guidance and Constraints

As blinding is performed transparently within the boundary of the cryptographic module, no user configuration or interaction is involved.