Thycotic Software LLC

Thycotic HSM Module v1.2.5

Non-Proprietary FIPS 140-2 Cryptographic Module Security Policy

Version 1.9 – 21/3/2019

Non-Proprietary

Change History

Date	Version	Updated By	Change
31 Oct 2017	1.0	John Allers	First release
16 Jul 2018	1.1	Ari Johnson	-Edits to Sections 1.5, 2.4.3, 2.4.6, 6.2 -Edits to Tables 2, 5, 6, 7 -Added “Non-Proprietary” wording to Cover Page
17 Jul 2018	1.2	Ari Johnson	-Edit to Table 5 -Edit to sections 3.1, 6.1
2 Aug 2018	1.3	Ari Johnson	-Reformatting Edits -Fixed document version mismatch & History table -Spelling edits -Edits to Sections 1.5, 2.1, 2.3, 3, 3.1, 3.2, 5, 6.1, 6.2, 9 including tables
6 Aug 2018	1.4	Ari Johnson	-Added Table 7 – Non-Approved Services, section 3.3
7 Aug 2018	1.5	Ari Johnson	-Created sections 2.4.3.1 - 2.4.3.3 -Added details to section 2.4.3.2 regarding password/passphrase length guidelines and reasoning
20 Aug 2018	1.6	Ari Johnson	-General edits in formatting, grammar, and technical wording adjustments
5 Feb 2019	1.7	John Allers	-Removed hyperlinks to CAVP Certificates in Table 3 -Edits to Tables 5, 6 and 8
12 Feb 2019	1.8	Tucker Hall	-Edits to Footnote 4 and 5 -Update to copyright notification
21 Mar 2019	1.9	John Allers	-Added Allowed Algorithms Table (Table 5)

Table of Contents

1. Introduction

This non-proprietary guide defines the Security Policy for Thycotic Software's Thycotic HSM Module. This module is a cryptographic library for the .NET Framework which serves as a wrapper around the Microsoft CNG (Cryptography, Next Generation) API. The module exposes the Microsoft CNG API through a .NET API for use by other applications that require a managed .NET interface.

The Microsoft CNG API relies on the Microsoft Windows Server 2012 R2 Cryptographic Primitives Library (bcryptprimitives.dll and ncryptsslp.dll) (certificate #2357) cryptographic module. And the Cryptographic Primitives Library has a functional dependency on ci.dll (Cert #2355) and cng.sys (Cert #2356).

Operational Environment	
Hardware Platform	Operating System
Intel Core i7 with AES-NI running on an Intel Maho Bay	Microsoft Windows Server 2012 R2 (x64)

Table 1 - Module Operational Environment

1.1. List of Cryptographic Module Binary Executables

THYCOTIC.HSM.DLL – Version 1.2.5 for Windows 2012 R2 Operating Environments

1.2. Brief Module Description

THYCOTIC.HSM.DLL provides cryptographic primitive services.

1.3. Cryptographic Boundaries

1.3.1. Logical Boundary

The library for the module is THYCOTIC.HSM.DLL. The Thycotic HSM Module wraps the Microsoft CNG API. By default, and when operating in FIPS mode, it uses the Cryptographic Primitives Library (bcryptprimitives.dll and ncryptsslp.dll) for cryptographic operations and the Microsoft Software Key Storage Provider (KSP) for key storage.

The Microsoft Software KSP, and all other CNG KSPs, are outside the module's logical cryptographic boundary.

The module also allows access to any CNG KSP available to the operating system. This provides the ability for the module to manage key storage through an alternative mechanism, such as a HSM.

Figure 1 shows the logical boundary of the module in relation to the operating system and hardware.

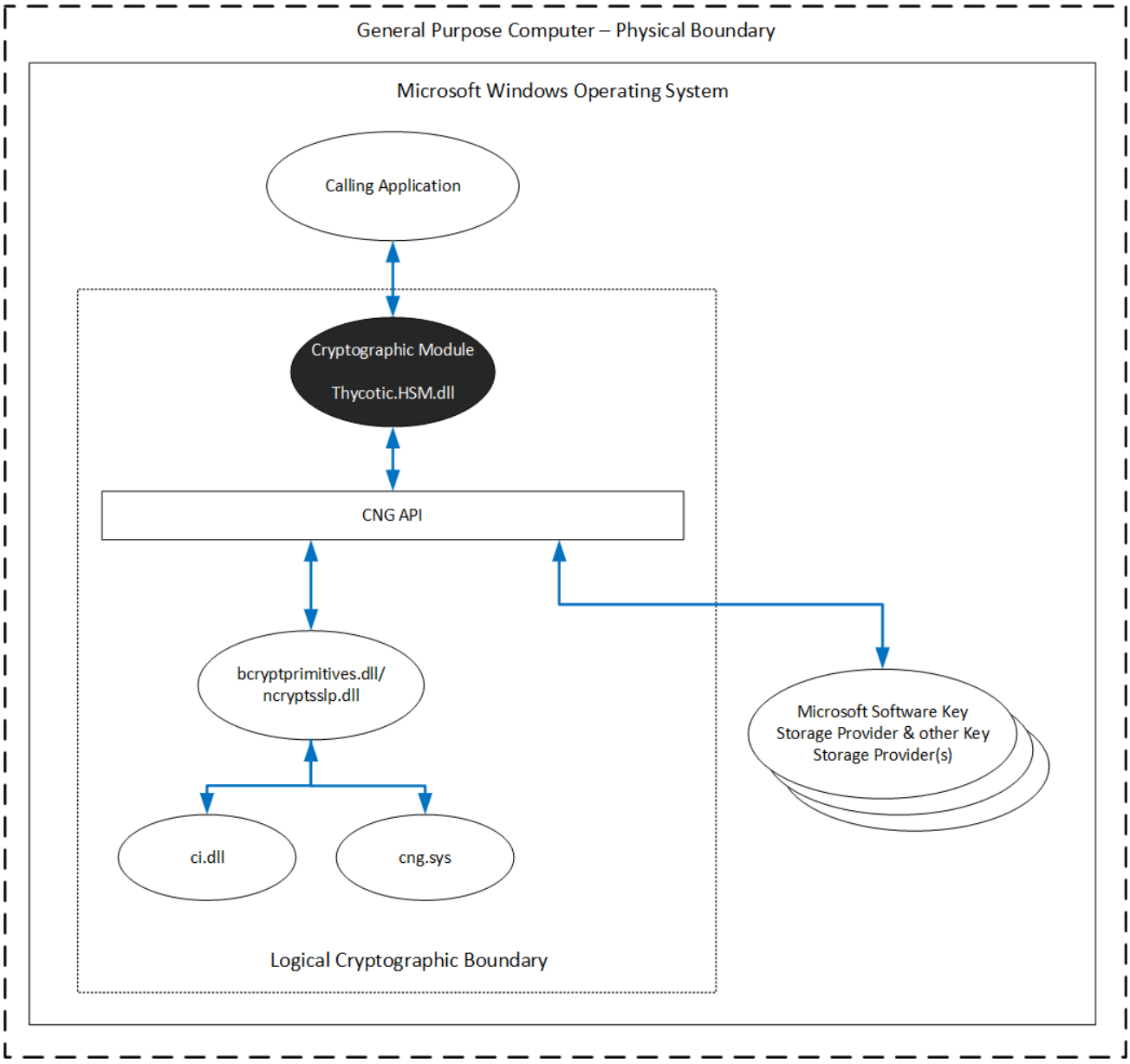

Figure 1 – Thycotic HSM Block Diagram

1.3.2. Physical Boundary

The Thycotic HSM Module runs on a General Purpose Computer (GPC) and does not include any shipped hardware components. In this case, the cryptographic boundary for the module is the computer case itself.

The module is defined as a “multi-chip standalone module” for usage under FIPS 140-2. Since the module is software only, control of the GPC ports and I/O is outside of its scope.

1.4. Ports and Interfaces

The module does not use the external physical ports of the GPC.

Interface	Module Equivalent
Data Input	API input parameters – plaintext and/or ciphertext data
Data Output	API output parameters and return values – plaintext and/or ciphertext data
Control Input	API method calls – method calls, or input parameters, that specify commands and/or control data used to control the operation of the module.
Status Output	For each API method, a CngException will be thrown when an erroneous state is encountered. The CngException includes relevant error codes. Otherwise, a successful state is assumed.

Table 2 – Ports and Interfaces

1.5. Modes of Operation

The Thycotic HSM Module can operate in either an Approved or non-Approved mode of operation. The cryptographic operator is responsible for implementing the following points to put the module into FIPS mode

The Thycotic HSM Module operates in its Approved mode of operation when the following rules are observed:

- Only approved and compliant algorithms are used during operation:
  - HMAC keys must be greater than or equal to 112 bits
  - Only RSA 2048 and 3072 are to be used in FIPS approved mode
- Cryptographic Primitives Library (bcryptprimitives.dll and ncryptsslp.dll) is installed in the operating system, validated to FIPS 140-2 under Cert. #2357.
- FIPS validated version of the Cryptographic Primitives Library dependencies are installed:
  - Code Integrity (ci.dll) validated to FIPS 140-2 under Cert. #2355.
  - Kernel Mode Cryptographic Primitives Library (cng.sys) validated to FIPS 140-2 under Cert. #2356. Used for entropy input when generating random values.
  - BitLocker Windows OS Loader (winload) validated to FIPS 140-2 under Cert. #2352.
  - Boot Manager validated to FIPS 140-2 under Cert. #2351.

- The Cryptographic Primitives Library and its dependencies are operating in FIPS mode, under the following requirements:
    - Windows is booted normally, meaning Debug mode is disabled and Driver Signing enforcement is enabled under FIPS Cert #2355.
    - After the operating system has been installed, it must be configured by enabling the "System cryptography: Use FIPS compliant algorithms for encryption, hashing, and signing" under FIPS Cert. #2351/#2357.
    - *FIPSProvider.IsFIPSComplianceEnabled* property can be used to check the DWORD registry value. Check that one of the following DWORD registry values is set to 1 in the Windows Registry:
        - HKLM\SYSTEM\CurrentControlSet\Control\Lsa\FIPSAlgorithmPolicy\Enabled
        - HKLM\SYSTEM\CurrentControlSet\Policies\Microsoft\Cryptography\Configuration\SelfTestAlgorithms

If all the above requirements are met, the module operates in a FIPS Approved mode upon initialization.

When switching between modes, all AES, RSA, and HMAC keys that were established must be zeroized as described in in Section 2.4.6 Key Zeroization.

2. Cryptographic Functionality and Configuration

The Module implements FIPS Approved as well as Non-Approved but allowed functions. If the Windows Operating system is set to enforce FIPS compliant algorithms, then the module will only allow FIPS approved algorithms.

2.1. Approved Cryptographic Algorithms

CAVP Cert	Algorithm	Standard	Description	Use
2832[1]	AES	FIPS 197, SP 800-38A	ECB (128, 192, 256) CBC (128, 192, 256) CFB128 (128, 192, 256) AES-CTR (128, 192, 256)	Data Encryption / Decryption
1487	RSA	FIPS 186-4	2048 3072	Key-Pair Generation
2373	SHS	FIPS 180-4	SHA-1 SHA-256 SHA-384 SHA-512	Message Digest

[1] CFB8, CCM, CMAC, GCM and GMAC are not used by the module.

1773	HMAC	FIPS 198-1	HMAC-SHA1 HMAC-SHA256 HMAC-SHA384 HMAC-SHA-512 Key length determined by required security strength (>= 112 bits)	Message Digest
489	DRBG	SP 800-90A	CTR_DRBG (AES-256)	Deterministic Random Bit Generation
323[2]	CVL (TLS KDF)	SP 800-135	TLS 1.2 KDF	Key Derivation
vendor affirmed	PBKDF	SP 800-132	PBKDF2-HMAC-SHA1 PBKDF2-HMAC-SHA256 PBKDF2-HMAC-SHA384 PBKDF2-HMAC-SHA512	Key Derivation

Table 3 – Approved Algorithms

2.2. Non-Approved Algorithms

The module supports the following FIPS 140-2 non-Approved algorithms, which may not be used when operating the module in a FIPS compliant manner:

Algorithm	Use
RSA Encrypt/Decrypt (2048-bit or 3072-bit key)	encrypt/decrypt

Table 4 - Non-Approved Cryptographic Algorithms

The module supports the following FIPS 140-2 non-Approved, but allowed algorithms:

Algorithm	Use
NDRNG	DRBG seed

Table 5 - Non-Approved, but Allowed Algorithms

[2] No parts of this protocol, other than the KDF, have been tested by the CAVP and CMVP. IKEv1 and IKEv2 are not used by the module.

2.3. Keys and Critical Security Parameters

Keys/CSPs	Description	Stored	Zeroization
Symmetric encryption/decryption keys	Keys used for AES encryption/decryption. Key sizes are 128, 192 and 256 bits.	RAM	Occurs when Dispose() is called on the corresponding SymmetricCngKey object.
HMAC keys	Keys used for HMAC-SHA1, HMAC-SHA256, HMAC-SHA384 and HMAC-SHA512. Key size must be greater than or equal to 112 bits.	RAM	Occurs when Dispose() is called on the corresponding HashTransform object.
RSA Private Keys[3]	Keys are generated and output to the calling application. Key sizes are 2048 and 3072 bits.	RAM/Disk/ HSM Device	Occurs when Dispose() is called on the corresponding AsymmetricCngKey object.
AES-CTR DRBG Seed[4]	A secret value maintained by the Cryptographic Primitives Library module that provides the seed material for AES-CTR DRBG output.	RAM	
AES-CTR DRBG Entropy Input[4]	A secret value maintained by the Cryptographic Primitives Library module that provides the entropy material for AES-CTR DRBG output. The module generates cryptographic keys whose strengths are modified by available entropy.	RAM	
AES-CTR DRBG V[4]	A secret value maintained by the Cryptographic Primitives Library module that provides the entropy material for AES-CTR DRBG output.	RAM	

[3] Key storage is outside the boundary and is the responsibility of the operator/User.

AES-CTR DRBG key[4]	A secret value maintained by the Cryptographic Primitives Library module that provides the entropy material for AES-CTR DRBG output.	RAM	
PBKDF Password	Input password	RAM	Occurs when Dispose() is called on the corresponding Pbkdf2Cng object.
TLS Derived Key	Keys derived using the TLS 1.2 PRF and SHA-256.	RAM	Occurs when Dispose() is called on the corresponding SafeNCryptSecretHandle object.

Table 6 – Keys and CSPs

2.4. Key Management

2.4.1. Key Material

The module contains a single key used for the HMAC-SHA512 self-integrity test. When the module is instantiated, no other keys exist in the process. The calling application is responsible for either importing keys into the module or using the module to generate keys.

2.4.2. Key Generation

The module can generate AES, RSA, and HMAC Keys within Microsoft's Cryptographic Primitive Library.

The AesCngAlgorithm class can be used to generate AES-128, AES-192, and AES-256 bit keys.

The RsaCngAlgorithm class can be used to generate RSA 2048-bit and RSA 3072-bit keys.

The RngAlgorithm class can be used to generate HMAC keys.

The module generates cryptographic keys whose strengths are modified by available entropy.

2.4.3. Key Establishment

[4] These Keys and CSPs belong to the bound module (Microsoft bcryptprimitives.dll).

The module provides key establishment using the KeyTransition (non-compliant), Pbkdf2Cng, and TlsPrfKdf classes.

2.4.3.1 KeyTransition

The KeyTransition class provides Microsoft's FIPS non-approved RSA encrypt/decrypt to encrypt AES keys. This establishment methodology provides 112-bits of encryption strength when using RSA 2048-bit keys. RSA 3072-bit keys provide 128-bits of encryption strength.

2.4.3.2 Pbkdf2Cng

Pbkdf2Cng provides functionality for a FIPS approved Password Based Key Derivation Function as specified in SP 800-132 (Section 5.3) by deriving a key from a hash value using the PBKDF2 key derivation algorithm as defined by RFC 2898. There are two (2) options presented in NIST SP 800 - 132, pages 8 - 10, that are used to derive the Data Protection Key (DPK) from the Master Key for password-based key derivation functions. Pbkdf2Cng uses option 1b.

Keys derived from passwords, as shown in SP 800 - 132, may only be used in storage applications. In order to run in a FIPS approved manner, it is up to the user and application to pick strong passwords and use them only for storage applications. The password/passphrase length is enforced by the caller of the PBKDF interfaces and not the cryptographic module. In order to run in a FIPS approved manner, the password must be chosen in accordance with the guidelines in NIST SP 800 - 63 Electronic Authentication Guideline and SP 800 - 118 DRAFT Guide to Enterprise Password Management. The upper bound for the probability of having the password guessed at random is to be computed following the SP 800 - 63 and SP 800 - 118 guidelines. The decision for the minimum length of a password used for key derivation is to be based on the SP 800 - 63 and SP 800 - 118 guidelines.

2.4.3.3 TlsPrfKdf

TlsPrfKdf provides functionality for a FIPS approved TLS Pseudo-Random Key Derivation Function

2.4.4. Key Entry and Output

AES keys can be imported into or generated by the module using the CngKeyFactory. No export functionality is provided for AES keys.

HMAC key can be provided to the module using the CreateHash() method for the appropriate algorithm (Sha1HmacHashCngAlgorithm, Sha384HmacHashCngAlgorithm, Sha256HmacHashCngAlgorithm, or Sha512HmacHashCngAlgorithm). CreateHash() will return a HashTransform which manages the handle to the key.

RSA keys can be generated by the module or opened from a CNG KSP. No other functionality is provided for importing RSA keys. No export functionality is provided for RSA keys.

2.4.5. Key Storage

The module does not store AES or HMAC keys. It is the responsibility of the calling application to manage those keys.

RSA keys may be generated and stored using the CngKeyFactory.GeneratePersistedKeys() method. The keys generated by this method are stored and managed using the Microsoft Software KSP, by default, but other CNG KSPs may be substituted. CNG KSPs are outside of the module's logical cryptographic boundary, regardless of which one is used.

2.4.6. Key Zeroization

The module also provides key zeroization for AES and RSA keys using the Cryptographic Primitives Library. The module only exposes a CNG key handle, not the key itself, through either the SymmetricCngKey or AsymmetricCngKey class.

Keys in memory can be zeroized by calling SymmetricCngKey.Dispose() or AsymmetricCngKey.Dispose(), respectively.

When Dispose() is called on either class, a call is made to the BCryptDestroyKey() method in the Cryptographic Primitives Library which will zeroize the key.

For a persisted RSA key, AsymmetricCngKey.DeleteKey() must be called to zeroize the key.

HMAC keys can be zeroized by calling HashTransform.Dispose().

The SafeNCryptSecretHandle secretHandle passed to the TlsPrfKdf.DeriveKey() function can be zeroized by calling SafeNCryptSecretHandle.Dispose().

3. Roles, Authentication and Services

3.1. Assumption of Roles

The module supports two operator roles that are implicitly assumed: Crypto-Officer (CO) and User.

The CO is responsible for setting up the operational environment and ensuring that it is running in a FIPS approved mode according to Section 1.5 Modes of Operation. The Crypto-officer is also responsible for installing the module.

The User is considered to be the owner of the thread, of which there can only be a single user concurrently. The User is responsible for choosing FIPS approved algorithms when running in FIPS approved mode, otherwise the algorithm will be subject to fail.

The module does not support authentication, nor does it implement a maintenance role or bypass capability.

3.2. Services

Service Name	Description	Role
Initialize Module	The CLR will call a static constructor to run the module integrity check and self-tests on initialization.	Crypto-Officer
Show Status	Call *CryptoStatus.IsReady()* to determine if the module is ready.	User
Data Encryption[5]	AesCngAlgorithm class can be used to encrypt data.	User
Data Decryption[5]	AesCngAlgorithm class can be used to decrypt data.	User
Asymmetric Key Generation[5]	RsaCngAlgorithm class can be used to generate RSA private and public keys.	User
Symmetric Key Generation[5]	AesCngAlgorithm class can be used to generate keys for AES and HMAC operations.	User
Message Hashing[5]	Sha1HashCngAlgorithm, Sha256HashCngAlgorithm, Sha384HashCngAlgorithm, and Sha512HashCngAlgorithm classes can be used to generate a SHA-1 or SHA-2 output.	User
Keyed Message Hashing[5]	Sha1HmacHashCngAlgorithm, Sha256HmacHashCngAlgorithm, Sha384HmacHashCngAlgorithm, and Sha512HmacHashCngAlgorithm can be used to calculate data integrity codes with HMAC.	User
Password-Based Key Derivation Function[5]	Pbkdf2Cng can be used to generate a key based on a secret input and a SHA-1 or SHA-2 based message digest.	User
TLS PRF KDF Key Derivation (SP 800-135) [5]	TlsPrfKdf class can be used to derive a key using the TLS 1.2 PRF and SHA-256.	User
DRBG (SP 800-90A) output[5]	RngAlgorithm class can be used to generate random numbers.	User
Utility	AlgorithmSupport – Determines if algorithm is supported by the operating system. *FIPSProvider.IsFIPSComplianceEnabled* property – Determines if FIPS compliance mode is enabled and enforced in the operating system. Checks if one of the following registry settings are enabled: HKLM\System\CurrentControlSet\Control\Lsa\FIPSAlgorithmPolicy\Enabled HKLM\SYSTEM\CurrentControlSet\Policies\Microsoft\Cryptography\Configuration\SelfTestAlgorithms	User

Table 7 - Service description and roles

[5] These Services belong to the bound module (Microsoft bcryptprimitives.dll). All other services belong the Thycotic HSM module.

3.3. Non-Approved Services

Service Name	Description	Role
RSA Encrypt	Used to encrypt data using a RSA key.	User
RSA Decrypt	Used to decrypt data using a RSA key.	User
Encrypted Key Entry	Used to encrypt/decrypt an AES key using a RSA key.	User

Table 8 – Non-Approved Service description and roles

Critical Security Parameters modes of access:

- W = Create/Write: The module generates or writes the CSP.
- X = Execute: The module executes using the CSP.
- D = Zeroization: The module removes the CSP.

Service Name	Symmetric encryption/decryption keys	HMAC keys	RSA Private keys	RSA Public keys	AES-CTR Seed, Entropy, Key, and V	PBKDF Input Key	TLS PRF Key
Initialize Module and Perform Self-Test							
Show Status							
Data Encryption/Decryption	X						
Key and Key-Pair Generation	W,D	W,D	W,D[6]	W,D			
Message Hashing							
Keyed Message Hashing		X					
Password-Based Key Derivation Function		X,D				X	
TLS Pseudo Random Function		X,D					W,D

[6] The key storage location is determined by the CNG KSP configured by the user. By default, the module will use the Microsoft Software Key Storage Provider, which stores the key in-memory.

DRBG (SP 800-90A) Output					X		
Utility							

Table 9 - CSP Access Rights within Services

4. Physical Security Policy

The module is software only and does not have any physical security mechanisms.

5. Self-tests

The module performs self-tests automatically when it is loaded. The self-tests include a mechanism to verify the integrity of the module. This is performed by generating a HMAC-SHA-512 value of the module file. If the integrity check fails, any attempt to perform a cryptographic operation will fail with a CryptoOperationError.

The Cryptographic Primitives Library, and its supported validated libraries, perform both power-up and conditional self-tests for their cryptographic algorithms.

The power-up self-tests can be run by an operator, on-demand, by reloading the module. The operator may check to see if the module is in a failed state by calling CryptoStatus.IsErrorStatus(). Details on why the module is in a failed state may be obtained by calling CryptoStatus.GetStatusMessage().

The module does not implement its own conditional self-tests. If the integrity check or any Cryptographic Primitives Library's self-tests fails, the module will not perform any services except the Show Status service.

6. Guidance and Secure Operation

6.1. Crypto-Officer Guidance

The key storage location may also be changed to disk or a HSM device. In the case of a HSM, the user would configure the module to use a CNG KSP provided by the HSM vendor. Once configured, the key will be stored and managed by the HSM. Any access and use of the key will go through the HSM.

The output of the RSA private key falls outside of the module's logical cryptographic boundary.

The Crypto-officer is responsible for setting up the operational environment and ensuring that it is running in a FIPS approved mode according to Section 1.5 Modes of Operation. The Crypto-officer is also responsible for installing the module.

After the operating system has been installed, it must be configured by enabling the "System cryptography: Use FIPS compliant algorithms for encryption, hashing, and signing" under FIPS Cert #2351/#2357

Windows must be booted normally, meaning Debug mode is disabled and Driver Signing enforcement is enabled. FIPS Cert #2355.

6.2. User Guidance

The calling application is responsible for managing all key material that resides outside of the module (i.e. key material that will be imported into the module). The calling application shall not use any non-approved algorithms defined in Table 4.

7. Mitigation of Other Attacks

The module provides no additional mitigation of other attacks.

8. Security Levels

Security Requirement	Security 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	1
Mitigation of Other Attacks	N/A

9. Acronyms

Term	Definition
AES	Advanced Encryption Standard

API	Application Programming Interface
CAVP	Cryptographic Algorithm Validation Program
CBC	Cipher-Block Chaining mode
CFB	Cipher Feedback mode
CNG	Cryptography, Next Generation
CLR	Common Language Runtime
CSP	Critical Security Parameter
CTR	Counter-mode
DRBG	Deterministic Random Bit Generator
GPC	General Purpose Computer
FIPS	Federal Information Processing Standards
HMAC	Hash-base Message Authentication Code
HSM	Hardware Security Module
KSP	Key Storage Provider
RSA	Rivest Shamir Adleman
RSADP	RSA Decryption Primitive
RSAEP	RSA Encryption Primitive
SHA	Secure Hash Algorithm
SHS	Secure Hash Standard