Microsoft

Windows CE and Windows Mobile Operating System

Microsoft Windows CE, Windows Mobile, and Windows Embedded Handheld Enhanced Cryptographic Provider (RSAENH)

(5.00.911762, 5.01.01603, 5.04.17228, 5.05.19202, 5.05.21840, 5.2.29344)

FIPS 140-2 Documentation: Security Policy

05/06/2015 11:21 AM

Abstract

This document specifies the security policy for the Microsoft Windows CE, Windows Mobile Enhanced Cryptographic Provider and Microsoft Windows Embedded Handheld (RSAENH) as described in FIPS PUB 140-2.

CONTENTS

INTRODUCTION

Microsoft Windows CE and Windows Mobile Enhanced Cryptographic Provider 5.00.911762, 5.01.01603, 5.04.17228, 5.05.19202, 5.05.21840 and Microsoft Windows Embedded Handheld 5.2.29344 (RSAENH) is a general-purpose, software-based, cryptographic module for Windows CE, Windows Mobile and Windows Embedded Handheld respectively. Like cryptographic providers that ship with Microsoft Windows CE, Windows Mobile and Microsoft Windows Embedded Handheld, RSAENH encapsulates several different cryptographic algorithms in an easy-to-use cryptographic module accessible via the Microsoft CryptoAPI. It can be dynamically linked into applications by software developers to permit the use of general-purpose cryptography. Microsoft Windows CE and Windows Mobile Enhanced Cryptographic Provider 5.00.911762, 5.01.01603, 5.04.17228, 5.05.19202, 5.05.21840 and Microsoft Windows Embedded Handheld 5.2.29344 (RSAENH) meet the Level 1 FIPS 140-2 Validation requirements.

Cryptographic Boundary

The Microsoft Windows CE and Windows Mobile Enhanced Cryptographic Provider 5.00.911762, 5.01.01603, 5.04.17228, 5.05.19202, 5.05.21840 and Microsoft Windows Embedded Handheld 5.2.29344 (RSAENH) consists of a single dynamically-linked library (DLL) named RSAENH.DLL. 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.

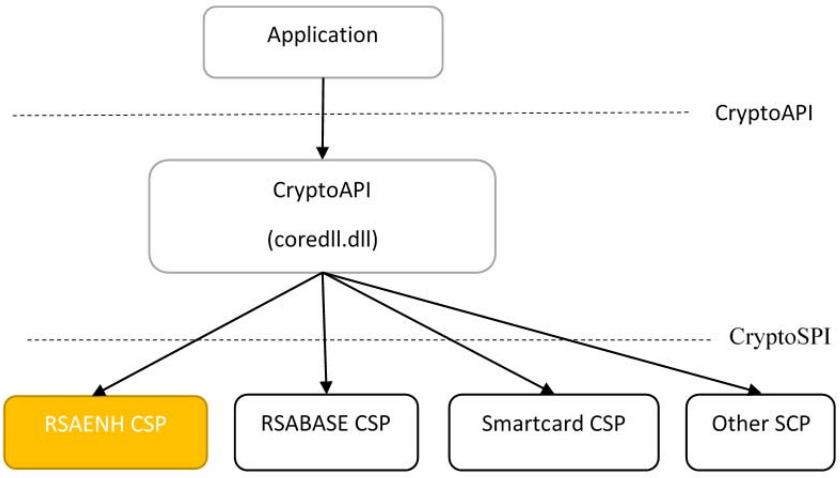
SECURITY POLICY

RSAENH operates under several rules that encapsulate its security policy.

- RSAENH is supported on Windows CE, Windows Mobile and Windows Embedded Handheld.
- Windows CE is a single user operating system, with only one interactive user context.
- All services implemented within RSAENH are available to the User and Cryptoofficer roles.

RSAENH stores RSA keys in the system registry, but relies on Microsoft Windows CE for the covering of the keys prior to storage. For FIPS purposes, these keys can be considered to be stored in PlainText, and are outside the module boundary.

The below diagram shows relationship of enhanced cryptographic provider with other components in CryptoAPI system
(cryptographic module shown in gold)



RSAENH supports the following FIPS 140-2 Approved algorithms

- RSA PKCS #1 (v1.5) / X9.31 sign and verify with private and public key
- Triple-DES keypair derivation - Derived keys cannot be used for encryption. They can be used to support authentication services only.
- Triple-DES keypair generation
- Triple-DES ECB / CBC encrypt/decrypt
- Triple-DES 112 keypair generation
- Triple-DES 112 ECB / CBC encrypt/decrypt
- AES 128 / 192 / 256 keypair derivation - Derived keys cannot be used for encryption. They can be used to support authentication services only.
- AES 128 / 192 / 256 keypair generation
  AES ECB / CBC encrypt/decrypt
- SHA-1 hash
- SHA-256, SHA-384, SHA-512
- SHA-1 based Keyed-Hash Message Authentication Code (HMAC) SHA-2 based Keyed-Hash Message Authentication Code (HMAC) i.e. HMAC-SHA-256, HMAC-SHA-384, HMAC-SHA-512.
- Approved Software Pseudo Random Number Generation (PRNG) (seeded by non-Approved PRNG) (FIPS 186-2, Appendix 3.1 and 3.3, Regular, XOriginal, SHA-1 G function, Seed-key 64 bytes only.)

The following table lists algorithms, product versions and their certificates

	Validation Certificates		
Algorithm	**5.01.01603 and 5.00.911762**	**5.04.17228, 5.05.19202, and 5.05.21840**	**5.2.29344**
AES ECB (e/d; 128 , 192 , 256); CBC (e/d; 128 , 192 , 256)	#224	#507	#3331
Triple-DES ECB(e/d; KO 1,2) ; CBC(e/d; KO 1,2)	#315	#517	#1902
FIPS186-2 RSA ALG[ANSIX9.31]: SIG(ver); 1024 , 1536 , 2048 , 3072 , 4096 ALG[RSASSA-PKCS1_V1_5]: SIG(gen) 2048 , 3072 , 4096 SIG(ver): 1024 , 1536 , 2048 , 3072 , 4096	#52	#222	#1711
FIPS186-4 RSA ALG[ANSIX9.31]: Sig(Ver): (1024 SHA(1)) (2048 SHA(1)) (3072 SHA(1)) ALG[RSASSA-PKCS1_V1_5] SIG(gen) (2048 SHA(256 , 384 , 512)) (3072 SHA(256 , 384 , 512))	N/A	N/A	#1711
RNG [(x-Original); (SHA-1)]	#66	#286	#1360
SHS **SHA-1** (BYTE-only) **SHA-256** (BYTE-only) **SHA-384** (BYTE-only) **SHA-512** (BYTE-only)	#305	#578	#2764
HMAC **HMAC-SHA1 (Key Sizes Ranges Tested:** KS<BS KS=BS KS>BS) **HMAC-SHA256 (Key Size Ranges Tested**: KS<BS KS=BS KS>BS) **HMAC-SHA384 (Key Size Ranges Tested**: KS<BS KS=BS KS>BS) **HMAC-SHA512 (Key Size Ranges Tested**: KS<BS KS=BS KS>BS)	#31	#260	#2122

- RSAENH supports the following non-Approved algorithms

  - DES ECB / CBC encrypt/decrypt
  - DES keypair derivation
  - DES keypair generation
  - DES ECB / CBC encrypt/decrypt

  - RSA keypair generation (key sizes from 384 to 16384) (the RSAENH module does not implement the Approved X9.31 algorithm for keypair generation)
  - RSA encrypt and decrypt with private and public key

  - RC2 keypair derivation (key sizes from 40 to 128)
  - RC2 keypair generation (key sizes from 40 to 128)
  - RC2 ECB / CBC encrypt/decrypt

  - RC4 keypair derivation (key sizes from 40 to 128)
  - RC4 keypair generation
  - RC4 encrypt/decrypt

  - MD2 hash
  - MD4 hash
  - MD5 hash
  - MD5 based Keyed-Hash Message Authentication Code (HMAC)

  - non-Approved Software Psuedo Random Number Generator (PRNG) (seeded by hardware data, and by application-provided data)
  - Lan Manager Hash Generation

- RSAENH performs these Power-On Self-Tests
  - AES Encrypt / Decrypt Known Answer Test
  - DES Encrypt / Decrypt Known Answer Test
  - Triple-DES Encrypt / Decrypt Known Answer Test
  - SHS (SHA -1, SHA-256, SHA-384, SHA-512) Known Answer Test
  - HMAC (SHA-1, SHA-256, SHA-386, SHA-512) Known Answer Test
  - RSA Sign / Verify using a Sign / Verify test with a Known Signature with PKCS#1 v1.5
  - RSA Sign / Verify using a Sign / Verify test with a Known Signature with X9.31
  - FIPS 186-2 PRNG Known Answer Test
  - Software Integrity Test

- RSAENH performs these Conditional Self-Tests -

  CRNGT test for Approved PRNG
  - Pair-wise consistency test for RSA key generation

PLATFORM COMPATIBILITY

RSAENH has been tested in the following configurations

1. Microsoft Windows CE 5.01 w/ x86 processor and Microsoft Windows CE 5.01 w/ ARM v4i processor
2. Microsoft Windows CE 5.00 w/ x86 processor, Microsoft Windows CE 5.00 w/
    a. ARM v4i processor and Microsoft Windows CE 5.00 w/ MIPS-IV processor
3. Microsoft Windows Mobile 6.0 w/ ARM v4i processor
4. Microsoft Windows Mobile 6.1 w/ ARM v4i processor
5. Microsoft Windows Mobile 6.5 w/ ARM v4i processor
6. Microsoft Windows Embedded Handheld 6.5 w/ ARM v4i processor

In addition, compliance is maintained on platforms for which the binary executable remains unchanged including (but not limited to):

1. Microsoft Windows Mobile 5.0 w/ x86 processor and Microsoft Windows Mobile 5.0 w/ ARM v4i processor

2. Microsoft Windows Mobile 5.0 with Messaging and Security Feature Pack w/ x86 processor and Microsoft Windows Mobile 5.0 with Messaging and Security Feature Pack w/ ARM v4i processor

PORTS AND INTERFACES

Data Input Interface

The Data Input Interface for the Microsoft Windows CE, Windows Mobile Enhanced Cryptographic Provider and Microsoft Windows Embedded Handheld (RSAENH) is a software interface, where applications invoke software functions to perform specific operations. Data and options are passed to the interface as parameters to the function. Data Input is kept separate from Control Input by passing Data Input in separate parameters from Control Input.

Data Output Interface

The Data Output Interface for the Microsoft Windows CE, Windows Mobile Enhanced Cryptographic Provider and Microsoft Windows Embedded Handheld (RSAENH) is a software interface, where applications invoke software functions to perform specific operations. Data and metadata are returned to the application in some cases as return values from the function and in other cases as output parameters from the function.

Control Input Interface

The Control Input Interface for the Microsoft Windows CE, Windows Mobile Enhanced Cryptographic Provider and Microsoft Windows Embedded Handheld (RSAENH) is a software interface, where applications invoke software functions to perform specific operations. Options for control operations are passed as parameters to the function. The specific functions in RSAENH are: CryptAcquireContext, CryptGetProvParam, CryptSetProvParam, CryptReleaseContext, CryptDeriveKey (Derived keys cannot be used for encryption. They can be used to support authentication services only.), CryptDestroyKey, CryptExportKey, CryptGenKey, CryptGenRandom, CryptGetKeyParam, CryptGetUserKey, CryptImportKey, CryptSetKeyParam, CryptDecrypt, CryptEncrypt, CryptCreateHash, CryptDestroyHash, CryptGetHashParam, CryptHashData, CryptHashSessionKey, CryptSetHashParam, CryptSignHash, CryptVerifySignature, CryptDuplicateHash, A_SHAInit, A_SHAUpdate, A_SHAFinal, BSafeComputeKeySizes, BSafeDecPrivate, BSafeEncPublic, BSafeGetPubKeyModulus, BSafeMakeKeyPair, tripledes3key, tripledes, CBC, DES_ECB_LM, HMACMD5Init, HMACMD5Update, HMACMD5Final, MD2Update, MD2Final, MD4Init, MD4Update, MD4Final, MD5Init, MD5Update, MD5Final, MDbegin, MDupdate, RC2Key, RC2KeyEx, RC2, deskey, des, rc4_key, rc4. These functions are described in more detail below. Data Input is kept separate from Control Input by passing Data Input in separate parameters from Control Input.

Status Output Interface

The Status Output Interface for the Microsoft Windows CE, Windows Mobile Enhanced Cryptographic Provider and Microsoft Windows Embedded Handheld (RSAENH) is a software interface, where applications invoke software functions to perform specific operations. For these functions, status information is returned to the application as the return value from the function, with a non-zero return value

indicating success, and a zero return value indicating failure, and the GetLastError function return a specific error code in the case of failure: CryptAcquireContext, CryptGetProvParam, CryptSetProvParam, CryptReleaseContext, CryptDeriveKey (Derived keys cannot be used for encryption. They can be used to support authentication services only.), CryptDestroyKey, CryptExportKey, CryptGenKey, CryptGenRandom, CryptGetKeyParam, CryptGetUserKey, CryptImportKey, CryptSetKeyParam, CryptDecrypt, CryptEncrypt, CryptCreateHash, CryptDestroyHash, CryptGetHashParam, CryptHashData, CryptHashSessionKey, CryptSetHashParam, CryptSignHash, CryptVerifySignature, CryptDuplicateHash. These function can not fail, and no status code is returned: A_SHAInit, A_SHAUpdate, A_SHAFinal, BSafeGetPubKeyModulus, tripledes3key, tripledes, CBC, HMACMD5Init, HMACMD5Update, HMACMD5Final, MD2Update, MD2Final, MD4Init, MD4Update, MD4Final, MD5Init, MD5Update, MD5Final, MDbegin, MDupdate, RC2Key, RC2KeyEx, RC2, deskey, des, rc4_key, rc4. These functions return non-zero on success, and zero on failure, with no specific error code being available: BSafeComputeKeySizes BSafeDecPrivate BSafeEncPublic BSafeMakeKeyPair. These functions return zero on success, and non-zero on failure, with no specific error code being available: DES_ECB_LM.

SPECIFICATION OF ROLES

RSAENH supports both a User and Cryptographic Officer roles (as defined in FIPS PUB 140-2). Both users have access to all services implemented in the cryptographic module.

An application requests the crypto module to generate keys for a user. Keys are generated, used and deleted as requested by applications. There are not implicit keys associated with a user.

Maintenance Roles

Maintenance roles are not supported by RSAENH.

Multiple Concurrent Operators

Multiple concurrent operators are not supported.

SPECIFICATION OF SERVICES

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

Key Storage Functions

For some functions, RSAENH stores keys in the system registry. The task of covering the keys prior to storage in the system registry is delegated to the Data Protection API (DPAPI) of Microsoft Windows CE, a separate component of the operating system, and outside the boundaries of the cryptographic module. For FIPS purposes, these keys can be considered to be stored in plaintext, and are outside the module boundary.

CryptAcquireContext

The CryptAcquireContext function is used to acquire a handle to a particular key container via a particular cryptographic service provider (CSP). This returned handle can then be used to make calls to the selected CSP.

This function performs two operations. It first attempts to find a CSP with the characteristics described in the *dwProvType* and *pszProvider* parameters. If the CSP 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 CSP.

CryptSetProvParam

The CryptSetProvParam function customizes various aspects of a provider's operations.

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 Functions

The following functions provide interfaces to the cryptographic module's key generation and exchange functions.

CryptDeriveKey

The CryptDeriveKey function generates cryptographic session keys derived from a hash value. This function guarantees that when the same CSP and algorithms are used, the keys generated from the same hash value are identical. The hash value is typically a cryptographic hash (SHA-1, etc.) of a password or similar secret user data.

This function is the same as CryptGenKey, except that the generated session keys are derived from the hash value instead of being random and CryptDeriveKey can only be used to generate session keys. It cannot generate public/private key pairs. (Derived keys cannot be used for encryption. They can be used to support authentication services only.)

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 CSP through CryptImportKey, this function zeroizes the key in memory and frees the memory that the key occupied. If the handle refers to a public/private key pair, this function destroys only the handle and zeroizes any in-memory copies – if the public/private key pair resides in the key storage area in the system registry, then to destroy that copy, the CryptAcquireContext function must be called, passing the CRYPT_DELETEKEYSET flag.

CryptExportKey

The CryptExportKey function exports cryptographic keys from a cryptographic service provider (CSP) in a secure manner for key archival purposes.
A handle to a private RSA key to be exported may be passed to the function, and the function returns a key blob. This private key blob can be sent over a non-secure transport or stored in a non-secure storage location. The private key blob is useless until the intended recipient uses the CryptImportKey function on it to import the key into the recipient's CSP. Key blobs are exported either in plaintext or encrypted with a symmetric key. If a symmetric key is used to encrypt the blob then a handle to the private RSA key is passed in to the module and the symmetric key referenced by the handle is used to encrypt the blob. Any of the supported symmetric cryptographic algorithms may be used to encrypt the private key blob (DES, TripleDES, RC4 or RC2).

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 encrypted with an RSA key using the CryptExportKey function. A handle to the symmetric key and a handle to the public RSA key to encrypt with are passed to the function. The function returns a blob (SIMPLEBLOB) which is the encrypted symmetric key.

Symmetric keys may also be exported by wrapping the keys with another symmetric key. The wrapped key is then exported as a blob and may be imported using the CryptImportKey function.

In order for this function to operate in a FIPS Approved manner, the operator must ensure that keys used to encrypt / protect other keys, should be at least as strong as the key that they are used to protect.

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.

This function uses the Approved PRNG implementation to generate the key, for both symmetric and asymmetric keys.

CryptGenRandom

The CryptGenRandom function fills a buffer with random bytes, implementing an Approved Pseudo Random Number Generator (PRNG). The random number generation algorithm is based on the SHS based RNG from FIPS 186. During the function initialization, a seed, to which SHA-1 is applied to create the output random, is created by calling the Windows CE function CeGenRandom that is implemented outside RSAENH, in the file system. CeGenRandom is pseudo random function that uses several sources of randomness from the OS and hardware:

- The process ID of the current process requesting random data
- The thread ID of the current thread within the process requesting random data
- A 32bit tick count since the system boot
- The current local date and time
- Platform provided hardware Random Number Seed, if available
- Platform provided unique serial number, if available
- CeGetRandomSeed – a 64-bit number that is updated with the low five bits of the current millisecond counter whenever there is a thread switch or a system call.
- The cursor position, as returned by GetMessagePos, on systems with a cursor.
- The amount of free and allocated memory as returned by GlobalMemoryStatus
- The amount of free and allocated space in the object store as returned by GetStoreInformation.
- Data passed to CeGenRandom by applications, as a random number 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 (CSP).

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

A symmetric key encrypted with an RSA public key is imported into the CryptoImportKey function. The function uses the RSA private key exchange key to decrypt the blob and returns a handle to the symmetric key.
Symmetric keys wrapped with other symmetric keys may also be imported using this function. The wrapped key blob is passed in along with a handle to a symmetric key which the module is supposed to use to unwrap the blob. If the function is successful then a handle to the unwrapped 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.

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 Functions

The following functions provide interfaces to the cryptographic module'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 CSP module and is referenced by the hKey parameter.

Hashing and Digital Signature Functions

The following functions provide interfaces to the cryptographic module'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 CSP 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, SHA2 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, RC4 or RC2). For creating a HMAC-FIPS compliant 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.

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.

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-FIPS compliant 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. The CSP is using the inner and outer string values as documented in the HMAC-FIPS as its default values. The caller should not specify the pbInnerString and pbOuterString fields in the HP_HMAC_INFO structure.

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 default format is PKCS#1 (v1.5), while the X9.31 format is supported by passing the CRYPT_X931_FORMAT 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 default format is PKCS#1 (v1.5), while the X9.31 format is supported by passing the CRYPT_X931_FORMAT 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.

Additional Low-Level Functions

These low-level functions are also available through RSAENH, and provide a subset of the functions described above. (The cryptographic functions described above are implemented on the basis of these lower-level functions.)

A_SHAInit A_SHAUpdate A_SHAFinal

The A_SHAInit function initializes a SHA1 hash object. A_SHAUpdate is used to hash data, and A_SHAFinal completes the hash operation, leaving the resulting hash value in the hash object.

BSafeComputeKeySizes

BSafeComputeKeySizes computes the number of bytes required for the public and private keys, based on a particular key size.

BSafeDecPrivate

BSafeDecPrivate decodes a block of encrypted text, resulting in plaintext. For this function, the application maintains storage of the key pair.

BSafeEncPublic

BSafeEncPublic encodes a block of plain text, resulting in encrypted text. For this function, the application maintains storage of the key pair.

BSafeGetPubKeyModulus

BSafeGetPubKeyModulus returns the modulus of a public key.

BSafeMakeKeyPair

BSafeMakeKeyPair generates a public / private key pair of the specified size, placing the keys in storage maintained by the application.

Tripledes3key tripledes

The tripledes2key and tripledes functions implement Triple-DES key generation, encryption, and decryption operations.

aes aeskey

The aes and aeskey functions implement AES key generation, encryption, and decryption operations. These are available only in versions 5.04.17228, 5.05.19202 and 5.05.21840.

CBC

CBC performs cipher block chaining.

DES_ECB_LM

The DES_ECB_LM function implements a Lan Manager hashing function. (This algorithm is not FIPS 140-2 Approved.)

HMACMD5Init HMACMD5Update HMACMD5Final

The HMACMD5Init, HMACMD5Update, and HMACMD5Final functions implement HMAC MD5 operations. (This algorithm is not FIPS 140- 2 Approved.)

MD2Update MD2Final

The MD2Update and MD2Final functions implement MD2 hashing operations. (This algorithm is not FIPS 140-2 Approved.)

MD4Init MD4Update MD4Final

The MD4Init, MD4Update and MD4Final functions implement MD4 hashing operations. (This algorithm is not FIPS 140-2 Approved.)

MD5Init MD5Update MD5Final

The MD5Init, MD5Update and MD5Final functions implement MD5 hashing operations. (This algorithm is not FIPS 140-2 Approved.)

MDbegin MDupdate

The MDbegin and MDupdate functions implement MD4 hashing operations. (This algorithm is not FIPS 140-2 Approved.)

RC2Key RC2KeyEx RC2

The RC2Key, RC2KeyEx, and RC2 functions implement RC2 key generation, encryption, and decryption operations. (This algorithm is not FIPS 140-2 Approved.)

deskey des

The des and deskey functions implement DES key generation, encryption, and decryption operations. (This algorithm is not FIPS 140-2 Approved.)

rc4_key rc4

The rc4_key and rc4 functions implement RC4 key generation, encryption, and decryption operations. (This algorithm is not FIPS 140-2 Approved.)

CRYPTOGRAPHIC KEY MANAGEMENT

The RSAENH cryptographic module manages keys in the following manner.

Key Material

RSAENH handles the following security-related information (secret and private cryptographic keys, authentication data, and other protected information):

Type of Key	Access Privileges	Roles With Access To
RSA Signature Keys	Read / Write / Update / Erase / Zeroize	User, Cryptographic Operator
RSA Key Exchange Keys	Read / Write / Update / Erase / Zeroize	User, Cryptographic Operator
AES Keys	Read / Write / Update / Erase / Zeroize	User, Cryptographic Operator

DES Keys	Read / Write / Update / Erase / Zeroize	User, Cryptographic Operator
Triple-DES Keys	Read / Write / Update / Erase / Zeroize	User, Cryptographic Operator
SHA-1 HMAC Keys	Read / Write / Update / Erase / Zeroize	User, Cryptographic Operator
SHA-2 256 / 384 / 512 HMAC Keys	Read / Write / Update / Erase / Zeroize	User, Cryptographic Operator

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.

Key Generation

Random keys can be generated by calling the CryptGenKey() function, or through the BSafeMakeKeyPair, tripledes3key, RC2Key (non-Approved), RC2KeyEx (nonApproved), deskey (non-Approved), or rc4_key (non-Approved) functions. Keys can also be derived from known values via the CryptDeriveKey() function (Derived keys cannot be used for encryption. They can be used to support authentication services only.)

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.

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. 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 CE Data Protection API (DPAPI) and then outputted to system registry 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.

In addition, specific functions require that the application hold the key material, with the RSAENH module not holding any copy of the key material between function invocations. The functions that operate this way are: BSafeDecPrivate, BSafeEncPublic, BSafeMakeKeyPair, tripledes3key, tripledes, CBC, DES_ECB_LM

(Non-Approved), RC2Key (non-Approved), RC2KeyEx (non-Approved), RC2 (nonApproved), deskey (non-Approved), des (non-Approved), rc4_key (non-Approved), rc4 (non-Approved).

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.

Key Storage

RSAENH offloads the key storage operations to the Microsoft Windows CE operating system. Keys are not stored in the cryptographic module; private keys are protected by the Microsoft Data Protection API (DPAPI) service, and then stored in the registry or file system. For purposes of FIPS validation, these keys are considered plaintext. Keys are zeroized from memory after use. Only 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 registry or file system.

RSA private and public keys are stored in named key containers. The key containers are stored in the following registry locations:
Key containers created with the CRYPT_MACHINE_KEYSET flag:
HKEY_LOCAL_MACHINE\Comm\Security\Crypto\UserKeys\Microsoft Enhanced Cryptographic Provider v1.0\<KeyContainerName>

Key containers created without the CRYPT_MACHINE_KEYSET flag:
HKEY_CURRENT_USER\Comm\Security\Crypto\UserKeys\ Microsoft Enhanced Cryptographic Provider v1.0\<KeyContainerName>

The persisted key container contains the following fields:

- Version
- Name of container
- Signature Public key
- Encrypted Signature Private key
- Signature key Exportability flag
- Key Exchange Public key
- Encrypted Key Exchange Private Key
- Key Exchange exportability flag
- Container random seed

The signature and key exchange fields are only present if the corresponding key has been generated or imported into the container.

In addition, specific functions require that the application hold the key material, with the RSAENH module not holding any copy of the key material between function invocations. The functions that operate this way are: BSafeDecPrivate, BSafeEncPublic, BSafeMakeKeyPair, tripledes3key, tripledes, CBC, DES_ECB_LM (Non-Approved), RC2Key (non-Approved), RC2KeyEx (non-Approved), RC2 (nonApproved), deskey (non-Approved), des (non-Approved), rc4_key (non-Approved), rc4 (non-Approved). For these functions, the application is responsible for maintaining key storage.

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.

In addition, specific functions require that the application hold the key material, with the RSAENH module not holding any copy of the key material between function invocations. The functions that operate this way are: BSafeDecPrivate, BSafeEncPublic, BSafeMakeKeyPair, tripledes3key, tripledes, CBC, DES_ECB_LM (Non-Approved), RC2Key (non-Approved), RC2KeyEx (non-Approved), RC2 (nonApproved), deskey (non-Approved), des (non-Approved), rc4_key (non-Approved), rc4 (non-Approved). For these functions, the application is responsible for key archival.

Key Destruction

All keys are destroyed and their memory location zeroized when the operator calls CryptDestroyKey on that key handle. Private keys (which are stored by the operating system in covered format in the Windows CE DPAPI system portion of the OS) are destroyed when the operator calls CryptAcquireContext with the CRYPT_DELETE_KEYSET flag.

In addition, specific functions require that the application hold the key material, with the RSAENH module not holding any copy of the key material between function invocations. The functions that operate this way are: BSafeDecPrivate, BSafeEncPublic, BSafeMakeKeyPair, tripledes3key, tripledes, CBC, DES_ECB_LM (Non-Approved), RC2Key (non-Approved), RC2KeyEx (non-Approved), RC2 (nonApproved), deskey (non-Approved), des (non-Approved), rc4_key (non-Approved), rc4 (non-Approved). While each of these functions is operating, a copy of the key material may be made by the function – this copy will be zeroized before

the function returns the application. For these functions, the application is responsible for key destruction of the copy of the keys which the application holds.

Security-Related Information Residing in RAM during Operation

RSAENH does not process passwords or PIN's, and thus they are not stored in RAM during operation. Public and private key material is stored in RAM by RSAENH when an application calls CryptAcquireContext. It is held in the memory space of the calling process, in plain text, until the calling process destroys the associated context by calling CryptReleaseContext for that context – when the context is released, the memory holding any keys is zeroized. For operations where key material is passed to a function through a parameter, for use during the function call, the key material may be copied in memory in plain text, with any copies being zeroized before the function returns.

SELF-TESTS

- RSAENH performs these Power-On Self-Tests
  - AES Encrypt / Decrypt Known Answer Test
  - DES Encrypt / Decrypt Known Answer Test
  - Triple-DES Encrypt / Decrypt Known Answer Test
  - SHS (SHA -1, SHA-256, SHA-384, SHA-512) Known Answer Test
  - HMAC (SHA-1, SHA-256, SHA-386, SHA-512) Known Answer Test
  - RSA Sign / Verify using a Sign / Verify test with a Known Signature with PKCS#1 v1.5
  - RSA Sign / Verify using a Sign / Verify test with a Known Signature with X9.31
  - FIPS 186-2 PRNG Known Answer Test
  - Software Integrity Test – RSA (w/ SHA-1) Digital Signature

- RSAENH performs these Conditional Self-Tests
  - CRNGT test for Approved PRNG
  - Pair-wise consistency test for RSA key generation

In all cases for any failure of a Power-On Self-Test, the RSAENH module will fail to load. For the application, this will appear as a failure result code returned from the CryptAcquireContext function. The only way to recover from the failure of a PowerOn Self-Test is to attempt to invoke CryptAcquireContext again, which will re-run the Self-Tests, and will only succeed if the Self-Tests pass.

In all cases for any failure of a Conditional Self-Test, a failure result code will be returned from the particular function that encountered the error. Conditional SelfTests will reset when the function call returns the error status to the application, and future function calls will run any applicable Conditional Self-Tests when they are called.

MISCELLANEOUS

The following items address requirements not addressed above.

Operating System Security

The RSAENH cryptographic module is intended to run on Windows CE, Windows Mobile and Windows Embedded Handheld in Single User Mode.

When an operating system process loads the Microsoft Windows CE, Windows Mobile Enhanced Cryptographic Module or and Microsoft Windows Embedded Handheld (RSAENH) module into memory, RSAENH performs an RSA signature check on the image of the RSAENH.DLL file as it resides in the system's file system, and if the signature check fails, the module load is aborted and an error returned.

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

Secure Operation

The Microsoft Windows CE, Windows Mobile and Windows Embedded Handheld Enhanced Cryptographic Module (RSAENH) is used in FIPS Approved Mode by application, through the invocation of individual functions in FIPS Approved Mode. The application is responsible for ensuring that it does not perform non-Approved functions in ways that make the application non-FIPS Compliant.

The non-Approved functions include:

- Any function using an algorithm which is non-Approved

Mitigation of Other Attacks

The Microsoft Windows CE, Windows Mobile Enhanced Cryptographic Module and Microsoft Windows Embedded Handheld (RSAENH) does not provide any mechanisms to mitigate other attacks.

For more information on Windows CE check out our World Wide Web site at http://www.microsoft.com/windows/embedded.

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