Security Policy

for FIPS 140-2 Validation

Microsoft Windows 8

Microsoft Windows Server 2012

Microsoft Windows RT

Microsoft Surface Windows RT

Microsoft Surface Windows 8 Pro

Microsoft Windows Phone 8

Enhanced Cryptographic Provider (RSAENH.DLL)

DOCUMENT INFORMATION

Version Number	1.1
Updated On	July 17, 2013

Table of Contents

1 Introduction

The Microsoft Corporation's Windows 8, Windows RT, Windows Server 2012, and Windows Phone 8 Enhanced Cryptographic Provider is a FIPS 140-2 Level 1 compliant, software-based, cryptographic service provider. Like other cryptographic providers that ship with Microsoft Windows 8, Windows RT, Windows Server 2012, and Windows Phone 8, Enhanced Cryptographic Provider encapsulates several different cryptographic algorithms in an easy-to-use cryptographic module accessible via the Microsoft CryptoAPI. Software developers can dynamically link the Microsoft Enhanced Cryptographic Provider module into their applications to provide FIPS 140-2 compliant cryptographic support.

1.1 List of Cryptographic Module Binary Executables

RSAENH.DLL – Version 6.2.9200 for Windows 8, Windows RT, Windows Server 2012, and Windows Phone 8

1.2 Brief Module Description

RSAENH.DLL is a dynamically-linked library (DLL) that provides cryptographic algorithms via CryptoAPI.

1.3 Validated Platforms

The Enhanced Cryptographic Provider component listed in Section 1.1 was validated using the following machine configurations:

x86 Microsoft Windows 8 Enterprise – Dell Dimension C521 (AMD Athlon 64 X2 Dual Core)
x64 Microsoft Windows 8 Enterprise – Dell PowerEdge SC430 (Intel Pentium D without AES-NI)
x64-AES-NI Microsoft Windows 8 Enterprise – Intel Client Desktop (Intel Core i7 with AES-NI)
x64 Microsoft Windows Server 2012 – Dell PowerEdge SC430 (Intel Pentium D without AES-NI)
x64-AES-NI Microsoft Windows Server 2012 – Intel Client Desktop (Intel Core i7 with AES-NI)
ARMv7 Thumb-2 Microsoft Windows RT – NVIDIA Tegra 3 Tablet (NVIDIA Tegra 3 Quad-Core)
ARMv7 Thumb-2 Microsoft Windows RT – Qualcomm Tablet (Qualcomm Snapdragon S4)
ARMv7 Thumb-2 Microsoft Windows RT – Microsoft Surface Windows RT (NVIDIA Tegra 3 Quad-Core)
x64-AES-NI Microsoft Windows 8 Pro – Microsoft Surface Windows 8 Pro (Intel x64 Processor with AES-NI)
ARMv7 Thumb-2 Microsoft Windows Phone 8 – Windows Phone 8 (Qualcomm Snapdragon S4)

The Enhanced Cryptographic Provider maintains FIPS 140-2 validation compliance (according to FIPS 140-2 PUB Implementation Guidance G.5) on the following platforms:

x86 Microsoft Windows 8
x86 Microsoft Windows 8 Pro

x64 Microsoft Windows 8
x64 Microsoft Windows 8 Pro
x64 Microsoft Windows Server 2012 Datacenter

x64-AES-NI Microsoft Windows 8
x64-AES-NI Microsoft Windows 8 Pro

x64-AES-NI Microsoft Windows Server 2012 Datacenter

1.4 Cryptographic Boundary

The software file that make up the cryptographic boundary for Enhanced Cryptographic Provider is RSAENH.DLL. The Crypto boundary is also is defined as the enclosure of the computer system, on which Enhanced Cryptographic Provider is to be executed. The physical configuration of Enhanced Cryptographic Provider, as defined in FIPS-140-2, is multi-chip standalone.

It should be noted that the Data Protection API of Microsoft Windows 8, Windows RT, Windows Server 2012, and Windows Phone 8 is not part of the module and should be considered to be outside the boundary.

2 Security Policy

Enhanced Cryptographic Provider operates under several rules that encapsulate its security policy.

- Enhanced Cryptographic Provider is supported on Windows 8, Windows RT, Windows Server 2012, and Windows Phone 8 (in a single-user environment).
- Enhanced Cryptographic Provider operates in FIPS mode of operation only when used with Windows 8, Windows RT, Windows Server 2012, and Windows Phone 8 Code Integrity (ci.dll) validated to FIPS 140-2 under Cert. #1897 for Windows 8 operating in FIPS mode, Microsoft Windows 8, Windows RT, Windows Server 2012, and Windows Phone 8 Cryptographic Primitives Library (bcryptprimitives.dll) validated to FIPS 140-2 under Cert. #1892 for Windows 8 operating in FIPS mode, and Microsoft Windows 8, Windows RT, Windows Server 2012, and Windows Phone 8 Kernel Mode Cryptographic Primitives Library (cng.sys) validated to FIPS 140-2 under Cert. #1891 for Windows 8 operating in FIPS mode.
- Enhanced Cryptographic Provider provides no user authentication. Roles are assumed implicitly. The authentication provided by the Windows 8, Windows RT, Windows Server 2012, and Windows Phone 8 operating system is not in the scope of the validation.
- Enhanced Cryptographic Provider 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.
- Enhanced Cryptographic Provider 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
- The registry security policy settings can be observed with the regedit tool to determine whether the module is in FIPS mode.
- All the services provided by the Enhanced Cryptographic Provider are available to the User and Crypto-officer roles.

- Keys created within Enhanced Cryptographic Provider by one user are not accessible to any other user via Enhanced Cryptographic Provider.

The following diagram illustrates the master components of the Enhanced Cryptographic Provider module:

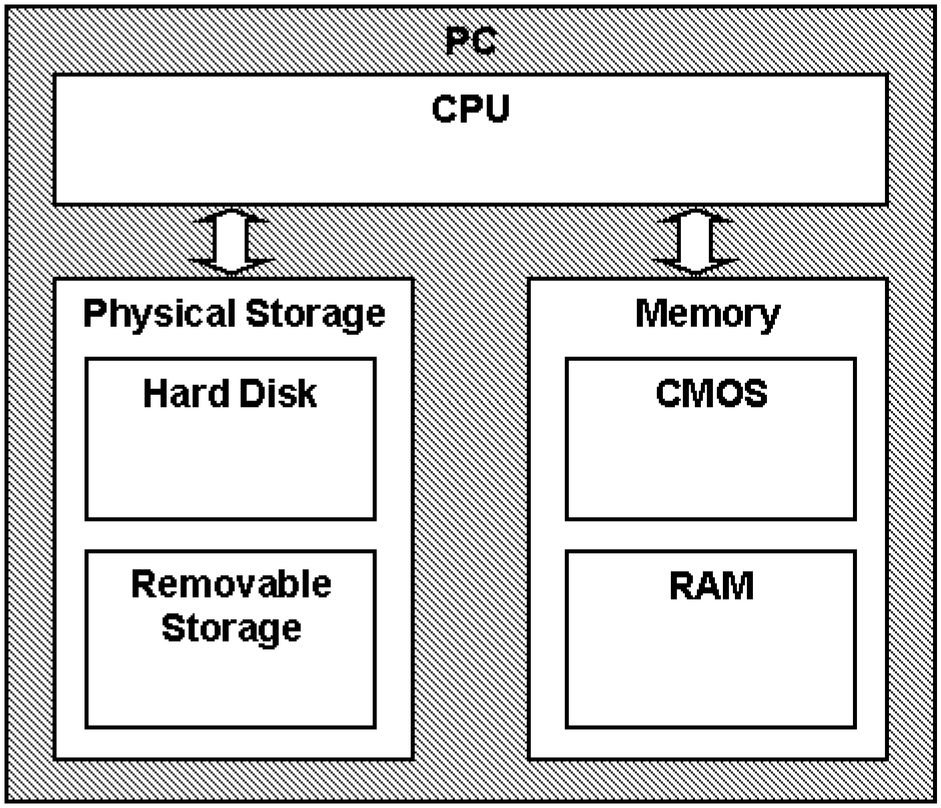

Figure 1 Master components of Enhanced Cryptographic Provider module

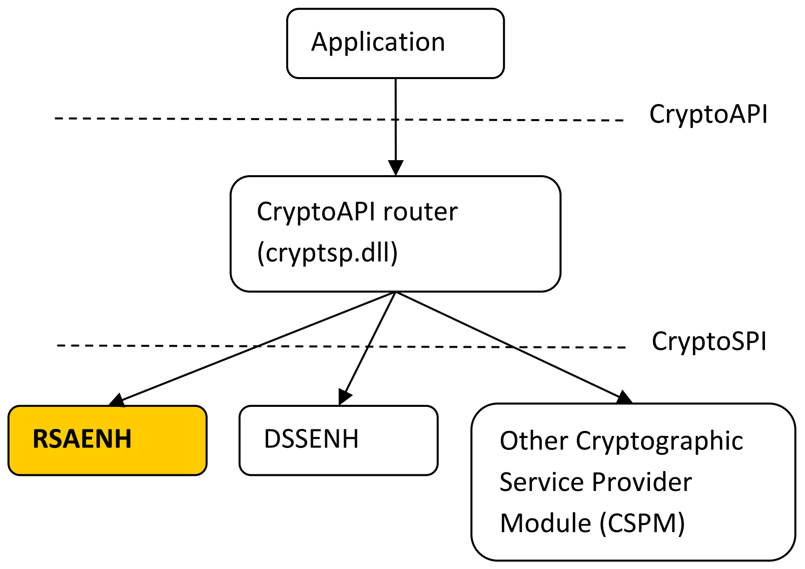
Application
CryptoAPI
CryptoAPI router (cryptsp.dll)
CryptoSPI
RSAENH
DSSENH
Other Cryptographic Service Provider Module (CSPM)

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

2.1 FIPS 140-2 Approved Algorithms

When operating this module under Windows 8, Windows RT, Windows Server 2012, and Windows Phone 8, the following algorithms are Approved security functions and can be used in FIPS mode:

- Triple-DES[1] (Cert. #1386)
- AES (Cert. #2196)
- SHA-1, SHA-256, SHA-384, SHA-512 (SHS Cert. #1902)
- HMAC SHA-1, HMAC SHA-256, HMAC SHA-384, HMAC SHA-512 (HMAC Cert. #1346)
- RSA (Cert. #1132)

2.2 Non-Approved Algorithms

Enhanced Cryptographic Provider implements the following non-Approved algorithm allowed in FIPS mode:

- RSA Key Transport (key wrapping; key establishment methodology provides between 80 and 150 bits of encryption strength)
- AES (Cert. #2196, key wrapping; key establishment methodology provides between 128 and 256 bits of encryption strength)
- Triple-DES (Cert. #1386, key wrapping; key establishment methodology provides 80 or 112 bits of encryption strength)

Enhanced Cryptographic Provider supports the following non-FIPS approved algorithms:

- DES
- RC4

[1] Two-key Triple-DES is *restricted* and *legacy-use* according to NIST SP 800-131A. Users should start transitioning away from this algorithm to better, stronger choices.

- RC2
- MD2
- MD4
- MD5

These algorithms may not be used when operating the module in a FIPS mode. To operate the module in a FIPS mode, applications must only use FIPS-approved algorithms.

2.3 Cryptographic Bypass

Cryptographic bypass is not supported by Enhanced Cryptographic Provider.

2.4 Machine Configurations

Enhanced Cryptographic Provider was tested using the machine configurations listed in Section 1.3 - Validated Platforms.

3 Operational Environment

The operational environment for Enhanced Cryptographic Provider is Windows 8, Windows RT, Windows Server 2012, and Windows Phone 8 running on the hardware listed in Section 1.3 - Validated Platforms.

The Enhanced Cryptographic Provider cryptomodule is intended to run on Windows 8, Windows RT, Windows Server 2012, and Windows Phone 8 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, and RSAENH relies on the operational environment to maintain this isolation.

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.

4 Integrity Chain of Trust

The Windows Code Integrity module (CI.DLL, certificate #1897) validates the integrity of Enhanced Cryptographic Provider before loading the latter into memory. An RSA signature with a 2048-bit key and SHA-256 message digest are used.

5 Ports and Interfaces

As depicted in Figure 2, RSAENH is a Cryptographic Service Provider Module (CSPM) and implements a set of export functions known as the CryptoSPI. These CryptoSPI functions correspond directly, in a one-to-one manner, to functions in the CryptoAPI, which is the programming interface used by Windows applications to access RSAENH and other such cryptographic providers. More information about the

CryptoSPI is available in the Windows Cryptographic Provider Development Kit, available from http://www.microsoft.com/download

5.1 Control Input Interface

The Control Input Interface for Enhanced Cryptographic Provider consists of the Enhanced Cryptographic Provider CryptoSPI export functions. Options for control operations are passed as input parameters to these functions. Data Input is kept separate from Control Input by passing Data Input in separate parameters from Control Input.

5.2 Status Output Interface

The Status Output Interface for Enhanced Cryptographic Provider consists of the CryptoSPI export functions. For each function, the status information is returned to the caller as the return value from the function.

5.3 Data Output Interface

The Data Output Interface for Enhanced Cryptographic Provider consists of the Enhanced Cryptographic Provider CryptoSPI export functions.

5.4 Data Input Interface

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

6 Specification of Roles

Enhanced Cryptographic Provider supports both User and Cryptographic Officer roles (as defined in FIPS-140-2). Both roles have access to all services implemented in Enhanced Cryptographic Provider.

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.

6.1 Maintenance Roles

Maintenance roles are not supported.

6.2 Multiple Concurrent Interactive Operators

There is only one interactive operator in Single User Mode. When run in this configuration, multiple concurrent interactive operators are not supported.

Each interactive operator may run a number of concurrent processes, and multiple such processes may access RSAENH. Each such process is provided a separate instance of the module. Each such instance will only contain keys or information that the process has placed within the module, and the process will have full access to all keys and information within its module instance.

6.3 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.

6.4 Show Status Services

The User and Cryptographic Officer roles have the same Show Status functionality, which is, for each function, the status information is returned to the caller as the return value from the function.

6.5 Self-Test Services

The User and Cryptographic Officer roles have the same Self-Test functionality, which is described in Section 10 Self-Tests.

6.6 Service Inputs / Outputs

The User and Cryptographic Officer roles have service inputs and outputs as specified in Section 5 Ports and Interfaces and Section 7 Services.

7 Services

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

Note that the functions named in this section are CryptoAPI functions; as mentioned in Section 4, these are called by the application and correspond directly to the CryptoSPI functions implemented by RSAENH.

7.1 Key Storage Services

The following functions provide interfaces to the crypto module's key container functions. Please see the Key Storage description under Section 9 - Security Relevant Data Items for more information.

7.1.1 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.

7.1.2 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.

7.1.3 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.

7.1.4 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.

7.2 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.

7.2.1 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.

7.2.2 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.

7.2.3 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.

7.2.4 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.

7.2.5 CryptGenRandom

The CryptGenRandom function fills a buffer with random bytes. This function merely forwards the call to a FIPS approved RNG from the Cryptographic Primitives Library (bcryptprimitives.dll) with DRBG (Cert. #258).

7.2.6 CryptGetKeyParam

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

7.2.7 CryptGetUserKey

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

7.2.8 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.

7.2.9 CryptSetKeyParam

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

7.2.10 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.

7.3 Data Encryption and Decryption Services

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

7.3.1 CryptDecrypt

The CryptDecrypt function decrypts data previously encrypted using CryptEncrypt function.

7.3.2 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.

7.4 Hashing and Digital Signature Services

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

7.4.1 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.

7.4.2 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.

7.4.3 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.

7.4.4 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.

7.4.5 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.

7.4.6 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.

7.4.7 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.

Note: this function accepts SHA-1 hashes, which according to NIST SP 800-131A is currently *deprecated* for digital signature generation and will be *disallowed* after the end of 2013. Similarly for RSA with keys shorter than 2048 bits. SHA-1 and RSA with keys shorter than 2048 bits are currently *legacy-use* for digital signature verification.

7.4.8 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.

Note: this function accepts SHA-1 hashes, which according to NIST SP 800-131A is currently *deprecated* for digital signature generation and will be *disallowed* after the end of 2013. Similarly for RSA with keys shorter than 2048 bits. SHA-1 and RSA with keys shorter than 2048 bits are currently *legacy-use* for digital signature verification.

7.4.9 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.

8 Authentication

The Enhanced Cryptographic Provider crypto module does not provide authentication. Roles are implicitly assumed based on the services that are executed.

9 Security Relevant Data Items

The Enhanced Cryptographic Provider crypto module uses the following security relevant data items:

Security Relevant Data Item	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

9.1 Access Control Policy

The Enhanced Cryptographic Provider crypto module allows controlled access to the security relevant data items 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 item. The User and Cryptographic Officer roles have the same access to keys so roles are not distinguished in the table.

Enhanced Cryptographic Provider crypto module Service Access Policy	Symmetric encryption/decryption keys	HMAC keys	RSA Public Keys	RSA Private Keys
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

9.2 Key Material

Enhanced Cryptographic Provider 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 Enhanced Cryptographic Provider or using RSAENH's functions to generate keys.

See the MSDN Library for more information about key formats and structures. (Dev Center - Desktop > Docs > Desktop app development documentation > Security and Identity > Cryptography API: Next Generation > CNG Reference > CNG Structures)

9.3 Key Generation

Random keys can be generated by calling the CryptGenKey() function. CryptGenKey() provides key generation for symmetric keys (such as AES and Triple-DES) and asymmetric keys (such as RSA). Random keys are generated following the standard that defines each algorithm.

See the MSDN Library. (Dev Center - Desktop > Docs > Desktop app development documentation > Security and Identity > Cryptography > Cryptography Reference > Cryptography Functions > CSP Key Generation and Exchange Functions)

9.4 Key Entry and Output

Keys can be both exported and imported out of and into Enhanced Cryptographic Provider via CryptExportKey() and CryptImportKey(). Exported private keys may be encrypted with a symmetric key passed into the CryptExportKey function. 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 encrypted with the Microsoft Windows 8, Windows RT, Windows Server 2012, and Windows Phone 8 Data Protection API (DPAPI) and then output to the file system in the encrypted 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 or keys may be output in plaintext.

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

See the MSDN Library. (Dev Center - Desktop > Docs > Desktop app development documentation > Security and Identity > Cryptography > Cryptography Reference > Cryptography Functions > CSP Key Generation and Exchange Functions)e

9.5 Key Storage

Enhanced Cryptographic Provider 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 8, Windows RT, Windows Server 2012, and Windows Phone 8. The DPAPI is a separate component of the operating system that is outside the boundaries of the cryptomodule. This section describes this functionality for information purposes only.

When a key container is deleted, the file is zeroized before being deleted. Enhanced Cryptographic Provider offloads the key storage operations to the Microsoft Windows 8, Windows RT, Windows Server 2012, and Windows Phone 8 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 8, Windows RT, Windows Server 2012, and Windows Phone 8 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 Enhanced Cryptographic Provider with the support of DPAPI.
Please refer to the technical paper "Windows Data Protection" (http://msdn.microsoft.com/en-us/library/ms995355.aspx) for further detail of DPAPI.

9.6 Key Archival

Enhanced Cryptographic Provider 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.

9.7 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 8, Windows RT, Windows Server 2012, and Windows Phone 8 DPAPI system portion of the OS) are destroyed when the operator calls CryptAcquireContext with the CRYPT_DELETE_KEYSET flag.

10 Self-Tests

Enhanced Cryptographic Provider provides all of the FIPS 140-2 required self-tests. As required, the module executes its self-tests upon power-on (startup) 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 Enhanced Cryptographic Provider in a FIPS mode.

10.1 Power-On Self-Tests

The following FIPS-approved algorithm tests are initiated upon power-on (startup) without operator intervention:

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

If the self-test fails, the module will not load and status will be returned. If the status is not STATUS_SUCCESS, then that is the indicator a self-test failed.

10.2 Conditional Self-Tests

The following self-test is initiated at key generation:

- RSA pairwise consistency test

11 Design Assurance

The secure installation, generation, and startup procedures of this cryptographic module are part of the overall Windows 8, Windows RT, and Windows Server 2012 operating system secure installation, configuration, and startup procedures. 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" policy setting followed by restarting the system. This procedure is all the crypto officer and user behavior necessary for the secure operation of this cryptographic module.

Windows Phone 8 does not use the same installation, configuration, and startup procedures as the Windows operating system on a computer, but rather, is securely installed and configured by the cellular telephone carrier.

The procedures required for maintaining security while distributing and delivering versions of a cryptographic module to authorized operators are:

1. The secure distribution method is via the physical medium for product installation delivered by Microsoft Corporation, which is a DVD in the case of Windows 8 and Windows Server 2012. In the case of Windows RT, Surface Windows RT, Surface Windows 8 Pro, and Windows Phone 8, the cryptographic module is already installed at the factory and is only distributed with the hardware.
2. An inspection of authenticity of the physical medium can be made by following the guidance at this Microsoft web site: http://www.microsoft.com/en-us/howtotell/default.aspx
3. The installed version of Windows 8, Windows RT, and Windows Server 2012 must be verified to match the version that was validated. See Appendix A for details on how to do this.

For Windows Updates, the client only accepts binaries signed by Microsoft certificates. The Windows Update client only accepts content whose SHA-2 hash matches the SHA-2 hash specified in the metadata. All metadata communication is done over a Secure Sockets Layer (SSL) port. Using SSL ensures that the client is communicating with the real server and so prevents a spoof server from sending the client harmful requests. The version and digital signature of new cryptographic module releases must be verified to match the version that was validated. See Appendix A for details on how to do this.

12 Miscellaneous

The following items address requirements not addressed above.

12.1 Operator Authentication

Enhanced Cryptographic Provider provides no authentication of operators. The Microsoft Windows 8, Windows RT, Windows Server 2012, and Windows Phone 8 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 8, Windows RT, Windows Server 2012, and Windows Phone 8 is for informational purposes only. Microsoft Windows 8, Windows RT, Windows Server 2012, and Windows Phone 8 requires authentication from a Trusted Computing Base (TCB) 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 8, Windows RT, Windows Server 2012, and Windows Phone 8 is naturally assigned the operator role when he/she accesses RSAENH.

12.2 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 Enhanced Cryptographic Provider in a FIPS compliant manner, this function should only be used if the hardware accelerator is FIPS validated.

13 Mitigation of Other Attacks

The following table lists the mitigations of other attacks for this cryptographic module:

Algorithm	Protected Against	Mitigation	Comments
SHA1	Timing Analysis Attack	Constant Time Implementation	
	Cache Attack	Memory Access pattern is independent of any confidential data	
SHA2	Timing Analysis Attack	Constant Time Implementation	
	Cache Attack	Memory Access pattern is independent of any confidential data	
AES	Timing Analysis Attack	Constant Time Implementation	
	Cache Attack	Memory Access pattern is independent of any confidential data	Protected Against Cache attacks only when used with AES NI

14 Additional Details

For the latest information on Microsoft Windows, check out the Microsoft web site at:

http://windows.microsoft.com

For more information about FIPS 140 validations of Microsoft products, please see:

http://technet.microsoft.com/en-us/library/cc750357.aspx

15 Appendix A – How to Verify Windows Versions and Digital Signatures

15.1 How to Verify Windows Versions

The installed version of Windows 8, Windows RT, and Windows Server 2012 must be verified to match the version that was validated using one of the following methods:

1. The ver command
   a. From Start, open the Search charm.
   b. In the search field type "cmd" and press the Enter key.
   c. The command window will open with a "C:\>" prompt.
   d. At the prompt, type "ver" and press the Enter key.
   e. You should see the answer "`Microsoft Windows  [Version 6.2.9200]`".
2. The systeminfo command
   a. From Start, open the Search charm.
   b. In the search field type "cmd" and press the Enter key.
   c. The command window will open with a "C:\>" prompt.
   d. At the prompt, type "systeminfo" and press the Enter key.
   e. Wait for the information to be loaded by the tool.
   f. Near the top of the output, you should see:

```
OS Name:                   Microsoft Windows 8 Enterprise
OS Version:                6.2.9200 N/A Build 9200
OS Manufacturer:           Microsoft Corporation
```

If the version number reported by the utility matches the expected output, then the installed version has been validated to be correct.

15.2 How to Verify Windows Digital Signatures

After performing a Windows Update that includes changes to a cryptographic module, the digital signature and file version of the binary executable file must be verified. This is done like so:

1. Open a new window in Windows Explorer.
2. Type “C:\Windows\” in the file path field at the top of the window.
3. Type the cryptographic module binary executable file name (for example, “CNG.SYS”) in the search field at the top right of the window, then press the Enter key.
4. The file will appear in the window.
5. Right click on the file’s icon.
6. Select Properties from the menu and the Properties window opens.
7. Select the Details tab.
8. Note the File version Property and its value, which has a number in this format: x.x.xxxx.xxxxx.
9. If the file version number matches one of the version numbers that appear at the start of this security policy document, then the version number has been verified.
10. Select the Digital Signatures tab.
11. In the Signature list, select the Microsoft Windows signer.
12. Click the Details button.
13. Under the Digital Signature Information, you should see: “This digital signature is OK.” If that condition is true then the digital signature has been verified.