Symantec Corporation

Symantec PGP Cryptographic Engine FIPS 140-2 Non-proprietary Security Policy

Document Version 1.0
Module Version 4.4
Revision Date 04/13/2020

Author	Date	Change
Symantec	04/13/2020	Initial release

Table of Contents

1 Introduction

The Symantec PGP Cryptographic Engine (SW Version 4.4) (hereafter referred to as the "cryptographic module" or the "module") is a software only cryptographic module validated to the standards set forth by the *FIPS PUB 140-2 Security Requirements for Cryptographic Modules* document published by the National Institute of Standards and Technology (NIST). The module is intended to meet the security requirements of FIPS 140-2 Level 1 overall.

This document, the *Symantec PGP Cryptographic Engine FIPS 140-2 Non-proprietary Security Policy*, also referred to as the *Security Policy*, specifies the security rules under which the module must operate.

2 Module Specifications

The PGP Cryptographic Engine (SW Version 4.4) is a software-only cryptographic module embodied as a shared library binary that executes on general-purpose computer systems and is available on a number of operating systems. The specific operating system and version to be validated is specified in the "Operational Environment" section of this document.

The PGP Cryptographic Engine cryptographic module is accessible to client applications through an application-programming interface (API). The module provides a FIPS Approved mode of operation and a non-FIPS mode of operation, both of which are described in the "Approved Mode of Operation" section of this document.

For the purposes of FIPS 140-2, the PGP Cryptographic Engine is classified as a multi-chip standalone module.

2.1 Approved Algorithms

The PGP Cryptographic Engine implements the following Approved algorithms in the FIPS Approved mode of operation.

CAVP Cert	Algorithm	Standard	Mode/ Method	Key Lengths, Curves or Moduli	Use	Service Name
C 511, C 512, C 513, C 514, C 515, C 516, C 517, C 518, C 519, C 520, C 521, C 522	AES	FIPS 197 SP 800-38A	ECB CBC CFB128	128 192 256	Data Encryption / Decryption	Symmetric Cipher
C 511, C 512, C 513, C 514, C 515, C 516, C 517, C 518, C 519, C 520, C 521, C 522	AES	FIPS 197 SP 800-38D	GCM	128 192 256	Data Encryption / Decryption	Authenticated Encryption
C 511, C 512, C 513, C 514, C 515, C 516, C 517, C 518, C 519, C 520, C 521, C 522	AES	FIPS 197 SP 800-38F	KW KWP	128 192 256	Key Wrapping / Unwrapping	Key Wrapping
C 511, C 512, C 513, C 514, C 515, C 516, C 517, C 518, C 519, C 520, C 521, C 522	CVL Partial ECC CDH	SP 800-56A	ECC	P-256 P-384 P-521	Shared Secret Computation	Shared Secret Generation
Vendor Affirmation	CKG	SP 800-133			Cryptographic Key Generation	Various[1]
C 511, C 512, C 513, C 514, C 515, C 516, C 517, C 518, C 519, C 520, C 521, C 522	DRBG	SP 800-90A	CTR-DRBG with AES-256 with Derivation Function	256	Deterministic Random Bit Generation	DRBG
C 511, C 512, C 513, C 514, C 515, C 516, C 517, C 518, C 519, C 520,	DSA	FIPS 186-4	KeyGen and PQGGen	L=2048, N=256 L=3072, N=256	Digital Signature Key Generation	Digital Signature

[1] Please see Appendix A and B for further details on SP 800-133 Cryptographic Key Generation.

C 521, C 522						
C 511, C 512, C 513, C 514, C 515, C 516, C 517, C 518, C 519, C 520, C 521, C 522	DSA	FIPS 186-4	SigGen SHA-256 SHA-384 SHA-512	L=2048, N=256 L=3072, N=256	Digital Signature Generation	Digital Signature
C 511, C 512, C 513, C 514, C 515, C 516, C 517, C 518, C 519, C 520, C 521, C 522	DSA	FIPS 186-4	SigVer SHA-1[2] SHA-224[2] SHA-256 SHA-384 SHA-512	L=1024, N=160 L=2048, N=256 L=3072, N=256	Digital Signature Verification	Digital Signature
C 511, C 512, C 513, C 514, C 515, C 516, C 517, C 518, C 519, C 520, C 521, C 522	ECDSA	FIPS 186-4	KeyGen KeyVer	P-256 P-384 P-512	Key Generation for Digital Signatures	Digital Signature
C 511, C 512, C 513, C 514, C 515, C 516, C 517, C 518, C 519, C 520, C 521, C 522	ECDSA	FIPS 186-4	SigGen SHA-256 SHA-384 SHA-512	P-256 (SHA-256) P-384 (SHA-256, SHA-384) P-521 (SHA-256, SHA-384, SHA-512)	Digital Signature Generation	Digital Signature
C 511, C 512, C 513, C 514, C 515, C 516, C 517, C 518, C 519, C 520, C 521, C 522	ECDSA	FIPS 186-4	SigVer SHA-256 SHA-384 SHA-512	P-256 (SHA-256) P-384 (SHA-256, SHA-384) P-521 (SHA-256, SHA-384, SHA-512)	Digital Signature Verification	Digital Signature
C 511, C 512, C 513, C 514, C 515, C 516, C 517, C 518, C 519, C 520, C 521, C 522	HMAC	FIPS 198-1	HMAC-SHA-1, HMAC-SHA-224, HMAC-SHA-256, HMAC-SHA-384, HMAC-SHA-512, HMAC-SHA3-224, HMAC-SHA3-256, HMAC-SHA3-384, HMAC-SHA3-512	112-512	Message Authentication	Message Authentication
C 511, C 512, C 513, C 514, C 515, C 516, C 517, C 518, C 519, C 520, C 521, C 522	KBKDF	SP 800-108	Counter Mode; HMAC-SHA-1, HMAC-SHA-224, HMAC-SHA-256, HMAC-SHA-384, HMAC-SHA-512	112	Key Derivation	Key Derivation

[2] SHA-1 and SHA-224 are approved only for L=1024,N=160

Vendor Affirmation	KTS	SP 800-56B[3]	RSA	2048 3072	Key Transport	RSA Encrypt/ Decrypt
Vendor Affirmation	PBKDF	SP 800-132	HMAC-SHA-1 HMAC-SHA-224 HMAC-SHA256 HMAC-SHA384 HMAC-SHA512		Key Derivation for Storage Applications	Key Derivation
C 383, C 384, C 385, C 386, C 387, C 388, C 389, C 390, C 391, C 392, C 393, C 478	RSA Decryption Primitive	SP 800-56B		2048	Decryption Primitive (Key Transport)	RSA Encrypt/ Decrypt
C 511, C 512, C 513, C 514, C 515, C 516, C 517, C 518, C 519, C 520, C 521, C 522	RSA	FIPS 186-4	KeyGen	2048 3072	Key Generation for Digital Signature or Key Transport	Digital Signature
C 511, C 512, C 513, C 514, C 515, C 516, C 517, C 518, C 519, C 520, C 521, C 522	RSA	FIPS 186-2	SigGen SHA-256, SHA-384, SHA-512; PKCS 1.5, PKCS PSS	4096	Digital Signature Generation	Digital Signature
C 632, C 633, C 634, C 635, C 636, C 637, C 638, C 639, C 640, C 641, C 642, C 643	RSA	FIPS 186-2	SigVer SHA2-256, SHA2-384, SHA2-512, PKCS 1.5, PKCS PSS	4096	Digital Signature Verification	Digital Signature
C 511, C 512, C 513, C 514, C 515, C 516, C 517, C 518, C 519, C 520, C 521, C 522	RSA	FIPS 186-4	SigGen SHA-256, SHA-384, SHA-512; PKCS 1.5, PKCS PSS	2048 3072	Digital Signature Generation	Digital Signature
C 511, C 512, C 513, C 514, C 515, C 516, C 517, C 518, C 519, C 520, C 521, C 522	RSA	FIPS 186-4	SigVer SHA-1, SHA-224, SHA-256, SHA-384, SHA-512; PKCS 1.5,	1024 2048 3072	Digital Signature Verification	Digital Signature

[3] Module implements SP 800-56B Section 6.3.1 and Section 9.2.3 (KTS-OAEP-basic). See “RSA Decryption Primitive” for associated CVL certificates.

			PKCS PSS			
C 511, C 512, C 513, C 514, C 515, C 516, C 517, C 518, C 519, C 520, C 521, C 522	SHS	FIPS 180-4 FIPS 202	SHA-1, SHA-224, SHA-256, SHA-384, SHA-512, SHA3-224, SHA3-256, SHA3-384, SHA3-512, SHAKE-128, SHAKE-256		Message Digest	Message Digest
C 511, C 512, C 513, C 514, C 515, C 516, C 517, C 518, C 519, C 520, C 521, C 522	Triple-DES	SP 800-67	TDES-ECB TDES-CBC TDES-CFB64	(3-Key) 192 bits	Data Encryption / Decryption	Symmetric Cipher

Table 1 - Approved algorithms

The PGP Cryptographic Engine also implements the following Allowed algorithms:

Algorithm	Caveat	Use
NDRNG	No assurance of the minimum strength of generated keys. The module generates 112 bits of entropy for use in key generation	Seeding for the DRBG

Table 2 - Allowed algorithms

2.2 Non-Approved Algorithms

The PGP Cryptographic Engine module provides the following non-FIPS approved algorithms only in non-FIPS mode of operation:

Algorithms	Service	Role
AES CFB2	Disk Block Encryption / Decryption	User/CO
AES EME2	Disk Block Encryption / Decryption	User/CO
AES PlumbCFB	Encryption / Decryption	User/CO
ARC4	Encryption / Decryption	User/CO
BlowFish	Encryption / Decryption	User/CO
CAST5	Encryption / Decryption	User/CO
DSA with MD-5, MD-2, RIPEMD160, SHA3-224, SHA3-256, SHA3-384, SHA3-512, SHAKE-128, or SHAKE-512 (non-compliant)	Signature Generation and Verification	User/CO
DSA with SHA-1	Signature Generation	User/CO
DSA with modulus size 1024 and larger than 4096 up to 65536 (non-compliant)	Signature Generation and Verification	User/CO
ECDSA with MD-5, MD-2, RIPEMD160, SHA3-224, SHA3-256, SHA3-384, SHA3-512, SHAKE-128, or SHAKE-512 (non-compliant)	Signature Generation and Verification	User/CO
ECDSA with SHA-1 or SHA-224 (non-compliant)	Signature Generation	User/CO
ElGamal	Encryption / Decryption	User/CO
HMAC with MD-5, RIPEMD160, MD-2	Keyed Hash	User/CO
HMAC with keys <112 bits in length	Keyed Hash	User/CO
HMAC with SHAKE-128 or SHAKE-512	Keyed Hash	User/CO
IDEA	Encryption / Decryption	User/CO
KBKDF-SHA3-224, KBKDF-SHA3-256, KBKDF-SHA3-384, KBKDF-SHA3-512, KBKDF-SHAKE-128 or KBKDF-SHAKE-512	Key Derivation	User/CO
MD-2	Hashing	User/CO
MD-5	Hashing	User/CO
OpenPGP S2K Iterated salted	Key Derivation	User/CO
PBKDF-SHA3-224, PBKDF-SHA3-256, PBKDF-SHA3-384, PBKDF-SHA3-512, PBKDF-SHAKE-128 or PBKDF-SHAKE-512	Key Derivation	User/CO
RC2	Encryption / Decryption	User/CO
RIPEMD160	Hashing	User/CO

RSA with MD-5, MD-2, RIPEMD160, SHA3-224, SHA3-256, SHA3-384, SHA3-512, SHAKE-128, or SHAKE-512	Signature Generation and Verification	User/CO
RSA with SHA-1 and SHA-224 (non-compliant)	Signature Generation	User/CO
RSA with modulus size 1024 and larger than 4096 up to 65536 (non-compliant)	Signature Generation and Verification	User/CO
RSA with X509_RAW packing format	Signature Generation and Verification	User/CO
TwoFish	Encryption / Decryption	User/CO

Table 2 – Non-Approved Algorithms

2.3 Cryptographic Boundary

The physical cryptographic boundary is defined as the computer's case that the PGP Cryptographic Engine is installed in and includes all the accompanying hardware. The module's logical cryptographic boundary is defined to be a subset of the PGP Cryptographic Engine binary software library as defined by the "Roles and Services" section of this document.
An operator is accessing (or using) the module whenever one of the library calls is executed through the API and thus the module logical interfaces are provided by the API calls.
Note that the dashed line represents the PGP Cryptographic Engine crypto boundary.

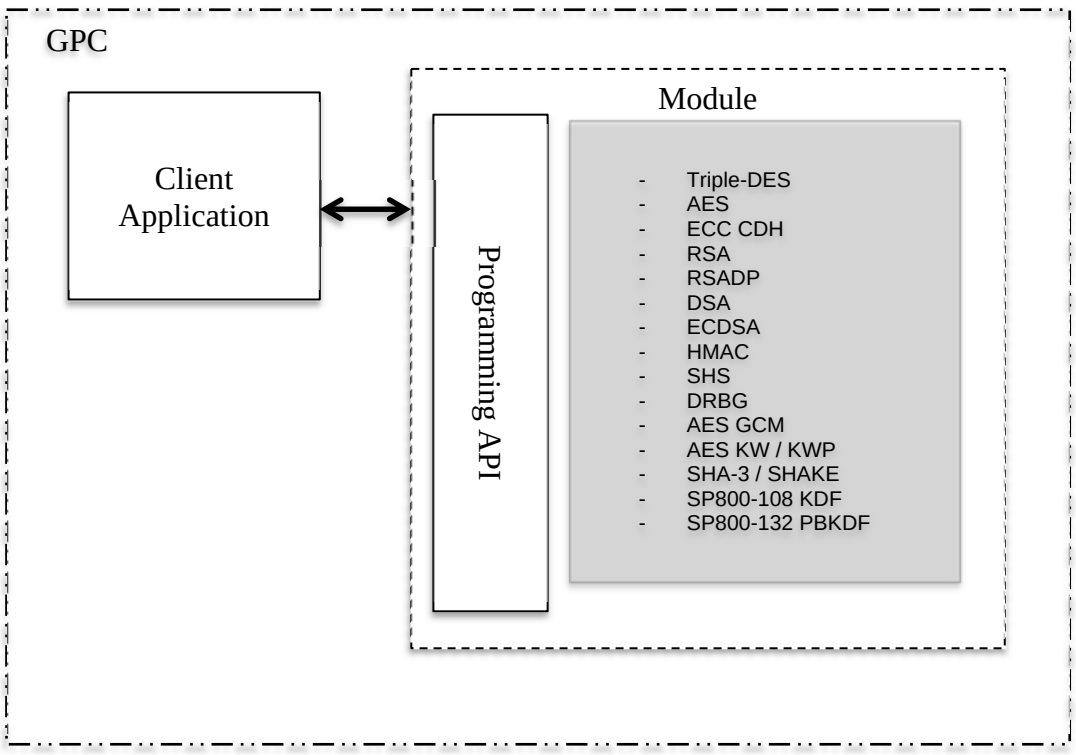

Figure 1 - Module Cryptographic Boundary

2.4 Ports and Interfaces

The module restricts all access to its Critical Security Parameters (CSPs) through the API calls as enumerated in the "Roles and Services" section of this document. This API acts as the logical interface to the module.

Although the computer's physical ports such as keyboards, mouse, displays, and network interfaces provide a means to interact with the GPC, the interface to the cryptographic module is via the module API.

For the purpose of FIPS 140-2, the logical interfaces can be modeled as described in the following table.

Data Input	Parameters passed to the module via API calls.
Data Output	Data returned by the module via API calls.
Control Input	Control Input – API function calls.
Status Output	Error and status codes returned by API calls.

Table 3 - PGP Cryptographic Engine Logical Ports

Input and output data can consist of plain-text, cipher-text, and cryptographic keys as well as other parameters. The module does not support a cryptographic bypass mode.
All data output is inhibited during an error state. Data output is also inhibited during the self-test process.

2.5 Security Level

The PGP Cryptographic Engine Module meets the overall security requirements of FIPS 140-2 Level 1.

Security Requirements Area	Level
Cryptographic Module Specification	1
Cryptographic Module Ports and Interfaces	1
Roles, Services and Authentication	1
Finite State Model	1
Physical Security	N/A
Operational Environment	1
Cryptographic Key Management	1
EMI/EMC	1
Self-Tests	1
Design Assurance	3
Mitigation of Other Attacks	N/A

Table 4 - Module Security Level Specification

2.6 Operational Environment

The following Operating Systems were used to operationally test and validate the PGP Cryptographic Engine to the requirements of FIPS-140-2.

Label	PAA	Operating Environment for testing Software	Operating System
OE #1	AESNI-Enabled	Windows 10 Pro (32-bit) on VMWare ESXi 6.5.0 on Intel® Xenon® CPU E5-2620 v3 @ 2.4GHz with AES-NI enabled	Windows 10 Pro (32-bit) on VMWare ESXi 6.5.0 running on Dell Precision Tower 7910
OE #2	none	Windows 10 Pro (32-bit) on VMWare ESXi 6.5.0 on Intel® Xenon® CPU E5-2620 v3 @ 2.4GHz with AES-NI disabled	Windows 10 Pro (32-bit) on VMWare ESXi 6.5.0 running on Dell Precision Tower 7910
OE #3	AESNI-Enabled	macOS High Sierra 10.13.6 (64-bit) on 2 GHz Intel Core i7 with AES-NI enabled	macOS High Sierra 10.13.6 (64-bit) running on Mac mini Server (Mid 2011)
OE #4	none	macOS High Sierra 10.13.6 (64-bit) on 2 GHz Intel Core i7 with AES-NI disabled	macOS High Sierra 10.13.6 (64-bit) running on Mac mini Server (Mid 2011)
OE #5	AESNI-Enabled	RHEL 7.5 (64-bit) on VMWare ESXi 6.5.0 on Intel® Xenon® CPU E5-2620 v3 @ 2.4GHz with AES-NI enabled	Red Hat Enterprise Linux 7.5 (64-bit) on VMWare ESXi 6.5.0 running on Dell Precision Tower 7910
OE #6	none	RHEL 7.5 (64-bit) on VMWare ESXi 6.5.0 on Intel® Xenon® CPU E5-2620 v3 @ 2.4GHz with AES-NI disabled	Red Hat Enterprise Linux 7.5 (64-bit) on VMWare ESXi 6.5.0 running on Dell Precision Tower 7910
OE #7	AESNI-Enabled	Windows 10 Pro (64-bit) on VMWare ESXi 6.5.0 on Intel® Xenon® CPU E5-2620 v3 @ 2.4GHz with AES-NI enabled	Windows 10 Pro (64-bit) on VMWare ESXi 6.5.0 running on Dell Precision Tower 7910
OE #8	none	Windows 10 Pro (64-bit) on VMWare ESXi 6.5.0 on Intel® Xenon® CPU E5-2620 v3 @ 2.4GHz with AES-NI disabled	Windows 10 Pro (64-bit) on VMWare ESXi 6.5.0 running on Dell Precision Tower 7910
OE #9	AESNI-Enabled	Windows 10 Pro (32-bit) kernel on VMWare ESXi 6.5.0 on Intel® Xenon® CPU E5-2620 v3 @ 2.4GHz with AES-NI enabled	Windows 10 Pro (32-bit) kernel on VMWare ESXi 6.5.0 running on Dell Precision Tower 7910
OE #10	none	Windows 10 Pro (32-bit) kernel on VMWare ESXi 6.5.0 on Intel® Xenon® CPU E5-2620 v3 @ 2.4GHz with AES-NI disabled	Windows 10 Pro (32-bit) kernel on VMWare ESXi 6.5.0 running on Dell Precision Tower 7910
OE #11	AESNI-Enabled	Windows 10 Pro (64-bit) kernel on VMWare ESXi 6.5.0 on Intel® Xenon® CPU E5-2620 v3 @ 2.4GHz with AES-NI enabled	Windows 10 Pro (64-bit) kernel on VMWare ESXi 6.5.0 running on Dell Precision Tower 7910
OE #12	none	Windows 10 Pro (64-bit) kernel on VMWare ESXi 6.5.0 on Intel® Xenon® CPU E5-2620 v3 @ 2.4GHz with AES-NI disabled	Windows 10 Pro (64-bit) kernel on VMWare ESXi 6.5.0 running on Dell Precision Tower 7910

As per FIPS Implementation Guidance (Section G.5), the PGP Cryptographic Engine module will remain compliant in all operational environments for which the binary executable remains unchanged. The Cryptographic Module Validation Program (CMVP) makes no statement as to the correct operation of the module or the security strengths of the generated keys if the specific operational environment is not listed on the validation certificate.

The following is list of representative examples of compliant operational environments:

- Microsoft Windows 7, 8, 8.1 32-bit
- Microsoft Windows 7, 8, 8.1 64-bit
- Any Microsoft Windows version where the module binary executable remains unchanged
- Microsoft Windows 7, 8, 8.1 Kernel 32-bit
- Microsoft Windows 7, 8, 8.1 Kernel 64-bit
- Any Microsoft Windows Kernel version where the module binary executable remains unchanged
- Apple Mac OS X/macOS 10.8, 10.9, 10.10, 10.11, 10.12, 10.13+
- Any Apple macOS version where the module binary executable remains unchanged
- RHEL 6.x and 7.x
- Centos 6.x and 7.x
- Any Linux-based distribution where the module binary executable remains unchanged

The tested operating systems segregate user processes into separate process spaces. Each process space is logically separated from all other processes by the operating system software and hardware. The non-kernel module functions entirely within the process space of the calling application, and implicitly satisfies the FIPS 140-2 requirement for a single user mode of operation.

The kernel mode module is a Windows Kernel export driver and is loaded into the kernel by the kernel on demand when API calls are made that require it to be loaded. In this context, the Windows Kernel is the single operator for the module.

2.7 Approved Mode of Operation

The PGP Cryptographic Engine provides a FIPS 140-2 compliant mode of operation. It is required that applications using the module use only approved algorithms (see Table 1 - Approved algorithms) and follow all security rules in section 3 in order to remain complaint. There are no configuration requirements to operate in FIPS mode – by using approved algorithms and following the security rules you will use the module in FIPS mode. It is possible to use various non-approved algorithms (see section 2.2). This results in using the module in non-FIPS mode; in this case the FIPS 140-2 self-tests are still required to be run and pass validation prior to using the non-approved algorithms.

The client application can, at any time, verify the status by performing the PGPceGetSDKErrorState API call. This function will return either 0 (kPGPError_NoErr) which represents no error or -11446 (kPGPError_SelfTestFailed) which represents the module is in an error state. No other status is returned by this API.

An application can also check the module error state and run all or any specific self-test through making the proper API calls.

3 Security Rules

Following is a list of security requirements that specify the Approved mode of operation and must be adhered to when complying with FIPS 140-2.

1. PGP Cryptographic Engine must be used as described in this document.
2. Installation of the module is the responsibility of the Crypto Officer.
3. The cryptographic module provides a FIPS 140-2 compliant mode of operation. Before the module can be used, it must be initialized as described in the "Approved Mode of Operation" section of this document.
4. The cryptographic module conforms to the EMI/EMC requirements specified by 47 Code of Federal Regulations, Part 15, Subpart B, Unintentional Radiators, Digital Devices, Class B (i.e., for Home use) which vacuously satisfies Class A.
5. Only FIPS approved or allowed cryptographic algorithms as enumerated in the "Supported Algorithms" section of this document are to be used.
6. The cryptographic module inhibits data output during self-tests and error states. The data output interface is logically disconnected from the processes performing zeroization.
7. The zeroization process can be achieved using the appropriate API function: PGPceFreeObject. Note, this function must be called for each key or CSP object created. There is no one single function to zeroize all CSP objects.
8. PGP Cryptographic Engine is designed to meet FIPS 140-2, security Level 1, therefore the module does not provide authentication mechanisms.
9. For those algorithms and modes that required an IV (AES CBC and AES CFB), the consuming application is responsible for generating and supplying an IV which meets SP 800-38A requirements; this can be done by generating a random IV using the approved SP 800-90A DRBG via API call PGPceContextGetRandomBytes.
10. In accordance with FIPS 140-2 IG D.12, the cryptographic module performs Cryptographic Key Generation (CKG) as per SP 800-133 (Vendor Affirmed). The resulting generated symmetric key and/or generated seed for asymmetric key generation, are from the unmodified output of the SP 800-90A DRBG via API call PGPceContextGetRandomBytes.
11. In accordance with SP 800-57, AES Keys used for one encryption mode (eg. CFB) must not be re-used in another mode (eg. ECB).
12. In accordance with SP 800-67rev2, a single Triple-DES key must not be used to encrypt more than 2^{16} 64-bit data blocks. The application using this module is responsible for ensuring that no more than 2^{16} 64-bit data blocks are encrypting using a single Triple-DES key in order to remain complaint.

3.1 Guidance for use of the Approved RSA Algorithm

RSA Signature Generation and Verification must be done using the PGPceSign and PGPceVerify APIs. Only RSA keys generated with kPGPcePublicKeyAlgorithm_RSASignOnly can be used for RSA digital signatures. Please refer to Section 2.1 for approved algorithms and Section 2.2 for non-approved algorithms.

RSA Key Transport and Key Wrapping (encrypt and decrypt) can only use RSA keys generated with kPGPcePublicKeyAlgorithm_RSAEncryptOnly.

3.2 Guidance for use of the Approved SHAKE Algorithms

The SHAKE Algorithms must only be used as standalone hash algorithms. They cannot be used with other higher-level algorithms like HMAC, Digital Signatures, PBKDF, or KBKDF. Please refer to Section 2.1 for approved algorithms and Section 2.2 for non-approved algorithms.

3.3 Guidance for use of the Approved DSA

DSA Signature Generation and Verification must be done using the PGPceSign and PGPceVerify APIs. Please refer to Section 2.1 for approved algorithms and Section 2.2 for non-approved algorithms.

3.4 Guidance for use of the Approved ECDSA Algorithm

ECDSA Signature Generation and Verification must be done using the PGPceSign and PGPceVerify APIs. Only ECDSA keys generated with kPGPcePublicKeyAlgorithm_ECSign can be used for ECDSA digital signatures. Please refer to Section 2.1 for approved algorithms and Section 2.2 for non-approved algorithms.

ECDH secret generation can only use ECDH keys generated with kPGPcePublicKeyAlgorithm_ECEncrypt.

3.5 Guidance for use of the Approved HMAC

HMAC Keys must be a minimum of 112 bits in length. In addition, keys must be at least half the size of the hash used in the HMAC algorithm. For example, when using SHA-256, the key must be at least 128 bits in length.
The following algorithms are not allowed for use in the approved HMAC: MD5, RIPEMD160, and MD2.

3.6 Guidance for use of the Approved PBKDF

For Password-based Key Derivation, the following restrictions apply:

- Keys generated using PBKDF2 shall only be used in data storage applications.
- The minimum password length input shall be 14 characters, which has a strength of approximately 112 bits, assuming a randomly selected password using the extended ASCII printable character set is used.
- For random passwords (that is, a string of characters from a given set of characters in which each character is equally likely to be selected), the strength of the password is given by: S=L*(log N/log 2) where N is the number of possible characters (for example, for the ASCII printable character set N = 95, for the extended ASCII printable character set N = 218) and L is the number of characters. A password of the strength S can be guessed at random with the probability of 1 in 2S.
- The minimum length of the randomly-generated portion of the salt shall be 16 bytes.

- The random portion of the salt must be generated using the unmodified output of the SP 800-90A DRBG using API call PGPceContextGetRandomBytes.
- The iteration count shall be as large as possible, with a minimum of 1000 iterations recommended as per SP800-132.
- The maximum key length is $(2^{32} -1) * hLen$, where hLen is the digest size of the message digest function.
- The minimum key length shall be 112 bits as per SP800-132.
- Derived keys can be used as specified in NIST Special Publication 800-132, Section 5.4, options 1 and 2.
- The following approved hash functions are supported for PBKDF: SHA1, SHA256, SHA384, and SHA512.

3.7 Guidance for use of the Approved KBKDF

For Key-based Key Derivation, the following restrictions apply:

- The input Key Derivation Key (KDK) shall be generated using the unmodified output of the SP 800-90A DRBG using API call PGPceContextGetRandomBytes. The KDK shall be 112 bits or greater.
- Per SP 800-108, the Derived Keying Material (DKM) shall not be used as a key stream for a stream cipher. The DKM shall be 112 bits or greater.
- The following approved keyed hash functions are supported for KBKDF: HMAC-SHA1, HMAC-SHA-224, HMAC-SHA-256, HMAC-SHA-384, and HMAC-SHA-512.

3.8 Roles, Services and Authentication

The module operator is defined as any client application that is linked to the PGP Cryptographic Engine shared library (*PGPce.dll* on the Windows platforms, *PGPce.sys* for the Windows kernel module, *libPGPCE.dylib* on OS X platforms, and *libPGPCE.so.4* on Linux platforms)

The cryptographic module supports two roles (described below). An operator accesses both roles while using the module and the means of access is the same for both roles. A role is implicitly assumed based on the services that are accessed.

The Crypto Officer (CO) is any entity that can install the module library onto the computer system, configure the operating system, and validate the compliance of the module. The Crypto Officer's role is implicitly selected when installing the module or configuring the operating system.

Installation is accomplished by copying the module library file, (one of *libPGPCE.so.4*, *libPGPCE.dylib*, or *PGPce.dll*) and the module signature file, (one of *libPGPCE.so.4.sig*, *libPGPCE.dylib.sig*, or *PGPce.dll.sig)*. The location of the directory is not critical to the installation of the module, except for the Windows kernel module.

The Windows kernel module (*PGPce.sys*) and signature file (*PGPce.sys.sig*) must be installed into the Windows system32\drivers directory (typically C:\Windows\System32\drivers).

The Crypto Officer should call `PGPceGetVersion` inside the module to verify the version. The expected return code is `0x04040000` for this version of the module.

The roles are defined as the following:

- User: Shall be allowed to perform all services provided by the module.
- Crypto Officer: Shall be allowed to perform all services provided by the module and additionally is responsible for the installation of the module.

Role	Authentication Type	Authentication Data
User	N/A	N/A
Crypto Officer (CO)	N/A	N/A

Table 5 – Roles and Required Identification and Authentication

Authentication Mechanism	Strength of Mechanism
N/A	N/A

Table 6 – Strength of Authentication Mechanisms

3.9 Access Control Policy

In the PGP Cryptographic Engine, access to critical security parameters is controlled. A module User or Crypto Officer can only read, modify, or otherwise access the security relevant data through the cryptographic module services provided by the module API interface. This section details the Critical Security Parameters (CSPs) in the cryptographic module that a User or Crypto Officer can access, how the CSPs can be accessed in the cryptographic module, and which services are used for access to the data item.

3.10 Critical Security Parameters and Public Keys

The Critical Security Parameters (CSPs) used by the PGP Cryptographic Engine module are protected from unauthorized disclosure, modification, and substitution.

Definition of CSPs and Public Keys:

- Triple-DES Key - used to Triple-DES encrypt/decrypt data.
- AES Key - used to AES encrypt/decrypt data; used for SP800-38F AES KeyWrap; used for AES GCM Authenticated Encryption.
- AES GCM IV – used for AES GCM Authenticated Encryption.
- RSA Key Pairs (RSA private and public key) - used for signing and verification; used for encrypt/decrypt (SP800-56B KTS).
- DSA Key Pairs (DSA private and public key) - used for signing and verification.
- ECDSA Key Pairs (ECDSA private and public key) - used for signing and verification.
- ECDSA secret k value – used for signing (per FIPS 186-4)
- ECC CDH Key Pairs (ECC CDH private and public key) – used to generate a shared secret.
- ECC CDH Shared Secret – output of the “Shared Secret Generation” service.
- DRBG Entropy Input, Nonce, Personalization String – used for random bit generation.
- DRBG Internal State – The values of V and Key.
- HMAC Key - used for message authentication of data.
- PBKDF Internal State – used for SP800-132 PBKDF Key Derivation.
- PBKDF Password – used for SP800-132 PBKDF Key Derivation.
- PBKDF MK - output of the SP800-132 PBKDF Key Derivation.
- KBKDF Internal State – used for SP800-108 KBKDF Key Derivation.
- KBKDF KDK – used for SP800-108 KBKDF Key Derivation.
- KBKDF DKM – output of the SP800-108 KBKDF Key Derivation.
- Module Integrity Authentication Key – ECDSA P-384 public key used to authenticate the software image of the module.

3.11 Accesses

The types of access to CSPs in the Symantec PGP Cryptographic Engine module are listed in the following table. Note, for the module read and execute are the same access – references to read access apply equally to execute.

Access	Description
create	The item is created.
destroy	The item is destroyed, in other words the data is cleared (actively overwritten) from any memory in the cryptographic module and then that memory is released.
read	The item is accessed for reading and use.
write	The item is modified or changed.

Table 7 - CSP Access Types

3.12 Service to CSP Access Relationship

The following table shows which CSPs are accessed by each service, the role(s) the operator must be in for access, and how the CSP is accessed on behalf of the operator when the service is performed.
Several services provided by the PGP Cryptographic Engine module do not access any CSPs and are included here for completeness.

Service	CO	User	CSPs and Public Keys	create	destroy	read	write
Symmetric Cipher	**X**	**X**	Triple-DES Key AES Key		●	●	
Digital Signature	**X**	**X**	RSA Key Pairs DSA Key Pairs ECDSA Key Pairs	●	●	●	●
	X	**X**	ECDSA secret k value	●	●		
Message Authentication	**X**	**X**	HMAC Key		●	●	
RSA Encrypt/Decrypt	**X**	**X**	RSA Key Pairs	●	●	●	●
Shared Secret Generation	**X**	**X**	ECC CDH Key Pairs	●	●	●	●
	X	**X**	ECC CDH Shared Secret	●	●	●	
Show status	**X**	**X**	N/A				
Run self-tests	**X**	**X**	Module Integrity Authentication Key				
Authenticated Encryption	**X**	**X**	AES Key		●	●	
			AES GCM IV	●	●	●	●

Service	CO	User	CSPs and Public Keys	create	destroy	read	write
Zeroize	X	X	Triple-DES Key AES Key AES GCM IV RSA Key Pairs DSA Key Pairs ECDSA Key Pairs ECDSA secret k value ECC CDH Key Pairs ECC CDH Shared Secret DRBG Entropy Input, Nonce, Personalization String DRBG Internal State HMAC Key PBKDF Internal State PBKDF Password PBKDF MK KBKDF Internal State KBKDF KDK KBKDF DKM		●		
Random Number Generation	X	X	DRBG Entropy Input, Nonce, Personalization String		●	●	
	X	X	DRBG Internal State	●	●		
Key Wrapping	X	X	AES Key		●	●	
Key Derivation	X	X	PBKDF Password		●	●	
	X	X	PBKDF MK	●	●	●	
	X	X	PBKDF Internal State	●	●		
	X	X	KBKDF KDK		●	●	

Service	CO	User	CSPs and Public Keys	create	destroy	read	write
	X	X	KBKDF DKM	●	●	●	
	X	X	KBKDF Internal State	●	●		
Message Digest	X	X	N/A				
Module Initialization	X	X	Module Integrity Authentication Key				
Cryptographic Padding	X	X	N/A				
Object Management	X	X	N/A				
Context Management	X	X	N/A				
Memory Management	X	X	N/A				

Table 8 - Module Services vs Role Access

4 Physical Security Policy

The PGP Cryptographic Engine is implemented as a software module, and as such the physical security section of FIPS 140-2 is not applicable.

Physical Security Mechanisms	Recommended Frequency of Inspection/Test	Inspection/Test Guidance Details
N/A	N/A	N/A

Table 9 - Inspection/Testing of Physical Security Mechanisms

5 Self-Tests

The PGP Cryptographic Engine provides for two forms of self-tests: power-on, and on-demand.

Software integrity test is performed using ECDSA P-384 with SHA-384 signature verification.

The FIPS integrity check and self-test are a mandatory operation and run automatically without operator intervention. The results of the integrity check and self-tests are reported by PGPceGetSDKErrorState().

All data output is prohibited during the self-test process.
If any of these test fail, the module will enter an error state, which can only be cleared by powering down the module. Once in an error state, all further cryptographic operations and data output is disabled.
A client application can also ascertain module at any time by using the PGPceGetSDKErrorState() function. Possible error codes returned by the self-test routines include:

- kPGPError_NoErr (0) – self-test was successful.
- kPGPError_SelfTestFailed (-11446) – self-test Failed.

5.1 Power-Up Tests

The following self-tests will run and in the following order until all the tests have been completed successfully or until one of the tests fail.

Algorithm	Test Attributes
Software Integrity Test	ECDSA P-384/SHA-384 signature verification
TDES	Encrypt, ECB mode, 3 key KAT[4] Encrypt, CFB mode, 3-key KAT Encrypt, CBC mode, 3-key KAT
TDES	Decrypt, ECB mode, 3 key KAT Decrypt, CFB mode, 3 key KAT Decrypt, CBC mode, 3 key KAT
DSA	Sign, Verify (using SHA-256) Pair-wise consistency
AES	Encrypt, CBC mode, 128-bit KAT Encrypt, CFB mode, 128-bit KAT Encrypt, ECB mode, 256-bit KAT
AES	Decrypt, CBC mode, 128-bit KAT Decrypt, CFB mode, 128-bit KAT Decrypt, ECB mode, 256-bit KAT
AES GCM	Encrypt, GCM, 128-bit, 192-bit, 256-bit KAT
AES GCM	Decrypt, GCM, 128-bit, 192-bit, 256-bit KAT
AES KW	KW-AE 128-bit, 192-bit, 256-bit KAT
AES KW	KW-AD 128-bit, 192-bit, 256-bit KAT
AES KWP	KWP-AE 128-bit, 192-bit, 256-bit KAT
AES KWP	KWP-AD 128-bit, 192-bit, 256-bit KAT
RSA	Sign, Verify using 2048-bit with SHA-256 KAT
RSA	2048-bit Encrypt KAT
RSA	2048-bit Decrypt KAT
SHA	SHA-1, SHA-2-256, SHA-2-512 from FIPS 180-4 KAT
HMAC	HMAC-SHA-1, HMAC-SHA-2-256, HMAC-SHA-2-512
SHA-3	HMAC-SHA3-256 KAT
SHAKE	SHAKE-256 KAT
ECDSA	Sign, Verify (using SHA-256) Pair-wise consistency

[4] Known Answer Test

DRBG	CTR_DRBG: AES, 256-bit KAT SP800-90A section 11.3 self-test
ECC CDH	ECC CDH P-256 Primitive “Z” computation KAT
SP 800-132 PBKDF	SP 800-132 with SHA-256 KAT
SP 800-108 KBKDF	SP 800-108 KBKDF with HMAC-SHA-1, HMAC-SHA-256 and HMAC-SHA-512 KAT

Table 10 - Power On Self Tests

5.2 Conditional self-tests

Algorithm	Test Attributes
ECDSA	Sign, Verify (using SHA-256) Pair-wise consistency
RSA	Sign, Verify (using SHA-256) Pair-wise consistency
RSA	Encrypt, Decrypt Pair-wise consistency
DSA	Sign, Verify (using SHA-256) Pair-wise consistency
NDRNG	Continuous Random Number Generation test
DRBG	Per FIPS 140-2 Implementation Guidance (Section 9.8) allowances, the DRBG does not perform the Continuous Random Number Generation test

Table 11 - Conditional self-tests

5.3 On-Demand Tests

Power-up tests can be initiated on-demand by power cycling the module. The client application can optionally initiate a specific test or all tests on demand by using the PGPceRunSelfTest() or PGPceRunAllSelfTests() functions respectively. Note that if the on-demand tests fail, the module will enter an error state in a manner identical to the power-on self-tests.

6 Mitigation of Other Attacks

The Mitigation of Other attacks security section of FIPS 140-2 is not applicable to the PGP Cryptographic Engine module. The module is not designed to mitigate against attacks outside the scope of FIPS 140-2.

Other Attacks	Mitigation Mechanism	Specific Limitations
N/A	N/A	N/A

Table 12 – Mitigation of Other Attacks

7 Appendix A: CSPs

Triple-DES Key

- Description/Usage: Used to encrypt & decrypt operator data. Key Length Supported is 192 bits (3 keys of length 64bits). Used in modes ECB, CBC, CFB.
- Generation: Supplied by caller from the unmodified output of the SP 800-90A DRBG. As per SP800-133 Section 7.1, key generation is performed as per the “Direct Generation” of Symmetric Keys which is an Approved key generation method.
- Storage: Stored in plaintext internally in structure pointed to by *PGPceSymmetricCipherContext.*
- Input: Input: Keys may be input into the module via *PGPceInitSymmetricCipher*, however, per IG 7.7 “CM Software to/from App Software via GPC INT Path” – Input is **N/A.**
- Output: **N/A.**
- Key-to-Entity: Symmetric Cipher Operations.
- Zeroization: Whenever the *PGPceSymmetricCipherContext* structure is deallocated either by the Application (*PGPceFreeObject*) or when module is powered down.

AES Key

- Description/Usage: Used to encrypt & decrypt operator data. Key lengths supported are 128, 192, and 256 bits. Used in modes ECB, CBC, CFB. Used for SP800-38F AES KeyWrap and used for AES GCM Authenticated Encryption.
- Generation: Supplied by caller from the unmodified output of the SP 800-90A DRBG. As per SP800-133 Section 7.1, key generation is performed as per the “Direct Generation” of Symmetric Keys which is an Approved key generation method.
- Storage: Stored in plaintext internally in structure pointed to by *PGPceSymmetricCipherContext.*
- Input: Input: Keys may be input into the module via *PGPceInitSymmetricCipher*, however, per IG 7.7 “CM Software to/from App Software via GPC INT Path” – Input is **N/A.**
- Output: **N/A**.
- Key-to-Entity: Symmetric Cipher Operations; Authenticated Encryption; Key Wrapping.
- Zeroization: Whenever the *PGPceSymmetricCipherContext* structure is deallocated either by the application (*PGPceFreeObject*) or when module is powered down.

AES GCM IV

- Description/Usage: Used for GCM Authenticated Encryption. Size is fixed to 96 bits.
- Generation: The resulting generated IV are from the unmodified output of the SP 800-90A DRBG.
- Storage: Stored in plaintext internally in structure pointed to by *PGPceGCMContext.*
- Input: No Entry for encryption; Input as parameter to *PGPceInitGCM* API call for decryption. However, per IG 7.7 “CM Software to/from App Software via GPC INT Path” – Input is **N/A.**
- Output: Output: Per IG 7.7 “CM Software to/from App Software via GPC INT Path” – Output is **N/A**.
- Key-to-Entity: Authenticated Encryption.
- Zeroization: Whenever the *PGPceGCMContext* structure is deallocated either by the application (*PGPceFreeObject*) or when module is powered down.

RSA private key

- Description/Usage: Used to sign operator data. Key lengths supported are 2048, 3072, 4096. Signature modes supported are PKCS 1.5 and PSS. Used to decrypt data per SP800-56B KTS with mode KTS-OAEP-basic.
- Generation: The resulting generated seed for asymmetric key generation is from the unmodified output of the SP 800-90A DRBG. Generated using *PGPceGeneratePublicKey* API call. As per SP800-133 Sections 6.1 and 6.2, key generation is performed as per FIPS 186-4 and SP800-56B which are Approved key generation methods.
- Storage: Stored in plaintext internally in structure pointed to by *PGPcePublicKeyContext.*
- Input: Keys may be input into the module via *PGPceNewPublicKeyContext*, however, per IG 7.7 "CM Software to/from App Software via GPC INT Path" – Input is **N/A**.
- Output: The key may be exported via API, however, per IG 7.7 "CM Software to/from App Software via GPC INT Path" – Output is **N/A.**
- Key-to-Entity: Digital Signature; RSA Encrypt/Decrypt.
- Zeroization: Whenever the *PGPcePublicKeyContext* structure is deallocated either by the application (*PGPceFreeObject*) or when module is powered down.

DSA private key

- Description/Usage: Used to sign operator data per DSA FIPS 186-4. Sizes supported are: L = 2048, N = 224; L = 2048, N = 256; L = 3072, N = 256.
- Generation: The resulting generated seed for asymmetric key generation is from the unmodified output of the SP 800-90A DRBG. Generated using *PGPceGeneratePublicKey* API call. As per SP800-133 Sections 6.1, key generation is performed as per FIPS 186-4 which is an Approved key generation method.
- Storage: Stored in plaintext internally in structure pointed to by *PGPcePublicKeyContext.*
- Input: Keys may be input into the module via *PGPceNewPublicKeyContext*, however, per IG 7.7 "CM Software to/from App Software via GPC INT Path" – Input is **N/A.**
- Output: The key may be exported via API, however, per IG 7.7 "CM Software to/from App Software via GPC INT Path" – Output is **N/A.**
- Key-to-Entity: Digital Signature.
- Zeroization: Whenever the *PGPcePublicKeyContext* structure is deallocated either by the application (*PGPceFreeObject*) or when module is powered down.

ECDSA private key

- Description/Usage: Used to sign operator data per FIPS 186-4. Curves supported are P-256, P-384, P-521.
- Generation: The resulting generated seed for asymmetric key generation is from the unmodified output of the SP 800-90A DRBG. Generated using *PGPceGeneratePublicKey* API call. As per SP800-133 Sections 6.1, key generation is performed as per FIPS 186-4 which is an Approved key generation method.
- Storage: Stored in plaintext internally in structure pointed to by *PGPcePublicKeyContext.*
- Input: Keys may be input into the module via *PGPceNewPublicKeyContext*, however, per IG 7.7 "CM Software to/from App Software via GPC INT Path" – Input is **N/A.**
- Output: The key may be exported via API, however, per IG 7.7 "CM Software to/from App Software via GPC INT Path" – Output is **N/A.**
- Key-to-Entity: Digital Signature.
- Zeroization: Whenever the *PGPcePublicKeyContext* structure is deallocated either by the application (*PGPceFreeObject*) or when module is powered down.

ECDSA secret *k* value

- Description: Used to create a digital signature per FIPS 186-4. Curves supported are P-256, P-384, P-521.
- Generation: The *k* value is generated internally from the unmodified output of the SP 800-90A DRBG. As per SP800-133 Sections 6.1, key generation is performed as per FIPS 186-4 which is an Approved key generation method.
- Storage: Stored in plaintext in working memory in a temporary data structure.
- Input: **N/A.**
- Output: **N/A.**
- Key-to-Entity: Digital Signature.
- Zeroization: The temporary data structures are cleared with *bnEnd* or when module is powered down.

ECC CDH private key

- Description/Usage: ECC CDH private key used to generate a shared secret. Curves supported are P-256, P-384, P-521.
- Generation: The resulting generated seed for asymmetric key generation is from the unmodified output of the SP 800-90A DRBG. Generated using *PGPceGeneratePublicKey* API call. As per SP800-133 Section 6.2, the random value (K) needed to generate key pairs for the elliptic curve is the output of the SP800-90 DRBG; this is Approved as per SP800-56A.
- Storage: Stored in plaintext internally in structure pointed to by *PGPcePublicKeyContext.*
- Input: Keys may be input into the module via *PGPceNewPublicKeyContext*, however, per IG 7.7 “CM Software to/from App Software via GPC INT Path” – Input is **N/A.**
- Output: The key may be exported via API, however, per IG 7.7 “CM Software to/from App Software via GPC INT Path” – Output is **N/A.**
- Key-to-Entity: Shared Secret Generation.
- Zeroization: Whenever the *PGPcePublicKeyContext* structure is deallocated either by the application (*PGPceFreeObject*) or when module is powered down.

ECC CDH shared secret

- Description/Usage: ECC CDH shared secret per SP 800-56A. Curves supported are P-256, P-384, P-521.
- Generation: Internally generated via SP 800-56A ECC CDH shared secret generation.
- Storage: Stored in plaintext in working memory in temporary data structures.
- Input: **N/A.**
- Output: The shared secret is output as derived, however, per IG 7.7 “CM Software to/from App Software via GPC INT Path” – Output is **N/A.**
- Key-to-Entity: Shared Secret Generation.
- Zeroization: The temporary data structures are cleared with *ecPointFree* or when module is powered down.

DRBG Entropy Input, Nonce, Personalization String

- Description/Usage: Used for DRBG Instantiation and Random Bit Generation. AES 256 CTR with Derivation Function. Entropy Input: 256 bits; Nonce: 128 bits; Personalization String Length: 0-256 bits.
- Generation: Entropy Input & Nonce supplied by caller from the unmodified output of the NDRNG. Personalization String is provided by the calling application.
- Storage: Stored in plaintext internally in structure pointer to by *PGPceDRBG.*
- Input: Input as parameters to *PGPceDRBGAddEntropy* API call, however, per IG 7.7 “CM Software to/from App Software via GPC INT Path” – Input is **N/A.**
- Output: N/A.
- Key-to-Entity: Random Bit Generation.
- Zeroization: Whenever the *PGPceDRBG* structure is deallocated either by the application (*PGPceFreeObject*) or when module is powered down.

DRBG Internal State

- Description/Usage: Used for DRBG Instantiation and Random Bit Generation. AES 256 CTR with Derivation Function internal state V and Key. V is 128 bits; Key is 256 bits.
- Generation: Internally generated via SP 800-90A DRBG.
- Storage: Stored in plaintext internally in structure pointer to by *PGPceDRBG.*
- Input: **N/A.**
- Output: **N/A.**
- Key-to-Entity: Random Bit Generation.
- Zeroization: Whenever the *PGPceDRBG* structure is deallocated either by the application (*PGPceFreeObject*) or when module is powered down.

HMAC key

- Description/Usage: Used for HMAC Message Authentication. Key lengths supported are 112 to 512 bits. The following SHA functions are supported: SHA1, SHA224, SHA256, SHA384, SHA512, SHA3-224, SHA3-256, SHA3-384, SHA3-512.
- Generation: Supplied by caller from the unmodified output of the SP 800-90A DRBG. As per SP800-133 Section 7.1, key generation is performed as per the “Direct Generation” of Symmetric Keys which is an Approved key generation method.
- Storage: Stored in plaintext internally in structure pointer to by *PGPceHMACContext.*
- Input: Keys may be input into the module via *PGPceNewHMACContext*, however, per IG 7.7 “CM Software to/from App Software via GPC INT Path” – Input is **N/A.**
- Output: **N/A.**
- Key-to-Entity: Message Authentication.
- Zeroization: Whenever the *PGPceNewHMACContext* structure is deallocated either by the application (*PGPceFreeObject*) or when module is powered down.

PBKDF Internal State

- Description: Used for SP800-132 PBKDF Key Derivation. The following HMAC SHA functions are supported: HMAC-SHA-1, HMAC-SHA-256, HMAC-SHA-384, HMAC-SHA-512.
- Generation: Internally generated via SP800-132 PBKDF Key Derivation algorithm.
- Storage: Stored in plaintext in working memory in structure pointer to by *PGPceHMACContext*.
- Input: **N/A.**
- Output: **N/A.**
- Key-to-Entity: Key Derivation.
- Zeroization: Whenever the *PGPceNewHMACContext* structure is deallocated either by the application via *PGPceFreeObject* or when module is powered down.

PBKDF Password

- Description: Used for SP800-132 PBKDF Key Derivation. The minimum password length input shall be 14 characters.
- Generation: The password is supplied by the calling application.
- Storage: Stored in plaintext in working memory in structure pointer to by *PGPceHMACContext*.
- Input: Input as parameter to *PGPceDeriveKey* API call, however, per IG 7.7 “CM Software to/from App Software via GPC INT Path” – Input is **N/A.**
- Output: **N/A.**
- Key-to-Entity: Key Derivation.
- Zeroization: Whenever the *PGPceNewHMACContext* structure is deallocated either by the application via *PGPceFreeObject* or when module is powered down.

PBKDF MK

- Description: Output of the SP800-132 PBKDF Key Derivation; Minimum: 112 bits; Maximum: (2^{32}-1) x hash length. The following HMAC SHA functions are supported: HMAC-SHA-1, HMAC-SHA-256, HMAC-SHA-384, HMAC-SHA-512.
- Generation: Internally generated via SP800-132 PBKDF Key Derivation algorithm. As per SP800-133 Section 7.5, SP800-132 PBKDF is an approved method for deriving keys from passwords for storage applications only.
- Storage: Stored in plaintext in working memory in temporary data structures.
- Input: **N/A.**
- Output: The MK is output as derived, however, per IG 7.7 “CM Software to/from App Software via GPC INT Path” – Output is **N/A.**
- Key-to-Entity: Key Derivation.
- Zeroization: The temporary data structures are cleared with *pgpClearMemory* or when module is powered down.

KBKDF Internal State

- Description: Used for SP800-108 KBKDF Key Derivation. The following HMAC SHA functions are supported: HMAC-SHA-1, HMAC-SHA-256, HMAC-SHA-384, HMAC-SHA-512.
- Generation: Internally generated via SP800-108 KBKDF Key Derivation algorithm.
- Storage: Stored in plaintext in working memory in structure pointer to by *PGPceHMACContext.*
- Input: **N/A.**
- Output: **N/A.**
- Key-to-Entity: Key Derivation.
- Zeroization: Whenever the *PGPceNewHMACContext* structure is deallocated either by the application via *PGPceFreeObject* or when module is powered down.

KBKDF KDK

- Description: Used for SP800-108 KBKDF Key Derivation. Minimum of 112 bits (per the output of the DRBG).
- Generation: Supplied by caller from the unmodified output of the SP 800-90A DRBG. As per SP800-133 Section 7.1, key generation is performed as per the "Direct Generation" of Symmetric Keys which is an Approved key generation method.
- Storage: Stored in plaintext in working memory in structure pointer to by *PGPceHMACContext.*
- Input: Input as parameter to *PGPceDerive* API call, however, per IG 7.7 "CM Software to/from App Software via GPC INT Path" – Input is **N/A.**
- Output: **N/A.**
- Key-to-Entity: Key Derivation.
- Zeroization: Whenever the *PGPceNewHMACContext* structure is deallocated either by the application via *PGPceFreeObject* or when module is powered down.

KBKDF DKM

- Description: Output of the SP800-108 KBKDF Key Derivation; Minimum: 112 bits; Maximum: (2^{32}-1) x hash length. The following HMAC SHA functions are supported: SHA1, SHA256, SHA384, SHA512
- Generation: Internally generated via SP800-108 KBKDF Key Derivation algorithm. As per SP800-133 Section 7.4, SP800-108 KBKDF is an approved method for deriving keys from a key derivation key.
- Storage: Stored in plaintext in working memory in temporary data structures.
- Input: **N/A.**
- Output: DKM output as derived, however, per IG 7.7 "CM Software to/from App Software via GPC INT Path" – Output is **N/A.**
- Key-to-Entity: Key Derivation.
- Zeroization: The temporary data structures are cleared with *memset* or when module is powered down.

8 Appendix B: Public Keys

RSA public key

- Description/Usage: Used to verify signed data. Key lengths supported are 2048, 3072, 4096. Signature modes supported are PKCS 1.5 and PSS. Used to encrypt data per SP800-56B KTS with mode KTS-OAEP-basic.
- Generation: The resulting generated seed for asymmetric key generation is from the unmodified output of the SP 800-90A DRBG. Generated using *PGPceGeneratePublicKey* API call. As per SP800-133 Sections 6.1 and 6.2, key generation is performed as per FIPS 186-4 and SP800-56B which are Approved key generation methods.
- Storage: Stored in plaintext internally in structure pointed to by *PGPcePublicKeyContext.*
- Input: Keys may be input into the module via *PGPceNewPublicKeyContext*, however, per IG 7.7 “CM Software to/from App Software via GPC INT Path” – Input is **N/A.**
- Output: The key may be exported via API, however, per IG 7.7 “CM Software to/from App Software via GPC INT Path” – Output is **N/A.**
- Key-to-Entity: Digital Signature; RSA Encrypt/Decrypt.
- Zeroization: Whenever the *PGPcePublicKeyContext* structure is deallocated either by the application (*PGPceFreeObject*) or when module is powered down.

DSA public key

- Description/Usage: Used to verify signed data. Sizes supported are: L = 2048, N = 224; L = 2048, N = 256; L = 3072, N = 256.
- Generation: The resulting generated seed for asymmetric key generation is from the unmodified output of the SP 800-90A DRBG. Generated using *PGPceGeneratePublicKey* API call. As per SP800-133 Sections 6.1, key generation is performed as per FIPS 186-4 which is an Approved key generation method.
- Storage: Stored in plaintext internally in structure pointed to by *PGPcePublicKeyContext.*
- Input: Keys may be input into the module via *PGPceNewPublicKeyContext*, however, per IG 7.7 “CM Software to/from App Software via GPC INT Path” – Input is **N/A.**
- Output: The key may be exported via API, however, per IG 7.7 “CM Software to/from App Software via GPC INT Path” – Output is **N/A.**
- Key-to-Entity: Digital Signature.
- Zeroization: Whenever the *PGPcePublicKeyContext* structure is deallocated either by the application (*PGPceFreeObject*) or when module is powered down.

ECDSA public key

- Description/Usage: Used to verify signed data. Curves supported are P-256, P-384, P-521.
- Generation: The resulting generated seed for asymmetric key generation is from the unmodified output of the SP 800-90A DRBG. Generated using *PGPceGeneratePublicKey* API call. As per SP800-133 Sections 6.1, key generation is performed as per FIPS 186-4 which is an Approved key generation method.
- Storage: Stored in plaintext internally in structure pointed to by *PGPcePublicKeyContext.*
- Input: Keys may be input into the module via *PGPceNewPublicKeyContext*, however, per IG 7.7 “CM Software to/from App Software via GPC INT Path” – Input is **N/A**.
- Output: The key may be exported via API, however, per IG 7.7 “CM Software to/from App Software via GPC INT Path” – Output is **N/A.**
- Key-to-Entity: Digital Signature.
- Zeroization: Whenever the *PGPcePublicKeyContext* structure is deallocated either by the application (*PGPceFreeObject*) or when module is powered down.

ECC CDH public key

- Description/Usage: ECC CDH public key used to generate a shared secret. Curves supported are P-256, P-384, P-521.
- Generation: The resulting generated seed for asymmetric key generation is from the unmodified output of the SP 800-90A DRBG. Generated using *PGPceGeneratePublicKey* API call. As per SP800-133 Section 6.2, the random value (K) needed to generate key pairs for the elliptic curve is the output of the SP800-90 DRBG; this is Approved as per SP800-56A.
- Storage: Stored in plaintext internally in structure pointed to by *PGPcePublicKeyContext.*
- Input: Keys may be input into the module via *PGPceNewPublicKeyContext*, however, per IG 7.7 “CM Software to/from App Software via GPC INT Path” – Input is **N/A.**
- Output: The key may be exported via API, however, per IG 7.7 “CM Software to/from App Software via GPC INT Path” – Output is **N/A.**
- Key-to-Entity: Shared Secret Generation.
- Zeroization: Whenever the *PGPcePublicKeyContext* structure is deallocated either by the application (*PGPceFreeObject*) or when module is powered down.

Module Integrity Authentication Key

- Description/Usage: P-384 ECDSA public key used to authenticate the software image of the module.
- Generation: **N/A** – Generated externally by Symantec.
- Storage: Stored in plaintext internally in module code.
- Input: **N/A.**
- Output: **N/A.**
- Key-to-Entity: Integrity Self-Test.
- Zeroization: **N/A.**