



# Windows Server 2003 Enhanced Cryptographic Provider (RSAENH)

(Windows Server 2003 SP2)

**FIPS 140-2 Documentation: Security Policy**

**September 20, 2007**

---

## **Abstract**

This document specifies the non-proprietary security policy for Windows Server 2003 (SP2) Enhanced Cryptographic Provider (RSAENH) as described in FIPS PUB 140-2.

---

*The information contained in this document represents the current view of Microsoft Corporation on the issues discussed as of the date of publication. Because Microsoft must respond to changing market conditions, it should not be interpreted to be a commitment on the part of Microsoft, and Microsoft cannot guarantee the accuracy of any information presented after the date of publication.*

*This document is for informational purposes only. MICROSOFT MAKES NO WARRANTIES, EXPRESS OR IMPLIED, AS TO THE INFORMATION IN THIS DOCUMENT.*

*Complying with all applicable copyright laws is the responsibility of the user. This work is licensed under the Creative Commons Attribution-NoDerivs-NonCommercial License (which allows redistribution of the work). To view a copy of this license, visit <http://creativecommons.org/licenses/by-nd-nc/1.0/> or send a letter to Creative Commons, 559 Nathan Abbott Way, Stanford, California 94305, USA.*

*Microsoft may have patents, patent applications, trademarks, copyrights, or other intellectual property rights covering subject matter in this document. Except as expressly provided in any written license agreement from Microsoft, the furnishing of this document does not give you any license to these patents, trademarks, copyrights, or other intellectual property.*

*The example companies, organizations, products, people and events depicted herein are fictitious. No association with any real company, organization, product, person or event is intended or should be inferred.*

*© 2003 Microsoft Corporation. All rights reserved.*

*Microsoft, Active Directory, Visual Basic, Visual Studio, Windows, the Windows logo, Windows NT, and Windows Server are either registered trademarks or trademarks of Microsoft Corporation in the United States and/or other countries.*

*The names of actual companies and products mentioned herein may be the trademarks of their respective owners.*

---

## **CONTENTS**

<b>INTRODUCTION .....</b>	<b>4</b>
<b>SECURITY POLICY.....</b>	<b>5</b>
<b>SPECIFICATION OF ROLES.....</b>	<b>6</b>
<b>SPECIFICATION OF SERVICES .....</b>	<b>7</b>
<b>CRYPTOGRAPHIC KEY MANAGEMENT .....</b>	<b>13</b>
<b>SELF-TESTS .....</b>	<b>15</b>
<b>MISCELLANEOUS.....</b>	<b>17</b>
<b>FOR MORE INFORMATION.....</b>	<b>20</b>

---

## **INTRODUCTION**

The Microsoft Corporation's Windows Server 2003 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 Server 2003, Windows Server 2003 Service Pack 1 and Service Pack 2, 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 Server 2003 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 5.2.3790.3959 [Service Pack 2]) tested on an x86, x64, and ia64 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 and Cryptographic API of Microsoft Windows Server 2003 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 Server 2003, Windows Server 2003 SP1 and SP2.
- RSAENH provides no user authentication; however, it relies on Microsoft Windows Server 2003 for the authentication of users.
- 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.
- RSAENH stores keys in the file system, but relies upon Microsoft Windows Server 2003 for the encryption of the keys prior to storage.
- When operating this module under Windows Server 2003 SP 2, the following algorithms are Approved security functions and can be used in FIPS mode:
  - FIPS-approved algorithms: Triple-DES, AES, SHA-1, SHA-256, SHA-384, SHA-512, HMAC, RSA and FIPS186-2 General Purpose random generator. RSAENH also provides the required self-tests for these FIPS-approved algorithms.
- RSAENH supports the following non-FIPS approved algorithms: X9.31 RSA key-pair generation, DES, RC4, RC2, MD2, MD4, and MD5<sup>1</sup>; and though these algorithms may not be used when operating the module in FIPS mode, the module provides power-up self-tests to provide extra security for non FIPS users.
- Operating the module in FIPS Approved mode requires the use of Approved algorithms listed above as well as Approved services which have not been excluded by this security policy.

---

<sup>1</sup> Applications may not use any of these non-FIPS algorithms if they need to be FIPS mode. To operate the module in a FIPS Approved mode, applications must only use FIPS-approved algorithms.

---

## **SPECIFICATION OF ROLES**

RSAENH module supports both a 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 Server 2003 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. This returned handle can then be used to make calls to the selected cryptographic service provider. 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 cryptographic service provider with the characteristics described in the *dwProvType* and *pszProvider* parameters. If the cryptographic service provider 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 cryptographic service provider.

#### **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

The following functions provide interfaces to the cryptomodule's key generation and exchange functions. Note: only FIPS approved PRNG are used for all the Key generation operations

### CryptDeriveKey

The CryptDeriveKey function creates cryptographic session keys from a hash value. This function guarantees that when the same cryptographic service provider and algorithms are used, the keys created from the same hash value are identical. The hash value is typically a cryptographic hash (SHA-1 must be used when operating in FIPS-mode) 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.

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 either the SSL 2.0, SSL 3.0, PCT 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. For more information see the SSL 2.0, SSL 3.0, PCT 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 cryptographic service provider 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 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 nonsecure transport or stored in a nonsecure 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 cryptographic service provider. 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, Triple-DES, RC4 or RC2<sup>2</sup>).

---

<sup>2</sup> Note that DES, RC2 and RC4 may not be used while operating RSAENH in FIPS Approved mode.

---

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.

### **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 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 based on the collection of all the data listed in the Miscellaneous section. `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.

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 `CryptImportKey` 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. 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 cryptographic service provider module 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 cryptographic service provider 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, CALG\_PCT1\_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, PCT 1.0, 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 cryptographic service provider 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.

---

## CRYPTOGRAPHIC KEY MANAGEMENT

The RSAENH cryptomodule manages keys in the following manner.

### Key Material

RSAENH can create and use keys for the following algorithms: RSA Signature, RSA Key Exchange, RC2, RC4, DES, Triple-DES, and AES<sup>3</sup>. 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. Keys can also be created from known values via the CryptDeriveKey() function. DES, Triple-DES, and AES keys<sup>3</sup> are generated following the techniques given in FIPS PUB 186-2, Appendix 3, Random Number Generation and in FIPS 140-2 RNGVS requirements.

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](#) > [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. 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<sup>3</sup>). When private keys are generated or imported from archival, they are covered with the Microsoft Windows Server 2003 Data Protection API (DPAPI) and then outputted 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.

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](#) )

---

<sup>3</sup> Note that DES, RC2 and RC4 may not be used while operating RSAENH in FIPS Approved mode.

---

[Reference](#) > [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 Server2003. 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 Server 2003 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 Server 2003 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 Server 2003 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 performs some of its self-tests upon power up and other self-tests upon encountering a specific condition (key pair or random number generation). Note that RSAENH also provides self-tests for non-FIPS approved algorithms, and though not required, RSAENH provides these tests for extra security. Finally, it should be noted that non-FIPS approved algorithms should not be used if operating RSAENH in FIPS mode.

### **Power-up**

The following FIPS-approved algorithm tests are initiated upon power-up

- Triple-DES ECB encrypt/decrypt KAT
- Triple-DES CBC encrypt/decrypt KAT
- Triple-DES 112 ECB encrypt/decrypt KAT
- Triple-DES 112 CBC encrypt/decrypt KAT
- SHA-1 hash KAT
- SHA-256 KAT
- SHA-384 KAT
- SHA-512 KAT
- SHA-1 HMAC hash KAT
- SHA-256 HMAC hash KAT
- SHA-384 HMAC hash KAT
- SHA-512 HMAC hash KAT
- RSA sign/verify power up test
- Software integrity test via a RSA signature verification of the DLL image
- AES 128 ECB encrypt/decrypt KAT
- AES 192 ECB encrypt/decrypt KAT
- AES 256 ECB encrypt/decrypt KAT
- AES 128 CBC encrypt/decrypt KAT
- AES 192 CBC encrypt/decrypt KAT
- AES 256 CBC encrypt/decrypt KAT
- FIPS186-2 General Purpose random generator

The following non-FIPS approved algorithms power-up tests include (may not be used in FIPS-mode)

- DES ECB encrypt/decrypt KAT
- DES CBC encrypt/decrypt KAT
- RC4 encrypt/decrypt KAT
- RC2 CBC encrypt/decrypt KAT
- RC2 ECB encrypt/decrypt KAT
- MD5 hash 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 Server 2003 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 Server 2003 is for informational purposes only. Microsoft Windows Server 2003 requires authentication from a trusted computer base (TCB<sup>4</sup>) 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 Server 2003 is naturally assigned the operator role when he/she accesses RSAENH.

### **ModularExpOffload**

The ModularExpOffload function offloads modular exponentiation from a cryptographic service provider to a hardware accelerator. The cryptographic service provider will check in the registry for the value HKLM\Software\Microsoft\Cryptography\ExpoOffload that can be the name of a DLL. The cryptographic service provider uses LoadLibrary to load that DLL and calls GetProcAddress to get the OffloadModExpo entry point in the DLL specified in the registry. The cryptographic service provider 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 FIPS mode, this function should not be used.

### **Operating System Security**

The RSAENH cryptomodule is intended to run on Windows Server 2003 in Single User Mode.

When an operating system process loads the cryptomodule into memory, the cryptomodule runs a RSA signature on the cryptomodule's disk image of RSAENH.DLL, excluding the RSA signature, checksum, and export signature resources. This signature is compared to the value stored in the RSA signature resource. Initialization will only succeed if the two values are equal.

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

---

<sup>4</sup> 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](http://www.radium.ncsc.mil/tpep/library/protection_profiles/index.html). At this time, Windows Server 2003 has not been evaluated.

---

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.

## **The Collection of Data Used to Create a Seed for Random Number**

The RSAEnh module uses the FIPS 186-2 general purpose approved PRNG to generate the random data required for symmetric & asymmetric key generation. The PRNG concatenates many different sources of information (detailed below) and the resulting byte stream is hashed with SHA-1 to produce a 20-byte seed value.

- User-supplied data
- 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
- The current system time of day information consisting of the boot time, current time, time zone bias, time zone ID, boot time bias, and sleep time bias
- The current hardware-platform-dependent high-resolution performance-counter value
- The information about the system's current usage of both physical and virtual memory, and page file
- The local disk information including the numbers of sectors per cluster, bytes per sector, free clusters, and clusters that are available to the user associated with the calling thread
- A hash of the environment block for the current process
- Some hardware CPU-specific cycle counters
- The system processor performance information consisting of Idle Process Time, Io Read Transfer Count, Io Write Transfer Count, Io Other Transfer Count, Io Read Operation Count, Io Write Operation Count, Io Other Operation Count, Available Pages, Committed Pages, Commit Limit, Peak Commitment, Page Fault Count, Copy On Write Count, Transition Count, Cache Transition Count, Demand Zero Count, Page Read Count, Page Read Io Count, Cache Read Count, Cache Io Count, Dirty Pages Write Count, Dirty Write Io Count, Mapped Pages Write Count, Mapped Write Io Count, Paged Pool Pages, Non Paged Pool Pages, Paged Pool Allocated space, Paged Pool Free space, Non Paged Pool Allocated space, Non Paged Pool Free space, Free System page table entry, Resident System Code Page, Total System Driver Pages, Total System Code Pages, Non Paged Pool Look aside Hits, Paged Pool Lookaside Hits, Available Paged Pool Pages, Resident System Cache Page, Resident Paged Pool Page, Resident System Driver Page, Cache manager Fast Read with No Wait, Cache manager Fast Read with Wait, Cache manager Fast Read Resource Missed, Cache manager Fast Read Not Possible, Cache manager Fast Memory Descriptor List Read with No Wait, Cache manager Fast Memory Descriptor List Read with Wait, Cache manager Fast Memory Descriptor List Read Resource Missed, Cache manager Fast Memory Descriptor List Read Not Possible, Cache manager Map Data with No Wait, Cache manager Map Data with Wait, Cache manager Map Data with No Wait Miss, Cache manager Map Data Wait Miss, Cache manager Pin-Mapped Data Count, Cache manager Pin-Read with No Wait, Cache manager Pin Read with Wait, Cache manager Pin-Read with No Wait Miss, Cache manager Pin-Read Wait Miss, Cache manager Copy-Read with No Wait, Cache manager Copy-

---

Read with Wait, Cache manager Copy-Read with No Wait Miss, Cache manager Copy-Read with Wait Miss, Cache manager Memory Descriptor List Read with No Wait, Cache manager Memory Descriptor List Read with Wait, Cache manager Memory Descriptor List Read with No Wait Miss, Cache manager Memory Descriptor List Read with Wait Miss, Cache manager Read Ahead IOs, Cache manager Lazy-Write IOs, Cache manager Lazy-Write Pages, Cache manager Data Flushes, Cache manager Data Pages, Context Switches, First Level Translation buffer Fills, Second Level Translation buffer Fills, and System Calls

- The system exception information consisting of Alignment Fix up Count, Exception Dispatch Count, Floating Emulation Count, and Byte Word Emulation Count
- The system lookaside information consisting of Current Depth, Maximum Depth, Total Allocates, Allocate Misses, Total Frees, Free Misses, Type, Tag, and Size
- The system interrupt information consisting of context switches, deferred procedure call count, deferred procedure call rate, time increment, deferred procedure call bypass count, and asynchronous procedure call bypass count
- The system process information consisting of Next Entry Offset, Number Of Threads, Create Time, User Time, Kernel Time, Image Name, Base Priority, Unique Process ID, Inherited from Unique Process ID, Handle Count, Session ID, Page Directory Base, Peak Virtual Size, Virtual Size, Page Fault Count, Peak Working Set Size, Working Set Size, Quota Peak Paged Pool Usage, Quota Paged Pool Usage, Quota Peak Non Paged Pool Usage, Quota Non Paged Pool Usage, Page file Usage, Peak Page file Usage, Private Page Count, Read Operation Count, Write Operation Count, Other Operation Count, Read Transfer Count, Write Transfer Count, and Other Transfer Count

---

**FOR MORE INFORMATION**

For the latest information on Windows Server 2003, check out our World Wide Web site at <http://www.microsoft.com/windows>.

Information regarding FIPS 140-2 and FIPS 186-2 can be found on the CMVP website at <http://csrc.nist.gov/cryptval/>.