PGP Corporation

PGP Software Developer's Kit 3.5.3
Cryptographic Module
FIPS 140-2 Security Policy

Version 2.5.3

Revision Date 2/10/06

Table of Contents

1 Introduction

The PGP Software Developer's Kit Version 3.5.3 (PGP SDK) is a software 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).

This module is responsible for the cryptographic services used by PGP Corporation's line of software products and is used as a building block to provide the secure exchange of email, network communications and storage of data.

This document, the *Cryptographic Module Security Policy* (CMSP), also referred to as the *Security Policy*, specifies the security rules under which the module must operate.

2 Module Specifications

The PGP SDK, Version 3.5.3 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.

For the purpose of FIPS validation, this document will only be concerned with the core cryptographic APIs described under Section 4 of this document.

Table 1: Embodiment of the PGP SDK core cryptographic module on various OS in FIPS Mode

Operating System	Typical Pathname
Windows	%windir%/system32/PGPsdk.dll
Mac OS X	/Library/Frameworks/PGP.framework

Table 2: Various OS in non-FIPS mode

Operating System	Typical Pathname
Unix (AIX, HPUX ,Linux32/64, Solaris)	/usr/lib/libPGPsdk.so

The PGP SDK cryptographic module is accessible to client applications through an application-programming interface (API). The API functions in the crypto module are enumerated in Roles and Services Section of this document.

The PGP SDK provides a FIPS mode of operation; which is described in the *Approved Mode of Operation* section of this document.

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

The PGP SDK does not support multiple concurrent operators.

The module does not support a bypass mode.

2.1 Supported Algorithms

The PGP SDK implements the following algorithms in FIPS mode of operation.

Table 3: Algorithms supported by the PGP SDK in FIPS mode

Type	Algorithm	FIPS Status
Symmetric Key	Triple-DES (3-Key) TECB, TCBC, TCFB64	FIPS 46-3 (cert #379)
	AES (128,192,256)	FIPS-197 (cert #308)
Asymmetric Key	RSA (key wrapping, key establishment methodology provides between 112 to 128 bits for encryption strength)	FIPS 186-2 for Sign/Verify Encrypt/Decrypt Allowed for Key Wrapping (cert #97)
	DSA	FIPS 186-2 (cert #144)
	ElGamal (key wrapping, key establishment methodology provides between 112-256 bits for encryption strength)	Commercially available key establishment technique Allowed for Key Wrapping
Message Digest	SHA-1, 256,384,512	FIPS 180-2 (cert #381)
Message Authentication	HMAC SHA-1, 256, 384, 512	FIPS-198 (cert #114)
Random Number Generation	ANSI X9.31	X9.31 (cert #131)
Split Knowledge	Shamir Threshold Secret Sharing	Allowed

In non-FIPS mode, the PGP SDK cryptographic module also provides the following non-FIPS Approved algorithms:

- Symmetric Key Encryption: CAST-5, IDEA, Two-Fish, Blow-Fish, Arc4-128
- Message Digest: MD5, HMAC-MD5, RIPEMD60

2.2 Cryptographic Boundary

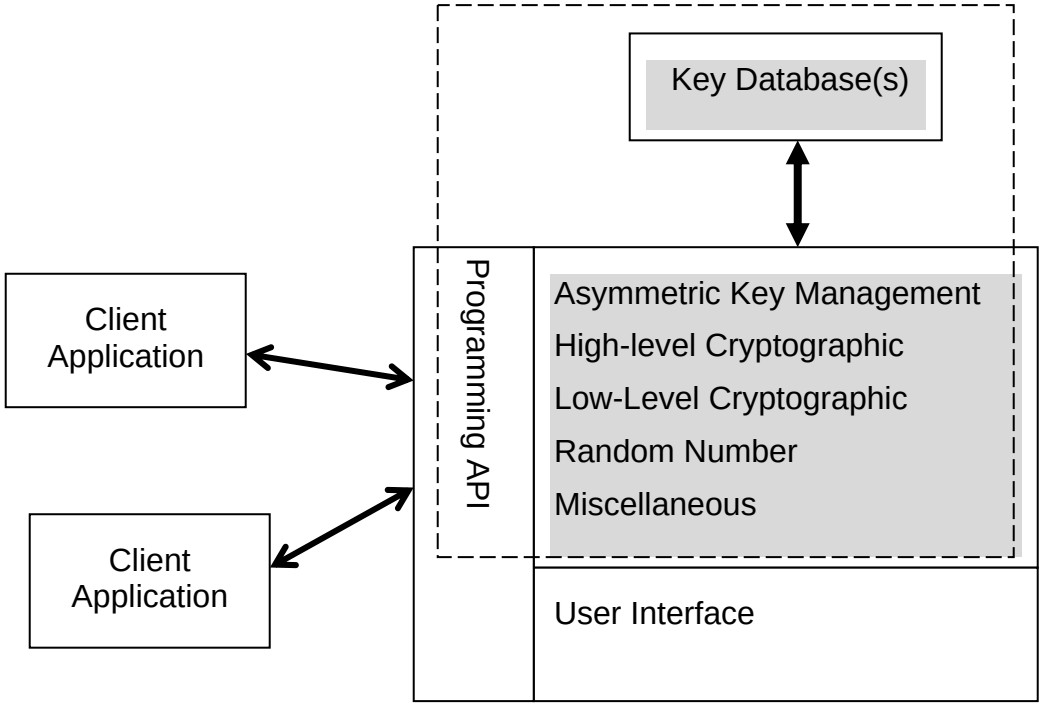
The physical cryptographic boundary is defined as the computer's case that the PGP SDK is installed in and includes all the accompanying hardware.

The module's logical cryptographic boundary is defined to be the PGP SDK binary software library as defined by 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.

In addition, the SDK's key database is included in the cryptographic boundary.

Figure 1: Module Cryptographic Boundary



Note that items within the dashed lines comprise the PGP SDK crypto boundary.

2.3 Ports and Interfaces

The PGP SDK software module restricts all access to its Critical Security Parameters (CSPs) through the API calls as enumerated in 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, hard disks, smart card interfaces, etc. provide a means to access the cryptographic module, the actual interface is via the API itself.

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

Table 4: PGP SDK Logical Ports

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

The module does not support maintenance ports. The general purpose PC used receives power via a power supply.

Input and output data can consist of plain-text, cipher-text, cryptographic keys as well as other parameters. However, there is no provision for the output of plaintext CSPs nor does the module support a cryptographic bypass mode.

All data output is prohibited whenever an error state occurs or during the self-test process.

2.4 Security Level

The PGP SDK cryptographic module meets the overall requirements applicable to Level 1 security of FIPS 140-2.

Table 5: Module Security Level Specification

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

2.5 Operational Environment

To comply with FIPS-140-2 the PGP SDK was tested using the following Operating System configuration.

- Mac OS X 10.4.2 (8C46) (ppc) running in single user mode
- Windows XP SP2 running in single user mode

The PGP SDK will also support the following Operating System Configurations

- Windows 2000 SP3
- Windows NT 4 SP6a
- Windows ME
- Windows 98SE
- Windows 98
- Mac OS 10.3.9

The operating system must be configured for single-user mode.

2.6 Approved Mode of Operation

The PGP SDK provides a FIPS 140-2 compliant mode of operation. Before any cryptographic operations can be performed, the client application must initialize (power-up) the module by invoking the `PGPEnableFIPSMode()` API.

Once the `PGPEnableFIPSMode()` call is made, the following events will occur:

1. The module will perform a series of power on self-tests as detailed in the *Self-Tests* section of this document.
2. If a self-test fails, the module will return from the `PGPEnableFIPSMode()` API call with a status or error indication.
3. If a self-test fails, the module will enter a FIPS persistent error state and no cryptographic functions will be allowed until re-initialization.
4. All data output will be prohibited whenever an error state occurs or during the self-test process.
5. Only FIPS Approved or allowed cryptographic algorithms as enumerated in the *Supported Algorithms* section of this documents can be used in FIPS mode.
6. When in FIPS mode, the module will not allow the use of zero length passphrases because the passphrase is used to associate a key with the correct entity
7. Once in FIPS mode the module will maintain that mode until the module is shut down by the `PGPsdkCleanup()` API call.
8. The module will also provide conditional tests specified in the *Self-Test* section of this document.

   The client application can, at any time, test if the module is in FIPS compliant mode by performing the `PGPGetSDKErrorState()` and `PGPGetFeatureFlags()` API call. The `PGPGetFeatureFlags( )` can be used to determine the module's status.

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

NOTE: Calling any function not listed under Section 4 causes the module to enter the non-Approved mode of operation.

3 Security Rules

The 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 SDK must be used as described in this document. Calling any function not listed under Section 4 of this document violates this security policy. For APIs that accept other functions as arguments, only those functions listed under Section 4 of this document are permitted.
2. All access to Critical Security Parameters (CSPs) must only be made through the API calls specified in Section 4 below.
3. Installation of the module is the responsibility of the Crypto-Officer.
4. The PGP SDK must be installed on a host computer where the operating system is configured for single user mode.
5. The PGP SDK 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.
6. Only FIPS Approved or allowed cryptographic algorithms as enumerated in the *Supported Algorithms* section of this document are to be used.
7. Plaintext keys and ASCII Armored keys may only be manually established; automated distribution of these types of keys is prohibited.
8. The `PGPCFBRandomWash` and `PGPWashSymmetricCipher` API functions cannot be used to generate cryptographic keys when in FIPS mode. The module inhibits data output during self-tests and error states. The data output interface is logically disconnected from the processes performing key generation and zeroization.
9. The zeroization process can be achieved as follows:
   a. Free all allocated contexts with the appropriate API function: `PGPFreePrivateKeyContext, PGPFreeSymmetricCipherContext, PGPFreeCBCContext, PGPFreeCFBContext, PGPFreeHashContext, PGPFreeHMACContext`, or `PGPFreeData`.
   b. Make calls to the `PGPWipeFile` API function and zeroize all persistently stored CSPs including ASCII armored files and all ".rnd" files.
   c. On exit, make a call to the `PGPsdkCleanup` API function

.

4 Roles and Services

As mentioned earlier the only access to the cryptographic module is through the PGP SDK API. Thus, the module operator is defined as any client application that is linked to the PGP SDK shared library.

The cryptographic module supports two roles. An operator accesses both roles while using the PGP SDK and the means of access is the same for both roles.

The roles are defined as the following:

- User: Shall be allowed to perform the Module Info Service.
- Cryptographic Officer: Shall be allowed to perform all security relevant services provided by the module.

The following table lists the services in the cryptographic module organized by service category.

Table 6: Asymmetric Key Management

Service	API	Description
Open key Database	PGPOpenKeyDBFile	Create a new key set and its underlying database based on key data archived in file(s). The key set contains all the keys in the file(s).
Free key Database	PGPFreeKeyDB PGPFreeKeySet	Release the storage for a key. The removal includes clearing any memory that held key material.
Import key(s)	PGPImport	Import key(s) that were previously exported into a new key set.
Export key(s)	PGPExport	Export key(s) from a key set into a specified file or buffer.
Manage Key Database	PGPFindKeyByKeyID PGPIncKeyDBRefCount PGPKeyDBIsMutable PGPKeyDBIsUpdated	Operations to manage and get status of a Key Database
Generate a key	PGPGenerateKey PGPGenerateSubKey	Generate a new wrapped PGP key and place it into

Table 6: Asymmetric Key Management

Service	API	Description
		a key set.
Change key passphrase	PGPChangePassphrase	Change the passphrase associated with a private key.
Update key properties	PGPAddAttributeUserID PGPAddKeyOptions PGPAddUserID PGPRevoke PGPSetKeyAxiomatic PGPCreateX509CRL	Change various values associated with a key including it current trust value, its current status (enabled, disabled, revoked), its user ID, its subkeys, and signatures.
Get key properties	PGPPassphraseIsValid PGPGetKeyDBObjDataProperty PGPGetKeyDBObjAllocatedDataProperty PGPGetKeyDBObjBooleanProperty PGPGetKeyDBObjTimeProperty PGPGetKeyDBObjNumericProperty	Obtain various values associated with a key including algorithms used, key sizes, user IDs, key Ids and fingerprints, key parameter data, and signature information.
Sign key	PGPCertifyUserID PGPCreateX509Certificate PGPCreateX509CertificateFromRequest PGPCreateSelfSignedX509Certificate	Digitally sign a particular key.
Split a binary passphrase	PGPSecretReconstructData PGPSecretShareData	Split a binary passphrase used to wrap an asymmetric key using the Shamir Threshold Secret Sharing scheme..
High Level binary passphrase split API.	PGPCombineShares PGPCopySharesFromFile PGPCopySharesToFile PGPCreateShares PGPFreeShareFile PGPFreeShares PGPGetPasskeyFromShares PGPNewShareFile PGPOpenShareFile PGPSaveShareFile	Wrapper code to facilitate Shamir Threshold Secret Sharing functions.

Table 6: Asymmetric Key Management

Service	API	Description
	PGPSplitShares	
Manage a key set	PGPCheckKeyRingSigs PGPNewKeySet PGPNewOneKeySet PGPNewOneInclusiveKeySet PGPAddKey(s) PGPDeleteKeys PGPDeleteKey PGPFreeKeySet PGPCleanSignatures	Operations to manage the references in a key set.

Table 7: High Level Cryptographic

Service	API	Description
Encrypt data	PGPEncode using PGPOEncryptToKeyDBObj	Encrypt the provided data with the provided key. This service formats the resultant cipher text based on the OpenPGP or S/MIME Message format.
Sign data	PGPEncode using PGPOSignwithKey	Digitally sign the provided data with the provided key. This service formats the resultant signature based on the OpenPGP or S/MIME Message format.
Decrypt data	PGPDecode	Decrypt the provided data with the provided key. This service assumes the data provided is formatted based on the OpenPGP or S/MIME Message format.
Validate signed data	PGPDecode	Verify the digital signature on the provided data using the provided key. This service assumes the data provided is formatted based on the OpenPGP or S/MIME Message format.
Encrypt to a Symmetric Key	PGPEncode using PGPOConventionalEncrypt	Encrypt the provided data with using the provided symmetric key. This service formats the resultant cipher text based on the OpenPGP or S/MIME Message format.

Table 8: Low Level Cryptographic

Service	API	Description
Hash data	PGPContinueHash PGPCopyHashContext PGPFinalizeHash PGPFreeHashContext PGPNewHashContext PGPResetHash PGPGetHashSize	Create a hash value based on the provided data.
Compute HMAC on data	PGPContinueHMAC PGPFinalizeHMAC PGPFreeHMACContext PGPNewHMACContext PGPResetHMAC	Compute the message authentication code on the provided data using the provided key.
Encrypt data via symmetric cipher	PGPCBCEncrypt PGPCFBEncrypt PGPSymmetricCipherEncrypt PGPCBCGetSymmetricCipher PGPCFBGetSymmetricCipher PGPCFBSync PGPCopyCBCContext PGPCopyCFBContext PGPCopySymmetricCipherContext PGPFreeCBCContext PGPFreeCFBContext PGPFreeSymmetricCipherContext PGPGetSymmetricCipherSizes PGPInitCBC PGPInitCFB PGPInitSymmetricCipher PGPNewCBCContext PGPNewCFBContext PGPNewSymmetricCipherContext PGPWashSymmetricCipher PGPWipeSymmetricCipher PGPSymmetricCipherRollback PGPImportTARCacheObj PGPOpenTarCacheFile PGPFreeTarCache	Encrypt the provided data with the provided key using a symmetric cipher algorithm.
Decrypt data via symmetric	PGPCBCDecrypt PGPCFBDecrypt	Decrypt the provided data with the provided key using a

Table 8: Low Level Cryptographic

Service	API	Description
cipher	PGPSymmetricCipherDecrypt PGPExportTARCacheObj	symmetric cipher algorithm.
Encrypt with public key *	PGPPublicKeyEncrypt PGPFreePublicKeyContext PGPNewPublicKeyContext	Encrypt the provided data with the public portion of a public/private key pair.
Verify signature with public key	PGPPublicKeyVerifyRaw PGPPublicKeyVerifySignature	Verify the digital signature on the provided data using the provided key.
Decrypt with private key *	PGPPrivateKeyDecrypt PGPFreePrivateKeyContext PGPNewPrivateKeyContext	Decrypt the provided data with the private portion of a public/private key pair.
Create signature w private key	PGPPrivateKeySign PGPPrivateKeySignRaw	Digitally sign the provided data with the provided key.

* Low-level Public Key Crypto functions can only be used for key wrapping.

Table 9: Random Number Services

Service	API	Description
Get random bytes	PGPCFBGetRandom PGPContextGetRandomBytes	Obtain random data.
Get random pool properties	PGPGlobalRandomPoolHasIntelRNG PGPGlobalRandomPoolGetEntropy PGPGlobalRandomPoolGetSize PGPGlobalRandomPoolGetMinimumEntropy PGPGlobalRandomPoolHasMinimumEntropy	Obtain information about the random pool.
Update random pool	PGPCFBRandomCycle PGPCFBRandomWash PGPGlobalRandomPoolAddKeystroke PGPGlobalRandomPoolAddSystemState PGPGlobalRandomPoolMouseMoved	Update the data in the random pool.

Table 10: Miscellaneous

Service	API	Description
Initialize SDK	PGPsdkInit	Initialize the library for use.
Cleanup SDK	PGPsdkCleanup	Cleanup the library after use.
Create context	PGPNewContext PGPNewContextCustom	Create a context for a particular use of the library.
Free context	PGPFreeContext	Destroy a context from a particular use of the library (zeroize the data from RAM).
Data storage management	PGPFreeData PGPNewData PGPNewSecureData PGPReallocData PGPWipeFile	Create and free memory for holding various data including plaintext and passphrases. The PGPWipeFile function is used to zeroize data from the hard drive.

Table 10: Miscellaneous

Service	API	Description
Data I/O	PGPOAllocatedOutputBuffer PGPOInputBuffer PGPOInputFile PGPOOutputBuffer PGPOOutputFile PGPOInputTARCache PGPOOutputTARCache PGPOOutputDirectory PGPOPasskeyBuffer PGPOPassphrase PGPOPassphraseBuffer PGPORawPGPInput	Deal with buffers and file references for data including plaintext and passphrases.
Option list manipulation	PGPAddJobOptions PGPAppendOptionList PGPBuildOptionList PGPCopyOptionList PGPFreeOptionList PGPNewOptionList	Create, modify, and free a list of options and parameters provided to SDK services.
Module status	PGPGetFeatureFlags PGPGetSDKErrorState PGPResetSDKErrorState	Obtain or reset the current status of the PGP SDK cryptographic module.
Run self tests	PGPRunAllSDKSelfTests PGPRunSDKSelfTest	Run the required self-tests.
Enable FIPS compliant Mode	PGPEnableFIPSMode	Power-Up the module into FIPS 140-2 mode
Module Info	PGPGetVersionString	Obtain module version number information.

5 Access Control Policy

In the PGP SDK, access to critical security parameters is controlled. A PGP SDK User or Cryptographic 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 Cryptographic Officer can access, how the CSPs can be accessed in the cryptographic module, and which services are used for access to the data item.

A PGP SDK operator using the module in the Cryptographic Officer role can access all of the module services, but an operator in the User role can only use a subset of those functions. More information on this can be found in the *Roles and Services* section of this document.

5.1 Critical Security Parameters

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

Definition of CSPs:

- DSA Signing Key - used to sign data with DSA.
- RSA Private Key - used to sign data with RSA (or) to decrypt RSA wrapped keys (Note: the same key may not be used for signing and decryption).
- ElGamal Private Key - used to decrypt ElGamal wrapped session keys.
- TDES Encrypting Key - used to TDES encrypt/decrypt data.
- AES Encrypting Key - used to AES encrypt/decrypt data.
- HMAC Key - used for message authentication of data.

Random Seed Pool - An internally maintained pool of data for seeding the random number functions.

Definition of Public Keys:

- RSA Public Key - used to verify RSA signatures (or) used to wrap session specific symmetric keys. (Note: the same key may not be used for signature verification and encryption)
- DSA Public Key - used to verify DSA signatures.
- ElGamal Public Key - used to wrap session specific symmetric keys.

5.2 Accesses

The types of access to CSPs and Public Keys in the PGP SDK are listed in the

following table.

Table 11: CSP Access Types

Access	Description
create	The item is created
destroy	The item is destroyed, in other words the data is cleared 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

5.3 Service to CSP and Public Key Access Relationship

The following table shows which CSPs and Public Keys are accessed by each service, the role(s) the operator must be in for access, and how the CSP or Public Key is accessed on behalf of the operator when the service is performed.

Several services provided by the PGP SDK do not access any CSPs and are included here for completeness.

Table 12: Module Services vs. CSP and Public Key Access vs. Role Access

Service	CO	User	CSP or Public Key	create	destroy	read	write
Open key Database	X		DSA Signing Key RSA Private Key ElGamal Private Key RSA Public Key DSA Public Key ElGamal Public Key			●	
Free Key Database	X		DSA Signing Key RSA Private Key ElGamal Private Key RSA Public Key DSA Public Key ElGamal Public Key		●		
Import key(s)	X		DSA Signing Key RSA Private Key ElGamal Private Key RSA Public Key DSA Public Key ElGamal Public Key				●

Table 12: Module Services vs. CSP and Public Key Access vs. Role Access

Service	CO	User	CSP or Public Key	create	destroy	read	write
Export key(s)	X		DSA Signing Key RSA Private Key ElGamal Private Key RSA Public Key DSA Public Key ElGamal Public Key			●	
Manage Key Database	X		DSA Signing Key RSA Private Key ElGamal Private Key RSA Public Key DSA Public Key ElGamal Public Key				●

Table 12: Module Services vs. CSP and Public Key Access vs. Role Access

Service	CO	User	CSP or Public Key	create	destroy	read	write
Generate a key	X		DSA Signing Key RSA Private Key ElGamal Private Key RSA Public Key DSA Public Key ElGamal Public Key Random Seed Pool	●		●	
Change key passphrase	X		DSA Signing Key RSA Private Key ElGamal Private Key RSA Public Key DSA Public Key ElGamal Public Key			●	●
Update key properties	X		DSA Signing Key RSA Private Key ElGamal Private Key RSA Public Key DSA Public Key ElGamal Public Key			●	●
Get key properties	X		DSA Signing Key RSA Private Key ElGamal Private Key RSA Public Key DSA Public Key ElGamal Public Key			●	

Table 12: Module Services vs. CSP and Public Key Access vs. Role Access

Service	CO	User	CSP or Public Key	create	destroy	read	write
Sign key	X		DSA Signing Key RSA Private Key ElGamal Private Key RSA Public Key DSA Public Key ElGamal Public Key	●		●	
Split a Binary Passphrase	X		N/A				
Manage a key set	X		N/A				
Encrypt data (Note: RSA and ElGamal are used to wrap the TDES and AES keys)	X		RSA Public Key ElGamal Public Key TDES Encrypt Key AES Encrypt Key	●		●	
Sign data	X		DSA Signing Key RSA Private Key ElGamal Private Key			●	
Decrypt data (Note: RSA and ElGamal are used to unwrap the TDES and AES keys)	X		RSA Private Key ElGamal Private Key TDES Encrypt Key AES Encrypt Key			●	
Validate signed data	X		RSA Public Key DSA Public Key			●	
Hash data	X		N/A				
Compute HMAC on data	X		HMAC Key	●	●	●	●
Encrypt data via symmetric cipher	X		TDES Encrypt Key	●	●	●	●

Table 12: Module Services vs. CSP and Public Key Access vs. Role Access

Service	CO	User	CSP or Public Key	create	destroy	read	write
			AES Encrypt Key				
Decrypt data via symmetric cipher	X		TDES Encrypt Key AES Encrypt Key	●	●	●	●
Encrypt with public key (Note: RSA and ElGamal are used to wrap keys)	X		RSA Public Key ElGamal Public Key	●	●	●	●
Verify signature with public key	X		RSA Public Key DSA Public Key	●	●	●	●
Decrypt with (Note: RSA and ElGamal are used to unwrap keys)	X		RSA Private Key ElGamal Private Key	●	●	●	●
Create signature with private key	X		RSA Private Key ElGamal Private Key	●	●	●	●
Get random bytes	X		Random Seed Pool			●	
Get random pool properties	X		Random Seed Pool			●	
Update random pool	X		Random Seed Pool				●
Initialize SDK	X		Random Seed Pool	●			
Cleanup SDK	X		N/A				

Table 12: Module Services vs. CSP and Public Key Access vs. Role Access

Service	CO	User	CSP or Public Key	create	destroy	read	write
Create context	X		N/A				
Free context	X		N/A				
Data storage management	X		N/A				
Data I/O	X		N/A				
Option list manipulation	X		N/A				
Module Status	X		N/A				
Run self-tests	X		N/A				
Enable FIPS compliant Mode	X		N/A				
Module Info		X	N/A				

6 Cryptographic Key Management

The PGP SDK takes a number of steps to protect secret keys and CSPs from unauthorized disclosure, modification, and substitution throughout the key life cycle.

6.1 Key Generation and Establishment

The PGP SDK only supports the generation and establishment of wrapped key pairs in OpenPGP or X.509 certificate formats.

- When in FIPS mode, the module performs key generation compliant with ANSI X9.31
- The generation of wrapped key pairs is an atomic operation to a module API client, and intermediate key values are never output nor are they ever accessible from the API.
- The PGP SDK key export and database services can be used to distribute the public keys as appropriate via manual processes.
- The module uniquely identifies key certificates by using the opaque data reference, `PGPKeyID`.

When establishing a symmetric key, an appropriate size public key and hash algorithm must be used.

Table 13a: Guidelines for equivalent strengths of cryptographic algorithms

Security (bits)	Symmetric encryption algorithm	Minimum size of Public Keys (bits) used in RSA
112	Triple-DES	2048
128	AES-128	3072
128	AES-128	4096

Table 13b: Guidelines for equivalent strengths of cryptographic algorithms

Security (bits)	Symmetric encryption algorithm	Minimum size of Public Keys (bits) used in ElGamal
112	Triple-DES	2048
128	AES-128	3072

192	AES-128	7680
256	AES-256	15360

7 Physical Security Policy

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

8 Self-Tests

The PGP SDK provides for three forms of self-tests: power-on, on-demand and conditional. Since the PGP SDK module provides a special FIPS Approved mode of operation, the module is defined as powering-up into FIPS mode when the `PGPEnableFIPSMode()` call is made by the client application.

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 a deliberate call to `PGPResetSDKErrorState()` or by powering down the module. Once in an error state, all further cryptographic operations and data output is disabled.

The resultant test status will be also returned by the `PGPEnableFIPSMode()` call. A client application can also ascertain module status at anytime by using the `PGPGetSDKErrorState()` function. Possible error codes return by the self-test routines include:

- `kPGPError_NoErr` – Self-test was successful.
- `kPGPError_SelfTestFailed` – Self Test Failed.

8.1 Power-Up Tests

When the client application invokes a `PGPEnableFIPSMode()` API call, the module will start the self-test process. The following self-tests will run and in the following order until all the tests have been completed or one of the tests fail.

- Triple-DES – Runs known answer test of a 64-byte block in ECB, CBC and CFB modes, then decrypts the block checking for consistency with original plain-text.
- DSA / ElGamal – Runs a pair-wise consistency sign/verify test of a 48-byte block using a 1024 bit DSS key. For encryption functionality the module then runs a pair-wise consistency encrypt/decrypt test of a 48-byte block using a 2048 bit ElGamal key.
- AES - Runs known answer test of a 16 byte block in ECB modes for 128, 192 and 256 bit keys then decrypts the block checking for consistency with original plain-text
- RSA – Signs and verifies using a 48 byte block of known data with a 2048 bit RSA key certificate pair, checks against a known answer then performs a pair-wise consistency test. For encryption functionality, the module however only executes the pair-wise consistency test, because the SDK support for asymmetrical encryption outputs padded cipher-text e.g. Open-PGP.
- SHA-1, 256, 384, 512 – Runs a series of known answer tests using test vectors derived from the FIPS-180 document.
- HMAC/SHA-1, 256, 384, 512 – Runs a series of known answer tests
- X9.31 – PRNG Known answer test
- Software Integrity – The module contains a compiled in DSA/SHA public key certificate used for SDK integrity verification. The PGP SDK library binary is signed by PGP Corporation at ship, and a detached OpenPGP signature file is added to the distribution. The module uses this public key to test the installed module binary against the signature file.

8.2 On-Demand Tests

After the `PGPEnableFIPSMode()` call is complete, the client application can optionally initiate a specific test or all tests on demand by using the `PGPRunSDKSelfTest()` or `PGPRunAllSDKSelfTests()` 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.

8.3 Conditional Tests

When in FIPS mode, the module will perform the following conditional tests.

- X9.31 Continuous test
- NDRNG Continuous Test
- Pair-wise consistency test when new asymmetric key pairs are generated: When a signing key-pair is generated, a plain-text value is signed by private key and verified by public key. Whenever an encryption key-pair is generated, a plain-text value is encrypted with the public key, the cipher-text is checked to not be the same as the original plaintext, the cipher-text is then decrypted with the private-key and checked against original plain-text.
- Critical Function Test – self signatures of key certificates are checked when keys are established via the `PGPimport()` API function.

9 Mitigation of Other Attacks

The Mitigation of Other attacks security section of FIPS 140-2 is not applicable to the PGP SDK cryptographic module.

Glossary

API: Application Programming Interface.

Asymmetric Key: a public or private key.

Context: a reference value used to accept (or refer to) an internal PGP SDK state for an ongoing operation. Examples of contexts include “asymmetric cipher context” or “hash context.”

CSP: Critical Security Parameters

Cipher: a cryptographic algorithm used for encryption and decryption.

Client: Application or program calling into the PGP SDK API

ElGamal: ElGamal is a cryptographic algorithm used by the SDK as a commercially available key establishment process

FIPS: Federal Information Processing Standards.

FIPS 140-2: FIPS for cryptographic modules.

FIPS Mode: FIPS 140-2 compliant mode of operation for PGP SDK.

High-level cryptographic: a high-level CAPI that abstracts away the details of the cryptographic algorithms to be used.

Key Pair: a pair of public/private asymmetric keys.

Key Set: a collection of asymmetric keys.

Low-level cryptographic: a low-level CAPI that includes the intimate details for specific cryptographic algorithms.

MAC: Message Authentication Code.

NIST: National Institute of Standards and Technology.

OpenPGP Message Format: the message-exchange packet formats used by OpenPGP and all PGP products. See "OpenPGP Message Format, IETF RFC-2440.”

Option List: a list of options that indicates how processing should proceed.

PGP: Pretty Good Privacy; an application and protocol (RFC 1991) for secure e-mail and file encryption developed by Phil R. Zimmermann. Originally published as Freeware, the source code has always been available for public scrutiny. PGP uses a variety of algorithms, like IDEA*, RSA, DSA, MD5, SHA-1 for providing encryption, authentication, message integrity, and key management. PGP is based on the Web-of-Trust model and has worldwide deployment.

PGP SDK: PGP Software Developer's Kit.

Passphrase: a value used to provide key to entity association, either an ASCII passphrase (a sequence of ASCII characters) or a binary passphrase (binary data).

Private Key: the secret portion of an asymmetric key pair.

Public Key: the public portion of an asymmetric key pair.

Random Number: a number generated randomly.

Random Pool: a collection of random bytes, global to the PGP SDK.

Signature: an encrypted hash of data that provides authentication and integrity for the data.

S/MIME message format: : the message-exchange packet formats specified by IETF RFC 3852, 3370, 2633 and 2632.

Symmetric Key: key material used for symmetric ciphers. A symmetric key can be provided by the operator or created as needed by the PGP SDK.

User ID: an identifier that is associated with an asymmetric key (via a Signature) that represents the entity (e.g., user) to which the key is assigned.