Windows 7 Enhanced Cryptographic Provider (RSAENH)

Document Version 1.6 (Windows 7)

FIPS 140-2 Documentation: Security Policy

July 21, 2010

Abstract

This document specifies the security policy for Windows 7 Enhanced Cryptographic Provider (RSAENH) as described in FIPS PUB 140-2.

CONTENTS

Introduction

The Microsoft Corporation's Windows 7 Enhanced Cryptographic Provider (RSAENH) is a FIPS 140-2 Level 1 compliant, software-based, cryptographic service provider. Like other cryptographic providers that ship with Microsoft Windows 7, RSAENH encapsulates several different cryptographic algorithms in an easy-to-use cryptographic module accessible via the Microsoft CryptoAPI. Software developers can dynamically link the Microsoft RSAENH module into their applications to provide FIPS 140-2 compliant cryptographic support.

Windows 7 does not ship the previously FIPS-140-1 validated Microsoft Base Cryptographic Provider (RSABASE) anymore. There is no loss of functionality as the RSABASE functionality has always been a subset of the RSAENH functionality.

Cryptographic Boundary

The Microsoft Enhanced Cryptographic Provider (RSAENH) consists of a single dynamically-linked library (DLL) named RSAENH.DLL (Software version 6.1.7600.16385) tested on an x86 and x64 processors, which comprises the module's logical boundary. The cryptographic boundary for RSAENH is defined as the enclosure of the computer system on which the cryptographic module is to be executed. The physical configuration of the module, as defined in FIPS PUB 140-2, is Multi-Chip Standalone. It should be noted that the Data Protection API of Microsoft Windows 7 is not part of the module and should be considered to be outside the boundary.

SECURITY POLICY

RSAENH operates under several rules that encapsulate its security policy.

- RSAENH is supported on Windows 7 (in a single-user environment).
- RSAENH operates in FIPS mode of operation only when used with the FIPS approved Windows 7 Code Integrity (ci.dll) validated to FIPS 140-2 under Cert. #1327 operating in FIPS mode and Microsoft Windows 7 Kernel Mode Cryptographic Primitives Library (cng.sys) validated to FIPS 140-2 under Cert. #1328 operating in FIPS mode
- RSAENH provides no user authentication. Roles are assumed implicitly. The authentication provided by the Windows 7 operating system is not in the scope of the validation.
- RSAENH is only in its Approved mode of operation when FIPS approved security functions are used and Windows is booted normally, meaning Debug mode is disabled and Driver Signing enforcement is enabled.
- RSAENH operates in its FIPS mode of operation only when one of the following DWORD registry values is set to 1:
  - HKLM\SYSTEM\CurrentControlSet\Control\Lsa\FIPSAlgorithmPolicy\Enabled
  - HKLM\SYSTEM\CurrentControlSet\Policies\Microsoft\Cryptography\Configuration\SelfTestAlgorithms
- All the services provided by the RSAENH DLL are available to the User and Crypto-officer roles.
- Keys created within RSAENH by one user are not accessible to any other user via RSAENH.
- When operating this module under Windows 7, the following algorithms are Approved security functions and can be used in FIPS mode:
  - FIPS-approved algorithms: Triple-DES (Cert. #846), AES (Cert. #1168), SHA-1, SHA-256, SHA-384, SHA-512 (SHS Cert. #1081), HMAC SHA-1, HMAC SHA-256, HMAC SHA-384, HMAC SHA-512 (HMAC Cert. #673), RSA (Cert. #557 and #559) and SP800-90 AES-256 based counter mode DRBG (Cert. #23). RSAENH also provides the required self-tests for these FIPS-approved algorithms.
- RSAENH also implements the following non-Approved algorithm allowed in FIPS mode:
  - RSA for Key Transport (key wrapping; key establishment methodology provides between 80 and 256 bits of encryption strength)
- RSAENH supports the following non-FIPS approved algorithms: DES, RC4, RC2, MD2, MD4, and MD5[1]; these algorithms may not be used when operating the module in a FIPS compliant manner.

RSAENH was tested using the following machine configurations:

x86	Windows 7 Ultimate – HP Compaq dc7600
x64	Windows 7 Ultimate – HP Compaq dc7600

The following diagrams illustrate the master components of the module and how it relates to the rest of the Windows CryptoAPI system:

[1] Applications may not use any of these non-FIPS algorithms if they need to be FIPS compliant. To operate the module in a FIPS compliant manner, applications must only use FIPS-approved algorithms.

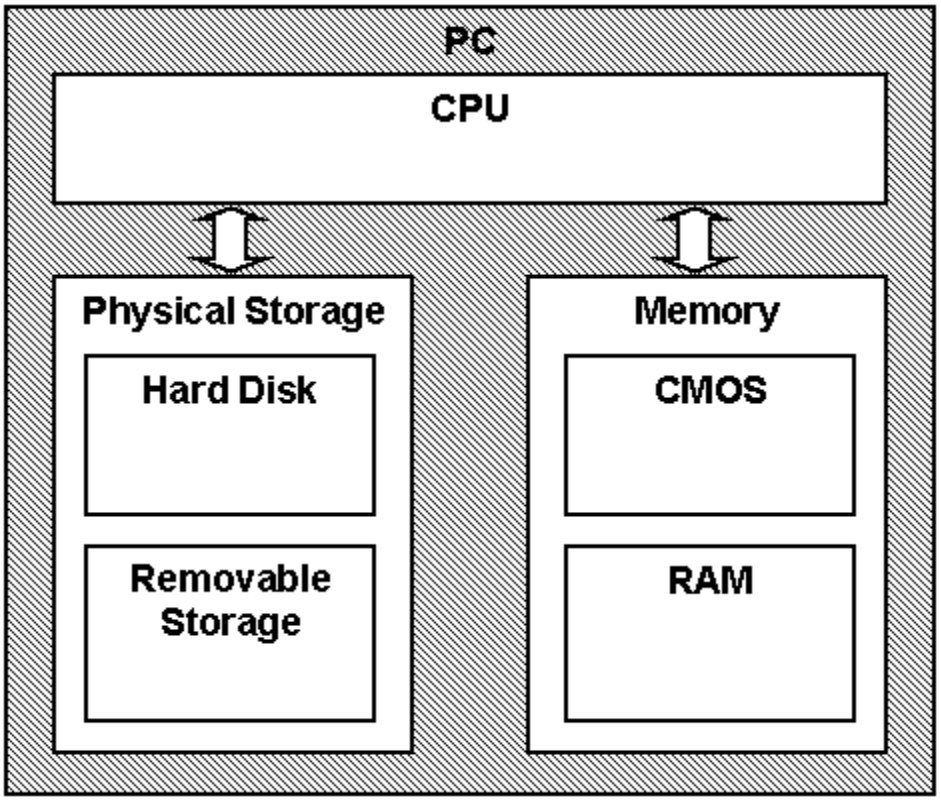

Figure 1 Master components of RSAENH module

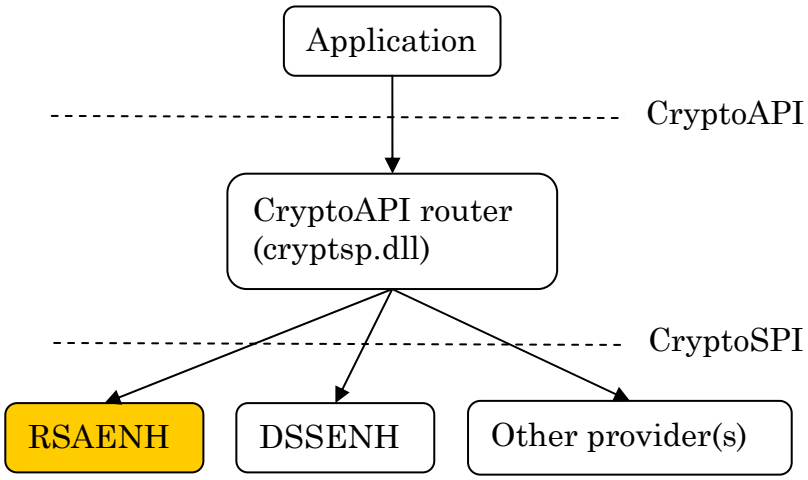

Figure 2 Relationship to other components in Windows CryptoAPI system – cryptographic module shown in gold.

SPECIFICATION OF ROLES

RSAENH module supports both User and Cryptographic Officer roles (as defined in FIPS PUB 140-2).

Both roles may access all the services implemented in the cryptographic module.

When an application requests the crypto module to generate keys for a user, the keys are generated, used, and deleted as requested by applications. There are no implicit keys associated with a user, and each user may have numerous keys, both signature and key exchange, and these keys are separate from other users' keys.

Maintenance Roles

Maintenance roles are not supported by RSAENH.

Multiple Concurrent Operators

RSAENH is intended to run on Windows 7 in Single User Mode. When run in this configuration, multiple concurrent operators are not supported.

Because the module is a DLL, each process requesting access is provided its own instance of the module. As such, each process has full access to all information and keys within the module. Note that no keys or other information are maintained upon detachment from the DLL, thus an instantiation of the module will only contain keys or information that the process has placed in the module.

Data Access

Because an operator is provided a separate instance of the module (a separate instantiation of the DLL), the operator has complete access to all of the security data items within the module.

Specification of Services

The following list contains all services available to an operator. All services are accessible by all roles.

Key Storage Services

The following functions provide interfaces to the crypto module's key container functions. Please see the Key Storage description under the Cryptographic Key Management section for more information.

CryptAcquireContext

The CryptAcquireContext function is used to acquire a programmatic context handle to a particular key container via a particular cryptographic service provider module (CSPM). This returned handle can then be used to make calls to the selected CSPM. Any subsequent calls to a cryptographic function need to reference the acquired context handle.

This function performs two operations. It first attempts to find a CSPM with the characteristics described in the *dwProvType* and *pszProvider* parameters. If the CSPM is found, the function attempts to find a key container matching the name specified by the *pszContainer* parameter.

With the appropriate setting of *dwFlags*, this function can also create and destroy key containers.

If *dwFlags* is set to CRYPT_NEWKEYSET, a new key container is created with the name specified by *pszContainer*. If *pszContainer* is NULL, a key container with the default name is created.

If *dwFlags* is set to CRYPT_DELETEKEYSET, The key container specified by pszContainer is deleted. If *pszContainer* is NULL, the key container with the default name is deleted. All key pairs in the key container are also destroyed and memory is zeroized.

When this flag is set, the value returned in *phProv* is undefined, and thus, the CryptReleaseContext function need not be called afterwards.

CryptGetProvParam

The CryptGetProvParam function retrieves data that governs the operations of the provider. This function may be used to enumerate key containers, enumerate supported algorithms, and generally determine capabilities of the CSPM.

CryptSetProvParam

The CryptSetProvParam function customizes various aspects of a provider's operations. This function is may be used to set a security descriptor on a key container.

CryptReleaseContext

The CryptReleaseContext function releases the handle referenced by the *hProv* parameter. After a provider handle has been released, it becomes invalid and cannot be used again. In addition, key and hash handles associated with that provider handle may not be used after CryptReleaseContext has been called.

Key Generation and Exchange Services

Approved Random Number Generators are used for all Key Generation. The following functions provide interfaces to the cryptomodule's key generation and exchange functions.

CryptDeriveKey

The CryptDeriveKey function creates cryptographic session keys from a hash value. This function guarantees that when the same CSPM and algorithms are used, the keys created from the same hash value are identical. The hash value is typically a cryptographic hash of a password or similar secret user data.

This function is the same as CryptGenKey, except that the generated session keys are created from the hash value instead of being random and CryptDeriveKey can only be used to create session keys. This function cannot be used to create public/private key pairs. This function can be used by a calling application as the pseudo-random function (PRF) of TLS 1.0; however, the use of this function as a standalone key derivation function is not allowed in FIPS mode.

If keys are being derived from a CALG_SCHANNEL_MASTER_HASH, then the appropriate key derivation process is used to derive the key. In this case the process used is from the SSL 2.0, SSL 3.0 or TLS specification of deriving client and server side encryption and MAC keys. This function will cause the key block to be derived from the master secret and the requested key is then derived from the key block. Which process is used is determined by which protocol is associated with the hash object. TLS must be used in FIPS mode. For more information see the SSL 2.0, SSL 3.0 and TLS specifications.

CryptDestroyKey

The CryptDestroyKey function releases the handle referenced by the *hKey* parameter. After a key handle has been released, it becomes invalid and cannot be used again.

If the handle refers to a session key, or to a public key that has been imported into the CSPM through CryptImportKey, this function zeroizes the key in memory and frees the memory that the key occupied. The underlying public/private key pair (which resides outside the crypto module) is not destroyed by this function. Only the handle is destroyed.

CryptExportKey

The CryptExportKey function exports cryptographic keys from a cryptographic service provider module (CSPM) in a secure manner for key archival purposes.

Public RSA keys are also exported using this function. A handle to the RSA public key is passed to the function and the public key is exported, always in plaintext as a blob. This blob may then be imported using the CryptImportKey function.

Symmetric keys may also be exported and wrapped with an RSA key using the CryptExportKey function. A handle to the symmetric key and a handle to the public RSA key to wrap with are passed to the function. The function returns a blob (SIMPLEBLOB) which is the wrapped symmetric key.

Symmetric keys may also be exported by encrypting the keys with another symmetric key (AES or Triple-DES). The encrypted key is then exported as a blob and may be imported using the CryptImportKey function.

CryptGenKey

The CryptGenKey function generates a random cryptographic key. A handle to the key is returned in *phKey*. This handle can then be used as needed with any CryptoAPI function requiring a key handle.

The calling application must specify the algorithm when calling this function. Because this algorithm type is kept bundled with the key, the application does not need to specify the algorithm later when the actual cryptographic operations are performed.

CryptGenRandom

The CryptGenRandom function fills a buffer with random bytes. The random number generation algorithm is the SP800-90 AES-256 based counter mode random number generation algorithm. During the function initialization, a seed is read from an in-kernel PRNG, which exists outside the cryptographic boundary. CryptGenRandom accepts caller supplied data through its in/out pbBuffer parameter. This data is mixed with the seed.

CryptGetKeyParam

The CryptGetKeyParam function retrieves data that governs the operations of a key.

CryptGetUserKey

The CryptGetUserKey function retrieves a handle of one of a user's public/private key pairs.

CryptImportKey

The CryptImportKey function transfers a cryptographic key from a key blob into a cryptographic service provider module (CSPM).

Private keys may be imported as blobs and the function will return a handle to the imported key.

A symmetric key wrapped with an RSA public key is imported into the CryptoImportKey function. The function uses the RSA private key exchange key to unwrap the blob and returns a handle to the symmetric key.

Symmetric keys encrypted with other symmetric keys (AES or Triple-DES) may also be imported using this function. The encrypted key blob is passed in along with a handle to a symmetric key, which the module is supposed to use to decrypt the blob. If the function is successful then a handle to the decrypted symmetric key is returned.

The CryptImportKey function recognizes a new flag CRYPT_IPSEC_HMAC_KEY. The flag allows the caller to supply the HMAC key material of size greater than 16 bytes. Without the CRYPT_IPSEC_HMAC_KEY flag, the CryptImportKey function would fail with NTE_BAD_DATA if the caller supplies the HMAC key material of size greater 16 bytes. For importing a HMAC key, the caller should identify the imported key blob as the PLAINTEXTKEYBLOB type and use CALG_RC2 as the key Algorithm identifier.

CryptSetKeyParam

The CryptSetKeyParam function customizes various aspects of a key's operations. This function is used to set session-specific values for symmetric keys.

CryptDuplicateKey

The CryptDuplicateKey function is used to duplicate, make a copy of, the state of a key and returns a handle to this new key. The CryptDestroyKey function must be used on both the handle to the original key and the newly duplicated key.

Data Encryption and Decryption Services

The following functions provide interfaces to the cryptomodule's data encryption and decryption functions.

CryptDecrypt

The CryptDecrypt function decrypts data previously encrypted using CryptEncrypt function.

CryptEncrypt

The CryptEncrypt function encrypts data. The algorithm used to encrypt the data is designated by the key held by the CSPM and is referenced by the hKey parameter.

Hashing and Digital Signature Services

The following functions provide interfaces to the cryptomodule's hashing and digital signature functions.

CryptCreateHash

The CryptCreateHash function initiates the hashing of a stream of data. It returns to the calling application a handle to a CSPM hash object. This handle is used in subsequent calls to CryptHashData and CryptHashSessionKey in order to hash streams of data and session keys. SHA-1 and MD5 are the cryptographic hashing algorithms supported. In addition, a MAC using a symmetric key is created with this call and may be used with any of the symmetric block ciphers support by the module (DES, Triple-DES, AES, RC4 or RC2). For creating a HMAC hash value, the caller specifies the CALG_HMAC flag in the Algid parameter, and the HMAC key using a hKey handle obtained from calling CryptImportKey.

A CALG_SCHANNEL_MASTER_HASH may be created with this call. If this is the case then a handle to one of the following types of keys must be passed in the hKey parameter, CALG_SSL2_MASTER, CALG_SSL3_MASTER, or CALG_TLS1_MASTER. This function with CALG_SCHANNEL_MASTER_HASH in the ALGID parameter will cause the derivation of the master secret from the pre-master secret associated with the passed in key handle. This key derivation process is done in the method specified in the appropriate protocol specification, SSL 2.0, SSL 3.0, or TLS. The master secret is then associated with the resulting hash handle and session keys and MAC keys may be derived from this hash handle. The master secret may not be exported or imported from the module. The key data associated with the hash handle is zeroized when CryptDestroyHash is called.

CryptDestroyHash

The CryptDestroyHash function destroys the hash object referenced by the *hHash* parameter. After a hash object has been destroyed, it can no longer be used. When a hash object is destroyed, the crypto module zeroizes the memory within the module where the hash object was held. The memory is then freed.
If the hash handle references a CALG_SCHANNEL_MASTER_HASH key then, when CryptDestroyHash is called, the associated key material is zeroized also.

All hash objects should be destroyed with the CryptDestroyHash function when the application is finished with them.

CryptGetHashParam

The CryptGetHashParam function retrieves data that governs the operations of a hash object. The actual hash value can also be retrieved by using this function.

CryptHashData

The CryptHashData function adds data to a specified hash object. This function and CryptHashSessionKey can be called multiple times to compute the hash on long data streams or discontinuous data streams. Before calling this function, the CryptCreateHash function must be called to create a handle of a hash object.

CryptHashSessionKey

The CryptHashSessionKey function computes the cryptographic hash of a key object. This function can be called multiple times with the same hash handle to compute the hash of multiple keys. Calls to CryptHashSessionKey can be interspersed with calls to CryptHashData. Before calling this function, the CryptCreateHash function must be called to create the handle of a hash object.

CryptSetHashParam

The CryptSetHashParam function customizes the operations of a hash object. For creating a HMAC hash associated with a hash object identified the hHash handle, the caller uses the CryptSetHashParam function with the HP_HMAC_INFO flag to specify the necessary SHA-1 algorithm using the CALG_SHA1 flag in the input HMAC_INFO structure. There is no need for the caller to specify the HMAC inner or outer strings as the CSPM is using the inner and outer string values as documented in the Draft FIPS for HMAC as its default values.

CryptSignHash

The CryptSignHash function signs data. Because all signature algorithms are asymmetric and thus slow, the CryptoAPI does not allow data be signed directly. Instead, data is first hashed and CryptSignHash is used to sign the hash. The crypto module supports signing with RSA. The X9.31 format may be specified by a flag.

CryptVerifySignature

The CryptVerifySignature function verifies the signature of a hash object. Before calling this function, the CryptCreateHash function must be called to create the handle of a hash object. CryptHashData or CryptHashSessionKey is then used to add data or session keys to the hash object. The crypto module supports verifying RSA signatures. The X9.31 format may be specified by a flag.

After this function has been completed, only CryptDestroyHash can be called using the hHash handle.

CryptDuplicateHash

The CryptDuplicateHash function is used to duplicate, make a copy of, the state of a hash and returns a handle to this new hash. The CryptDestroyHash function must be used on both the handle to the original hash and the newly duplicated hash.

Data Input and Output Interfaces

The Data Input Interface for RSAENH.DLL consists of the RSAENH export functions. Data and options are passed to the interface as input parameters to the RSAENH export functions. Data Input is kept separate from Control Input by passing Data Input in separate parameters from Control Input.

The Data Output Interface for RSAENH.DLL also consists of the RSAENH export functions.

Control Input Interface

The Control Input Interface for RSAENH.DLL also consists of the RSAENH export functions. Options for control operations are passed as input parameters to the RSAENH export functions.

Status Input Interface

The Status Output Interface for RSAENH.DLL also consists of the RSAENH export functions. For each function, the status information is returned to the caller as the return value from the function.

Cryptographic Key Management

The RSAENH crypto module manages keys in the following manner.

Cryptographic Keys, CSPs, and SRDIs

The RSAENH crypto module contains the following security relevant data items:

Security Relevant Data Item	**SRDI Description**
Symmetric encryption/decryption keys	Keys used for AES or TDEA encryption/decryption.
HMAC keys	Keys used for HMAC-SHA1, HMAC-SHA256, HMAC-SHA384, and HMAC-SHA512
RSA Public Keys	Keys used for the verification of RSA digital signatures or key transport.
RSA Private Keys	Keys used for the calculation of RSA digital signatures or key transport

Access Control Policy

The RSAENH crypto module allows controlled access to the SRDIs contained within it. The following table defines the access that a service has to each. The permissions are categorized as a set of four separate permissions: read (r), write (w), execute (x), delete (d). If no permission is listed, the service has no access to the SRDI.

RSAENH crypto module SRDI/Service Access Policy	Security Relevant Data Item	Symmetric encryption/decryption keys	HMAC keys	RSA Public Keys	RSA Private Keys
Service					
Key Storage Services		r / x	r / x	r / x	r / x
Key Generation and Exchange Services		r / w / d	r / w / d	r / w / d	r / w / d
Data Encryption and Decryption Services		x			
Hashing and Digital Signature Services			x	x	x

Key Material

RSAENH can create and use keys for the following algorithms: RSA Signature, RSA Key Exchange, RC2, RC4, DES, Triple-DES, and AES. Each time an application links with RSAENH, the DLL is instantiated and no keys exist within. The user application is responsible for importing keys into RSAENH or using RSAENH's functions to generate keys.

See MSDN Library\Platform SDK\Windows Base Services\Security\CryptoAPI 2.0\CryptoAPI Reference\CryptoAPI Structures\Cryptography Structures for more information about key formats and structures.
(MSDN Home > MSDN Library > Win32 and COM Development > Security > Cryptography > Cryptography Reference > General Cryptography Structures)

Key Generation

Random keys can be generated by calling the CryptGenKey() function. Triple-DES, and AES keys are generated following the techniques given in SP800-90 AES-256 based counter mode random number generation algorithm.

See MSDN Library\Platform SDK\Windows Base Services\Security\CryptoAPI 2.0\CryptoAPI Reference\CryptoAPI Functions\Base Cryptography Functions\Key Generation and Exchange Functions for more information.
(MSDN Home > MSDN Library > Win32 and COM Development > Security > Cryptography > Cryptography Reference > Cryptography Functions > Key Generation and Exchange Functions)

Key Entry and Output

Keys can be both exported and imported out of and into RSAENH via CryptExportKey() and CryptImportKey(). Exported private keys may be encrypted with a symmetric key passed into the CryptExportKey function or the keys can be exported in plaintext. Any of the symmetric algorithms supported by the crypto module may be used to encrypt private keys for export (AES, DES, Triple-DES, RC4 or RC2). When private keys are generated or imported from archival, they are covered with the Microsoft Windows 7 Data Protection API (DPAPI) and then output to the file system in the covered form.

Symmetric key entry and output is done by exchanging keys using the recipient's asymmetric public key. Symmetric key entry and output may also be done by exporting a symmetric key wrapped with another symmetric key.

Exporting the RSA private key by supplying PRIVATEKEYBLOB to the BlobType parameter of CryptExportKey() is not allowed in FIPS mode.

See MSDN Library\Platform SDK\Windows Base Services\Security\CryptoAPI 2.0\CryptoAPI Reference\CryptoAPI Functions\Base Cryptography Functions\Key Generation and Exchange Functions for more information.
(MSDN Home > MSDN Library > Win32 and COM Development > Security > Cryptography > Cryptography Reference > Cryptography Functions > Key Generation and Exchange Functions)

Key Storage

RSAENH does not provide persistent storage of keys. While, it is possible to store keys in the file system, this functionality is outside the scope of this validation. The task of protecting (or encrypting) the keys prior to storage in the file system is delegated to the Data Protection API (DPAPI) of Microsoft Windows 7. The DPAPI is a separate component of the operating system that is outside the boundaries of the cryptomodule but relies upon RSAENH for all cryptographic functionality. This section describes this functionality for information purposes only.

When a key container is deleted, the file is zeroized before being deleted. RSAENH offloads the key storage operations to the Microsoft Windows 7 operating system, which is outside the cryptographic boundary. Because keys are not persistently stored inside the cryptographic module, private keys are instead encrypted by the Microsoft Data Protection API (DPAPI) service and stored in the Microsoft Windows 7 file system. Keys are zeroized from memory after use. As an exception, the key used for power up self-testing is stored in the cryptographic module.

When an operator requests a keyed cryptographic operation from RSAENH, his/her keys are retrieved from the file system by RSAENH with the support of DPAPI.

The readers may refer to the technical paper “Windows Data Protection” (http://msdn.microsoft.com/library/en-us/dnsecure/html/windataprotection-dpapi.asp) for further detail of DPAPI.

Key Archival

RSAENH does not directly archive cryptographic keys. The operator may choose to export a cryptographic key labeled as exportable (cf. “Key Input and Output” above), but management of the secure archival of that key is the responsibility of the user.

Key Destruction

All keys are destroyed and their memory location zeroized when the operator calls CryptDestroyKey on that key handle. Private keys that reside outside the cryptographic boundary (ones stored by the operating system in encrypted format in the Windows 7 DPAPI system portion of the OS) are destroyed when the operator calls CryptAcquireContext with the CRYPT_DELETE_KEYSET flag.

Self-Tests

RSAENH provides all of the FIPS 140-2 required self-tests. As required, the module its self-tests upon power up without operator intervention and other self-tests upon encountering a specific condition (key pair or random number generation). Finally, it should be noted that non-FIPS approved algorithms should not be used if operating RSAENH in a FIPS compliant manner.

Power-up

The following FIPS-approved algorithm tests are initiated upon power-up without operator intervention:

- Triple-DES ECB encrypt/decrypt KAT
- SHA-384 KAT
- SHA-512 KAT
- SHA-1 HMAC KAT
- SHA-256 HMAC KAT
- RSA sign/verify power up test
- AES 128 ECB encrypt/decrypt KAT
- SP 800-90 CTR_DRBG KAT

Conditional

The following are initiated at key generation and random number generation respectively:

- RSA pairwise consistency test
- Continuous random number generator test

Miscellaneous

The following items address requirements not addressed above.

Cryptographic Bypass

A cryptographic bypass is not supported in RSAENH.

Operator Authentication

RSAENH provides no authentication of operators. However, the Microsoft Windows 7 operating system upon which it runs does provide authentication, but this is outside of the scope of RSAENH's FIPS validation. The information about the authentication provided by Microsoft Windows 7 is for informational purposes only. Microsoft Windows 7 requires authentication from a trusted computer base (TCB[2]) before a user is able to access system services. Once a user is authenticated from the TCB, a process is created bearing the operator's security token. All subsequent processes and threads created by that operator are implicitly assigned the parent's (thus the operator's) security token. Every user that has been authenticated by Microsoft Windows 7 is naturally assigned the operator role when he/she accesses RSAENH.

ModularExpOffload

The ModularExpOffload function offloads modular exponentiation from a CSPM to a hardware accelerator. The CSPM will check in the registry for the value HKLM\Software\Microsoft\Cryptography\ExpoOffload that can be the name of a DLL. The CSPM uses LoadLibrary to load that DLL and calls GetProcAddress to get the OffloadModExpo entry point in the DLL specified in the registry. The CSPM uses the entry point to perform all modular exponentiations for both public and private key operations. Two checks are made before a private key is offloaded. Note that to use RSAENH in a FIPS compliant manner, this function should only be used if the hardware accelerator is FIPS validated.

Operating System Security

The RSAENH cryptomodule is intended to run on Windows 7 in Single User Mode, where there is only one interactive user during a logon session.

Each operating system process creates a unique instance of the cryptomodule that is wholly dedicated to that process. The cryptomodule is not shared between processes.

Each process requesting access is provided its own instance of the module. As such, each process has full access to all information and keys within the module. Note that no keys or other information

[2] The TCB is the part of the operating system that is designed to meet the security functional requirements of the Controlled Access Protection Profile, which can be found at <http://www.radium.ncsc.mil/tpep/library/protection_profiles/index.html>. At this time, Windows 7 has not been evaluated.

are maintained upon detachment from the DLL, thus an instantiation of the module will only contain keys or information that the process has placed in the module.

For More Information

For the latest information on Windows 7, check out our World Wide Web site at http://www.microsoft.com/windows.