Windows NT®

Operating System

Microsoft Base Cryptographic Provider

FIPS 140-1 Documentation: Security Policy

September 20, 2000 11:42 AM

Abstract

This document specifies the security policy for the Microsoft Base Cryptographic Provider (RSABASE) as described in FIPS PUB 140-1.

CONTENTS

{ TC "INTRODUCTION" \F SP }INTRODUCTION

Microsoft Base Cryptographic Provider (RSABASE) is a FIPS 140-1 Level 1 compliant, general-purpose, software-based, cryptographic module. Like other cryptographic providers that ship with Microsoft Windows NT 4.0, RSABASE 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 FIPS 140-1 Level 1 compliant cryptography.

Cryptographic Boundary

The Microsoft Base Cryptographic Provider (RSABASE) consists of a single dynamically-linked library (DLL) named RSABASE.DLL. The cryptographic boundary for RSABASE 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 140-1, is Multi-Chip Standalone.

{ TC "SECURITY POLICY" \F SP }SECURITY POLICY

RSABASE operates under several rules that encapsulate its security policy.

- RSABASE is supported on Windows NT 4.0 with Service Pack 6 or later.
- RSABASE relies on Microsoft Windows NT 4.0 for the authentication of users.
- RSABASE enforces a single role, Authenticated User, which is a combination of the User and Cryptographic Officer roles as defined in FIPS PUB 140-1.
- All users authenticated by Microsoft Windows NT 4.0 employ the Authenticated User role.
- All services implemented within RSABASE are available to the Authenticated User role.
- Keys created within RSABASE by one user are not accessible to any other user via RSABASE.
- RSABASE relies on Microsoft Windows NT 4.0 for the secure storage of keys.
- RSABASE performs the following self-tests upon power up:
  - RC4 encrypt/decrypt
  - RC2 ECB encrypt/decrypt
  - DES ECB encrypt/decrypt
  - RC2 CBC encrypt/decrypt
  - DES CBC encrypt/decrypt
  - MD5 hash
  - SHA-1 hash
- RSABASE performs a pairwise consistency test upon each invocation of RSA key generation as defined in FIPS PUB 140-1.

{ TC "SPECIFICATION OF ROLES" \F SP }SPECIFICATION OF ROLES

RSABASE combines the User and Cryptographic Officer roles (as defined in FIPS PUB 140-1) into a single role hereon called the Authenticated User role. The Authenticated User may access 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. Each user may have numerous keys, signature and key exchange, and these keys are separate from other users' keys.

Maintenance Roles

Maintenance roles are not supported by RSABASE.

Multiple Concurrent Operators

RSABASE is intended to run on Windows NT 4.0 with Service Pack 6 or later in Single User Mode. When run in this configuration, multiple concurrent operators are not supported.

{ TC "SPECIFICATION OF SERVICES" \F SP }SPECIFICATION OF SERVICES

The following list contains all services available to an operator. All services are accessible by all Authenticated Users, the one and only role supported by RSABASE.

Key Storage

RSABASE does not store keys. It delegates that task to the Protected Storage Service (PSTORE) of Microsoft Windows NT 4.0, a separate component of the operating system, and outside the boundaries of the cryptomodule.

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, from PSTORE, via the CSP 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.

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 container, enumerate supported algorithms, and generally determine capabilities of the CSP.

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

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

The module allows the use of RSA Keys symmetric key distribution. Symmetric key distribution is supported through the CryptExportKey and CryptImportKey functions. Encryption with a RSA public key and decryption with the RSA private key is only used for symmetric key distribution.

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.

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. The underlying public/private key pair is not destroyed by this function. Only the handle is destroyed.

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 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 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 algorithm's may be used to encrypt the private key blob (DES, 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.

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.

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.

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

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 CSP module and is referenced by the hKey parameter.

Hashing and Digital Signatures

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 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 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, RC4 or RC2).

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.

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.

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.

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.

{ TC "CRYPTOGRAPHIC KEY MANAGEMENT" \F SP }CRYPTOGRAPHIC KEY MANAGEMENT

The RSABASE cryptomodule manages keys in the following manner.

Key Material

RSABASE can create and use keys for the following algorithms: RSA Signature, RSA Key Exchange, RC2, RC4, and DES.

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. Keys can also be derived from known values via the CryptDeriveKey() function.   DES key are generated and validated following the manner described in FIPS PUB 46-2 and FIPS PUB 81.

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 RSABASE 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 (DES, RC4 or RC2). When private keys are generated or imported from archival, they are outputted to the Microsoft Windows NT 4.0 PSTORE in a covered form.

Symmetric key entry and output is done by exchanging keys using the recipient’s asymmetric public 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.

Key Storage

RSABASE offloads the key storage operations to the Microsoft Windows NT 4.0 operating system. Keys are not stored in the cryptographic module, private keys are stored in the Microsoft Protected Storage System (PSTORE) service of the Microsoft Windows NT 4.0 in the manner described below. Keys are zeroized from memory after use.   Only the key used for power up self-testing is stored in the cryptographic module.

When an Authenticated User requests a keyed cryptographic operation from RSABASE his/her keys are retrieved from the Microsoft Protected Storage System (PSTORE) service of the Microsoft Windows NT 4.0. The PSTORE service, called via a local procedure call (LPC) from RSABASE, runs as Local System (analogous to the UNIX root account) and receives the caller's security token via the LPC call parameters. PSTORE then impersonates the user and retrieves the Security Identity Descriptor (SID) from the user token. PSTORE uses the SID to navigate to the following System Registry location:

```
HKEY_CURRENT_USER\Software\Microsoft\Protected Storage System Provider\<retrieved
user SID>Data\<data GUID>\
```

The <data GUID> is a PSTORE specific identifier that describes the type of data being protected. This System Registry location contains the private keys in a covered format. The System Registry folders containing cryptographic keys are additionally protected by Access Control Lists (ACLs) that limit access to only Local System or BUILTIN\Administrator. The PSTORE services, after retrieving the covered key, uncovers the key and returns it via LPC back to the caller.

Key Archival

RSABASE does not directly archive cryptographic keys. The Authenticated User 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 Authenticated User calls CryptDestroyKey on that key handle. Private keys (which are stored by the operating system in covered format in the protected storage system portion of the NT4.0 OS) are destroyed when the Authenticated User calls CryptAcquireContext with the CRYPT_DELETE_KEYSET flag.

{ TC "SELF-TESTS" \F SP }SELF-TESTS

Mandatory

Software tests via a DES MAC of library image

- RC4 encrypt/decrypt KAT
- RC2 ECB encrypt/decrypt KAT
- DES ECB encrypt/decrypt KAT
- RC2 CBC encrypt/decrypt KAT
- DES CBC encrypt/decrypt KAT
- MD5 hash KAT
- SHA-1 hash KAT
- RSA pairwise consistency test

Conditional

The following are initiated at key generation:

- RSA pairwise consistency test

{ TC "MISCELLANEOUS" \F SP }MISCELLANEOUS

The following items address requirements not addressed above.

Cryptographic Bypass

Cryptographic bypass is not support in RSABASE.

Operation Authentication

RSABASE inherits all authentication from the Microsoft Windows NT 4.0 operating system upon which it runs. Microsoft Windows NT 4.0 requires authentication from a trusted control 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 Authenticated User's security token. All subsequent processes and threads created by that Authenticated User are implicitly assigned the parent's (thus the Authenticated User's) security token. Every user that has been authenticated by Microsoft Windows NT 4.0 is naturally assigned the Authenticated User role when he/she accesses RSABASE.

Identity-based Authentication

While all Authenticated Users are assigned the same role and thus have access to the same complete set of services, individual Authenticated Users may only access key containers which they themselves have created. RSABASE assumes the authentication of the user and enforces it by running in a thread with the Authenticated User's security token.

Operating System Security

The RSABASE cryptomodule is intended to run on Windows NT 4.0 with Service Pack 6 or later in Single User Mode.

When an operating system process loads the cryptomodule into memory, the cryptomodule runs a DES MAC on the cryptomodule's disk image of RSABASE.DLL, excluding the DES MAC, checksum, and export signature resources. This MAC is compared to the value stored in the DES MAC 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.

{ TC "FOR MORE INFORMATION" \F SP }FOR MORE INFORMATION

For the latest information on Windows NT Server, check out our World Wide Web site at http://www.microsoft.com/ntserver or the Windows NT Server Forum on the MSN™ network of Internet services (GO WORD: MSNTS).

®

Microsoft®

Windows NT®

Operating System

Microsoft Base DSS Cryptographic Provider

FIPS 140-1 Documentation: Security Policy

September 20, 2000 11:36 AM

Abstract

This document specifies the security policy for the Microsoft Base DSS Cryptographic Provider (DSSBASE) as described in FIPS PUB 140-1.

CONTENTS

{ TC "INTRODUCTION" \F SP }INTRODUCTION

Microsoft Base DSS Cryptographic Provider (DSSBASE) is a FIPS 140-1 Level 1 compliant, general-purpose, software-based, cryptographic module. Like other cryptographic providers that ship with Microsoft Windows NT 4.0, DSSBASE 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 FIPS 140-1 Level 1 compliant cryptography.

Cryptographic Boundary

The Microsoft Base DSS Cryptographic Provider (DSSBASE) consists of a single dynamically-linked library (DLL) named DSSBASE.DLL. The cryptographic boundary for DSSBASE 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 140-1, is Multi-Chip Standalone.

{ TC "SECURITY POLICY" \F SP }SECURITY POLICY

DSSBASE operates under several rules that encapsulate its security policy.

- DSSBASE is supported on Windows NT 4.0 with Service Pack 6 or later.
- DSSBASE relies on Microsoft Windows NT 4.0 for the authentication of users.
- DSSBASE enforces a single role, Authenticated User, which is a combination of the User and Cryptographic Officer roles as defined in FIPS PUB 140-1.
- All users authenticated by Microsoft Windows NT 4.0 employ the Authenticated User role.
- All services implemented within DSSBASE are available to the Authenticated User role.
- Keys created within DSSBASE by one user are not accessible to any other user via DSSBASE.
- DSSBASE relies on Microsoft Windows NT 4.0 for the secure storage of keys.
- DSSBASE performs the following self-tests upon power up:
  - RC4 encrypt/decrypt
  - RC2 ECB encrypt/decrypt
  - DES ECB encrypt/decrypt
  - DES40 ECB encrypt/decrypt
  - RC2 CBC encrypt/decrypt
  - DES CBC encrypt/decrypt
  - DES40 CBC encrypt/decrypt
  - MD5 hash
  - SHA-1 hash
- DSSBASE performs a pairwise consistency test upon each invocation of DSA key generation as defined in FIPS PUB 140-1 and FIPS PUB 186.

{ TC "SPECIFICATION OF ROLES" \F SP }SPECIFICATION OF ROLES

DSSBASE combines the User and Cryptographic Officer roles (as defined in FIPS PUB 140-1) into a single role hereon called the Authenticated User role. The Authenticated User may access 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. Each user may have numerous keys, signature and key exchange, and these keys are separate from other users' keys.

Maintenance Roles

Maintenance roles are not supported by DSSBASE.

Multiple Concurrent Operators

DSSBASE is intended to run on Windows NT 4.0 with Service Pack 6 or later in Single User Mode. When run in this configuration, multiple concurrent operators are not supported.

{ TC "SPECIFICATION OF SERVICES" \F SP **}**SPECIFICATION OF SERVICES

The following list contains all services available to an operator. All services are accessible by all Authenticated Users, the one and only role supported by DSSBASE.

Key Storage

DSSBASE does not store keys. It delegates that task to the Protected Storage Service (PSTORE) of Microsoft Windows NT 4.0, a separate component of the operating system, and outside the boundaries of the cryptomodule.

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, from PSTORE, via the CSP 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.

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 container, enumerate supported algorithms, and generally determine capabilities of the CSP.

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

The following functions provide interfaces to the cryptomodule'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.

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. The underlying public/private key pair is not destroyed by this function. Only the handle is destroyed.

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 DSS/DH 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 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 DSS/DH 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 algorithm's may be used to encrypt the private key blob (DES, DES40, RC4 or RC2).

Public DSS/DH keys are also exported using this function. A handle to the DSS/DH 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.

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.

Generation of a DSS key for signatures requires the operator to complete several steps before a DSS key is generated. CryptGenKey is first called with CRYPT_PREGEN set in the dwFlags parameter. The operator then sets the P, Q, and G for the key generation via CryptSetKeyParam, once for each parameter. The operator calls CryptSetKeyParam with KP_X set as dwParam to complete the key generation.

CryptGenRandom

The CryptGenRandom function fills a buffer with random bytes. The random number generation algorithm is the SHS based RNG from FIPS 186.

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.

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

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 CSP module and is referenced by the hKey parameter.

Hashing and Digital Signatures

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 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 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, DES40, RC4 or RC2).

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.

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 DSS.

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 DSS signatures.

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.

{ TC "CRYPTOGRAPHIC KEY MANAGEMENT" \F SP }CRYPTOGRAPHIC KEY MANAGEMENT

The DSSBASE cryptomodule manages keys in the following manner.

Key Material

DSSBASE can create and use keys for the following algorithms: DSS, Diffie-Hellman, RC2, RC4, DES, and DES40.

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. Keys can also be derived from known values via the CryptDeriveKey() function. DSS keys are generated and validated following the manner described in FIPS PUB 186-1. DES key are generated and validated following the manner described in FIPS PUB 46-2 and FIPS PUB 81.

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 DSSBASE 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 (DES, DES40, RC4 or RC2). When private keys are generated or imported from archival, they are outputted to the Microsoft Windows NT 4.0 PSTORE in a covered form.

Symmetric key entry and output is done by exchanging keys using the recipient’s asymmetric public 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.

Key Storage

DSSBASE offloads the key storage operations to the Microsoft Windows NT 4.0 operating system. Keys are not stored in the cryptographic module, private keys are stored in the Microsoft Protected Storage System (PSTORE) service of the Microsoft Windows NT 4.0 in the manner described below. Keys are zeroized from memory after use.   Only the key used for power up self-testing is stored in the cryptographic module.

When an Authenticated User requests a keyed cryptographic operation from DSSBASE his/her keys are retrieved from the Microsoft Protected Storage System (PSTORE) service of the Microsoft Windows NT 4.0. The PSTORE service, called via a local procedure call (LPC) from DSSBASE, runs as Local System (analogous to the UNIX root account) and receives the caller's security token via the LPC call parameters. PSTORE then impersonates the user and retrieves the Security Identity Descriptor (SID) from the user token. PSTORE uses the SID to navigate to the following System Registry location:

```
HKEY_CURRENT_USER\Software\Microsoft\Protected Storage System Provider\<retrieved
user SID>Data\<data GUID>\
```

The <data GUID> is a PSTORE specific identifier that describes the type of data being protected. This System Registry location contains the private keys in a covered format. The System Registry folders containing cryptographic keys are additionally protected by Access Control Lists (ACLs) that limit access to only Local System or BUILTIN\Administrator. The PSTORE services, after retrieving the covered key, uncovers the key and returns it via LPC back to the caller.

Key Archival

DSSBASE does not directly archive cryptographic keys. The Authenticated User 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 Authenticated User calls CryptDestroyKey on that key handle. Private keys (which are stored by the operating system in covered format in the protected storage system portion of the NT4.0 OS) are destroyed when the Authenticated User calls CryptAcquireContext with the CRYPT_DELETE_KEYSET flag.

{ TC "SELF-TESTS" \F SP }SELF-TESTS

Mandatory

Software tests via a DES MAC of library image

- RC4 encrypt/decrypt KAT
- RC2 ECB encrypt/decrypt KAT
- DES ECB encrypt/decrypt KAT
- DES40 ECB encrypt/decrypt KAT
- RC2 CBC encrypt/decrypt KAT
- DES CBC encrypt/decrypt KAT
- DES40 CBC encrypt/decrypt KAT
- MD5 hash KAT
- SHA-1 hash KAT
- DSS pairwise consistency test
- Diffie-Hellman pairwise consistency test

Conditional

The following are initiated at key generation:

- DSS pairwise consistency test
- Diffie-Hellman pairwise consistency test

{ TC "MISCELLANEOUS" \F SP }MISCELLANEOUS

The following items address requirements not addressed above.

Cryptographic Bypass

Cryptographic bypass is not support in DSSBASE.

Operation Authentication

DSSBASE inherits all authentication from the Microsoft Windows NT 4.0 operating system upon which it runs. Microsoft Windows NT 4.0 requires authentication from a trusted control 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 Authenticated User's security token. All subsequent processes and threads created by that Authenticated User are implicitly assigned the parent's (thus the Authenticated User's) security token. Every user that has been authenticated by Microsoft Windows NT 4.0 is naturally assigned the Authenticated User role when he/she accesses DSSBASE.

Identity-based Authentication

While all Authenticated Users are assigned the same role and thus have access to the same complete set of services, individual Authenticated Users may only access key containers which they themselves have created. DSSBASE assumes the authentication of the user and enforces it by running in a thread with the Authenticated User's security token.

Operating System Security

The DSSBASE cryptomodule is intended to run on Windows NT 4.0 with Service Pack 6 or later in Single User Mode.

When an operating system process loads the cryptomodule into memory, the cryptomodule runs a DES MAC on the cryptomodule's disk image of DSSBASE.DLL, excluding the DES MAC, checksum, and export signature resources. This MAC is compared to the value stored in the DES MAC 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.

{ TC "FOR MORE INFORMATION" \F SP }FOR MORE INFORMATION

For the latest information on Windows NT Server, check out our World Wide Web site at http://www.microsoft.com/ntserver or the Windows NT Server Forum on the MSN™ network of Internet services (GO WORD: MSNTS).

®

Microsoft®

Windows NT®

Operating System

Microsoft Enhanced Cryptographic Provider

FIPS 140-1 Documentation: Security Policy

September 20, 2000 11:45 AM

Abstract

This document specifies the security policy for the Microsoft Enhanced Cryptographic Provider (RSAENH) as described in FIPS PUB 140-1.

CONTENTS

{ TC "INTRODUCTION" \F SP }INTRODUCTION

Microsoft Enhanced Cryptographic Provider (RSAENH) is a FIPS 140-1 Level 1 compliant, general-purpose, software-based, cryptographic module. Like other cryptographic providers that ship with Microsoft Windows NT 4.0, 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 FIPS 140-1 Level 1 compliant cryptography.

Cryptographic Boundary

The Microsoft Enhanced Cryptographic Provider (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 140-1, is Multi-Chip Standalone.

{ TC "SECURITY POLICY" \F SP **}**SECURITY POLICY

RSAENH operates under several rules that encapsulate its security policy.

- RSAENH is supported on Windows NT 4.0 with Service Pack 6 or later.
- RSAENH relies on Microsoft Windows NT 4.0 for the authentication of users.
- RSAENH enforces a single role, Authenticated User, which is a combination of the User and Cryptographic Officer roles as defined in FIPS PUB 140-1.
- All users authenticated by Microsoft Windows NT 4.0 employ the Authenticated User role.
- All services implemented within RSAENH are available to the Authenticated User role.
- Keys created within RSAENH by one user are not accessible to any other user via RSAENH.
- RSAENH relies on Microsoft Windows NT 4.0 for the secure storage of keys.
- RSAENH performs the following self-tests upon power up:
  - RC4 encrypt/decrypt
  - RC2 ECB encrypt/decrypt
  - RC2 CBC encrypt/decrypt
  - DES ECB encrypt/decrypt
  - DES CBC encrypt/decrypt
  - DES40 ECB encrypt/decrypt
  - 3DES 112 CBC encrypt/decrypt
  - 3DES 112 ECB encrypt/decrypt
  - 3DES CBC encrypt/decrypt
  - 3DES ECB encrypt/decrypt
  - MD5 hash
  - SHA-1 hash
- RSAENH performs a pairwise consistency test upon each invocation of RSA key generation as defined in FIPS PUB 140-1.

{ TC "SPECIFICATION OF ROLES" \F SP }SPECIFICATION OF ROLES

RSAENH combines the User and Cryptographic Officer roles (as defined in FIPS PUB 140-1) into a single role hereon called the Authenticated User role. The Authenticated User may access 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. Each user may have numerous keys, 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 NT 4.0 with Service Pack 6 or later in Single User Mode. When run in this configuration, multiple concurrent operators are not supported.

{ TC "SPECIFICATION OF SERVICES" \F SP }SPECIFICATION OF SERVICES

The following list contains all services available to an operator. All services are accessible by all Authenticated Users, the one and only role supported by RSAENH.

Key Storage

RSAENH does not store keys. It delegates that task to the Protected Storage Service (PSTORE) of Microsoft Windows NT 4.0, a separate component of the operating system, and outside the boundaries of the cryptomodule.

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, from PSTORE, via the CSP 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.

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 container, enumerate supported algorithms, and generally determine capabilities of the CSP.

With the RSAENH module a function table may be retrieved using CryptGetProvParam. On NT4 SP6 this function table is used by the secure channel engine, schannel.dll. For more information see the section on "SCHANNEL Integration with RSAENH".

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

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

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 is passed to the function. The function returns a blob which is the encrypted symmetric key.

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.

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. The underlying public/private key pair is not destroyed by this function. Only the handle is destroyed.

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 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 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 algorithm's may be used to encrypt the private key blob (DES, 3DES, 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.

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.

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.

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.

SCHANNEL INTEGRATION WITH RSAENH

SCHANNEL has a specialized process for calling some functions within the CSP directly:

1) SCHANNEL calls CryptAcquireContext to get a handle to the RSAENH CSP.

2) SCHANNEL requests a pointer to an opaque blob from RSAENH using the CryptGetProvParam function with parameter 41 (This value is not documented anywhere).

3) RSAENH creates an RSA32_FUNCTION_TABLE by setting dwVersion to 1, dwMagic to "NTSH", and putting pointers to its internal functions into the other fields:

```
typedef struct _RSA32_FUNCTION_TABLE {
    DWORD                           dwVersion;
    DWORD                           dwMagic;

    MD2_UPDATE_FN                   MD2Update;
    MD2_FINAL_FN                    MD2Final;
    MD5_INIT_FN                     MD5Init;
    MD5_UPDATE_FN                   MD5Update;
    MD5_FINAL_FN                    MD5Final;
    SHA_INIT_FN                     SHAInit;
    SHA_UPDATE_FN                   SHAUpdate;
    SHA_FINAL_FN                    SHAFinal;

    RC4_KEY_FN                      rc4_key;
    RC4_FN                          rc4;
    RC2_KEY_EX_FN                   RC2KeyEx;
    RC2_FN                          RC2;
    DES_KEY_FN                      deskey;
    DES_FN                          des;
    DES3_KEY_FN                     des3key;
    DES3_FN                         des3;

    RSA_PUBLIC_ENCRYPT_FN           RSAPublicEncrypt;
    RSA_PRIVATE_DECRYPT_FN          RSAPrivateDecrypt;
    COMPUTE_KEY_SIZES_FN            ComputeKeySizes;
    MAKE_KEY_PAIR_FN                MakeKeyPair;
    GET_PUB_KEY_MODULUS_FN          GetPubKeyModulus;

} RSA32_FUNCTION_TABLE, *PRSA32_FUNCTION_TABLE;
```

4) RSAENH converts the RSA32_FUNCTION_TABLE into an blob by covering it with RC4, this is to discourage other applications from using this function table, and returns the blob to SCHANNEL.

MD2UPDATE

void MD2Update(MD2_CTX *ctx, unsigned char *pbData, unsigned int cbData)

The MD2Update function adds data to a specified hash object. This function can be called multiple times to compute the hash on long data streams or discontinuous data streams. The MD2Final function must be called before retrieving the hash value.

MD2Final

void MD2Final(MD2_CTX *ctx)

The MD2Final function computes the final hash of the data entered by the MD2Update function.

MD5Init

void MD5Init(MD5_CTX *ctx)

The MD5Init function initiates the hashing of a stream of data.

MD5Update

void MD5Update(MD5_CTX *, const unsigned char *, unsigned int)

The MD5Update function adds data to a specified hash object. This function can be called multiple times to compute the hash on long data streams or discontinuous data streams. The MD5Final function must be called before retrieving the hash value.

MD5Final

void MD5Final(MD5_CTX *)

The MD5Final function computes the final hash of the data entered by the MD5Update function.

SHAInit

void SHAInit(A_SHA_CTX *)

The SHAInit function initiates the hashing of a stream of data.

SHAUpdate

void SHAUpdate(A_SHA_CTX *, unsigned char *, unsigned int)

The SHAUpdate function adds data to a specified hash object. This function can be called multiple times to compute the hash on long data streams or discontinuous data streams. The SHAFinal function must be called before retrieving the hash value.

SHAFinal

void SHAFinal(A_SHA_CTX *, unsigned char [A_SHA_DIGEST_LEN])

The SHAFinal function computes the final hash of the data entered by the MD5Update function.

rc4_key

void rc4_key(struct RC4_KEYSTRUCT *pKS, DWORD dwLen, unsigned char *pbKey)

Generate the key control structure. Keys can be any size. The keys are checked twice before the routine returns.

rc4

void rc4(struct RC4_KEYSTRUCT *pKS, DWORD dwLen, unsigned char *pbuf)

Encrypt the buffer using RC4.

RC2KeyEx

void RC2KeyEx(WORD *keyTable, BYTE *key, DWORD keyLen, DWORD eSpace)

Generate the key control structure. Key can be any size. The keys are checked twice before the routine returns.

RC2

void RC2(BYTE *pbIn, BYTE *pbOut, void *pwKT, int op)

Encrypt the buffer using RC2.

deskey

void deskey(DESTable *,unsigned char *)

Fill in the DESTable struct with the decrypt and encrypt key expansions.

Assumes that the second parameter points to DES_BLOCKLEN bytes of key. The keys are checked twice before the routine returns.

des

void des(BYTE *pbOut, BYTE *pbIn, void *key, int op)

Encrypt the buffer using DES. The keys are zeroized after use.

des3key

void des3key(PDES3TABLE pDES3Table, PBYTE pbKey)

Fill in the DES3Table structs with the decrypt and encrypt key expansions.

Assumes that the second parameter points to 3 * DES_BLOCKLEN bytes of key. The keys are checked twice before the routine returns.

des3

void des3(PBYTE pbIn, PBYTE pbOut, void *pKey, int op)

Encrypt the buffer using DES3. Keys are zeroized after use.

RSAPublicEncrypt

BOOL RSAPublicEncrypt(const LPBSAFE_PUB_KEY pBSPubKey, BYTE *pbInput, BYTE *pbOutput)

RSA encrypt a buffer of size key->keylen, filled with data of size key->datalen with the public key pointed to by key, returning the encrypted data in part_out.

RSAPrivateDecrypt

BOOL RSAPrivateDecrypt(const LPBSAFE_PRV_KEY pBSPrivKey, BYTE *pbInput, BYTE *pbOutput)

RSA decrypt a buffer of size keylen, containing key->datalen bytes of data with the private key pointed to by key, returning the decrypted data in part_out. Zeroizes the key after use.

ComputeKeySizes

BOOL BSafeComputeKeySizes(LPDWORD PublicKeySize, LPDWORD PrivateKeySize, LPDWORD bits)

Returns pointers to the parts of a private key, and the length of the modulus in bytes.

MakeKeyPair

BOOL BSafeMakeKeyPair(LPBSAFE_PUB_KEY public_key, LPBSAFE_PRV_KEY private_key, DWORD bits)

Generate an RSA key pair. Two checks are made before the key is exported from the module. A pairwise consistency test is done on the key after it is generated.

GetPubKeyModulus

BYTE * BSafeGetPubKeyModulus(LPBSAFE_PUB_KEY key)

Returns pointer to the modulus of a public key.

Data Encryption and Decryption

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 CSP module and is referenced by the hKey parameter.

Hashing and Digital Signatures

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 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 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, 3DES, RC4 or RC2).

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.

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.

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.

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.

{ TC "CRYPTOGRAPHIC KEY MANAGEMENT" \F SP }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, and 3DES.

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. Keys can also be derived from known values via the CryptDeriveKey() function.   DES key are generated and validated following the manner described in FIPS PUB 46-2 and FIPS PUB 81.

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 (DES, 3DES, RC4 or RC2). When private keys are generated or imported from archival, they are outputted to the Microsoft Windows NT 4.0 PSTORE in a covered form.

Symmetric key entry and output is done by exchanging keys using the recipient’s asymmetric public 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.

Key Storage

RSAENH offloads the key storage operations to the Microsoft Windows NT 4.0 operating system. Keys are not stored in the cryptographic module, private keys are stored in the Microsoft Protected Storage System (PSTORE) service of the Microsoft Windows NT 4.0 in the manner described below. Keys are zeroized from memory after use.   Only the key used for power up self-testing is stored in the cryptographic module.

When an Authenticated User requests a keyed cryptographic operation from RSAENH his/her keys are retrieved from the Microsoft Protected Storage System (PSTORE) service of the Microsoft Windows NT 4.0. The PSTORE service, called via a local procedure call (LPC) from RSAENH, runs as Local System (analogous to the UNIX root account) and receives the caller's security token via the LPC call parameters. PSTORE then impersonates the user and retrieves the Security Identity Descriptor (SID) from the user token. PSTORE uses the SID to navigate to the following System Registry location:

```
HKEY_CURRENT_USER\Software\Microsoft\Protected Storage System Provider\<retrieved
user SID>Data\<data GUID>\
```

The <data GUID> is a PSTORE specific identifier that describes the type of data being protected. This System Registry location contains the private keys in a covered format. The System Registry folders containing cryptographic keys are additionally protected by Access Control Lists (ACLs) that limit access to only Local System or BUILTIN\Administrator. The PSTORE services, after retrieving the covered key, uncovers the key and returns it via LPC back to the caller.

Key Archival

RSAENH does not directly archive cryptographic keys. The Authenticated User 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 Authenticated User calls CryptDestroyKey on that key handle. Private keys (which are stored by the operating system in covered format in the protected storage system portion of the NT4.0 OS) are destroyed when the Authenticated User calls CryptAcquireContext with the CRYPT_DELETE_KEYSET flag.

{ TC "SELF-TESTS" \F SP }SELF-TESTS

Mandatory

Software tests via a DES MAC of library image

- RC4 encrypt/decrypt KAT
- RC2 ECB encrypt/decrypt KAT
- DES ECB encrypt/decrypt KAT
- 3DES ECB encrypt/decrypt KAT
- 3DES 112 ECB encrypt/decrypt KAT
- RC2 CBC encrypt/decrypt KAT
- DES CBC encrypt/decrypt KAT
- 3DES CBC encrypt/decrypt KAT
- 3DES 112 CBC encrypt/decrypt KAT
- MD5 hash KAT
- SHA-1 hash KAT
- RSA pairwise consistency test

Conditional

The following are initiated at key generation:

- RSA pairwise consistency test

{ TC "MISCELLANEOUS" \F SP }MISCELLANEOUS

The following items address requirements not addressed above.

Cryptographic Bypass

Cryptographic bypass is not support in RSAENH.

Operation Authentication

RSAENH inherits all authentication from the Microsoft Windows NT 4.0 operating system upon which it runs. Microsoft Windows NT 4.0 requires authentication from a trusted control 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 Authenticated User's security token. All subsequent processes and threads created by that Authenticated User are implicitly assigned the parent's (thus the Authenticated User's) security token. Every user that has been authenticated by Microsoft Windows NT 4.0 is naturally assigned the Authenticated User role when he/she accesses RSAENH.

Identity-based Authentication

While all Authenticated Users are assigned the same role and thus have access to the same complete set of services, individual Authenticated Users may only access key containers which they themselves have created. RSAENH assumes the authentication of the user and enforces it by running in a thread with the Authenticated User's security token.

Operating System Security

The RSAENH cryptomodule is intended to run on Windows NT 4.0 with Service Pack 6 or later in Single User Mode.

When an operating system process loads the cryptomodule into memory, the cryptomodule runs a DES MAC on the cryptomodule's disk image of RSAENH.DLL, excluding the DES MAC, checksum, and export signature resources. This MAC is compared to the value stored in the DES MAC 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.

{ TC "FOR MORE INFORMATION" \F SP }FOR MORE INFORMATION

For the latest information on Windows NT Server, check out our World Wide Web site at http://www.microsoft.com/ntserver or the Windows NT Server Forum on the MSN™ network of Internet services (GO WORD: MSNTS).

Windows NT®

Operating System

Microsoft DSS/Diffie-Hellman Enhanced Cryptographic Provider

FIPS 140-1 Documentation: Security Policy

September 20, 2000 11:36 AM

Abstract

This document specifies the security policy for the Microsoft DSS/Diffie-Hellman Enhanced Cryptographic Provider (DSSENH) as described in FIPS PUB 140-1.

CONTENTS

{ TC "INTRODUCTION" \F SP }INTRODUCTION

Microsoft DSS/Diffie-Hellman Enhanced Cryptographic Provider (DSSENH) is a FIPS 140-1 Level 1 compliant, general-purpose, software-based, cryptographic module. Like other cryptographic providers that ship with Microsoft Windows NT 4.0, DSSENH 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 FIPS 140-1 Level 1 compliant cryptography.

Cryptographic Boundary

The Microsoft DSS/Diffie-Hellman Enhanced Cryptographic Provider (DSSENH) consists of a single dynamically-linked library (DLL) named DSSENH.DLL. The cryptographic boundary for DSSENH 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 140-1, is Multi-Chip Standalone.

{ TC "SECURITY POLICY" \F SP }SECURITY POLICY

DSSENH operates under several rules that encapsulate its security policy.

- DSSENH is supported on Windows NT 4.0 with Service Pack 6 or later.
- DSSENH relies on Microsoft Windows NT 4.0 for the authentication of users.
- DSSENH enforces a single role, Authenticated User, which is a combination of the User and Cryptographic Officer roles as defined in FIPS PUB 140-1.
- All users authenticated by Microsoft Windows NT 4.0 employ the Authenticated User role.
- All services implemented within DSSENH are available to the Authenticated User role.
- Keys created within DSSENH by one user are not accessible to any other user via DSSENH.
- DSSENH relies on Microsoft Windows NT 4.0 for the secure storage of keys.
- DSSENH performs the following self-tests upon power up:
  - RC4 encrypt/decrypt
  - RC2 ECB encrypt/decrypt
  - DES ECB encrypt/decrypt
  - DES40 ECB encrypt/decrypt
  - 3DES 112 ECB encrypt/decrypt
  - 3DES ECB encrypt/decrypt
  - RC2 CBC encrypt/decrypt
  - DES CBC encrypt/decrypt
  - DES40 CBC encrypt/decrypt
  - 3DES 112 CBC encrypt/decrypt
  - 3DES CBC encrypt/decrypt
  - MD5 hash
  - SHA-1 hash
- DSSENH performs a pairwise consistency test upon each invocation of DSA key generation as defined in FIPS PUB 140-1 and FIPS PUB 186.

{ TC "SPECIFICATION OF ROLES" \F SP }SPECIFICATION OF ROLES

DSSENH combines the User and Cryptographic Officer roles (as defined in FIPS PUB 140-1) into a single role hereon called the Authenticated User role. The Authenticated User may access 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. Each user may have numerous keys, signature and key exchange, and these keys are separate from other users' keys.

Maintenance Roles

Maintenance roles are not supported by DSSENH.

Multiple Concurrent Operators

DSSENH is intended to run on Windows NT 4.0 with Service Pack 6 or later in Single User Mode. When run in this configuration, multiple concurrent operators are not supported.

{ TC "SPECIFICATION OF SERVICES" \F SP }SPECIFICATION OF SERVICES

The following list contains all services available to an operator. All services are accessible by all Authenticated Users, the one and only role supported by DSSENH.

Key Storage

DSSENH does not store keys. It delegates that task to the Protected Storage Service (PSTORE) of Microsoft Windows NT 4.0, a separate component of the operating system, and outside the boundaries of the cryptomodule.

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, from PSTORE, via the CSP 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.

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 container, enumerate supported algorithms, and generally determine capabilities of the CSP.

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

The following functions provide interfaces to the cryptomodule'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.

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. The underlying public/private key pair is not destroyed by this function. Only the handle is destroyed.

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 DSS/DH 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 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 DSS/DH 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 algorithm's may be used to encrypt the private key blob (DES, 3DES, DES40, RC4 or RC2).

Public DSS/DH keys are also exported using this function. A handle to the DSS/DH 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.

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.

Generation of a DSS key for signatures requires the operator to complete several steps before a DSS key is generated. CryptGenKey is first called with CRYPT_PREGEN set in the dwFlags parameter. The operator then sets the P, Q, and G for the key generation via CryptSetKeyParam, once for each parameter. The operator calls CryptSetKeyParam with KP_X set as dwParam to complete the key generation.

CryptGenRandom

The CryptGenRandom function fills a buffer with random bytes. The random number generation algorithm is the SHS based RNG from FIPS 186.

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.

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

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 CSP module and is referenced by the hKey parameter.

Hashing and Digital Signatures

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 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 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, 3DES, DES40, RC4 or RC2).

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.

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 DSS.

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 DSS signatures.

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.

{ TC "CRYPTOGRAPHIC KEY MANAGEMENT" \F SP }CRYPTOGRAPHIC KEY MANAGEMENT

The DSSENH cryptomodule manages keys in the following manner.

Key Material

DSSENH can create and use keys for the following algorithms: DSS, Diffie-Hellman, RC2, RC4, DES, DES40, and 3DES.

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. Keys can also be derived from known values via the CryptDeriveKey() function. DSS keys are generated and validated following the manner described in FIPS PUB 186-1. DES key are generated and validated following the manner described in FIPS PUB 46-2 and FIPS PUB 81.

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 DSSENH 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 (DES, 3DES, DES40, RC4 or RC2). When private keys are generated or imported from archival, they are outputted to the Microsoft Windows NT 4.0 PSTORE in a covered form.

Symmetric key entry and output is done by exchanging keys using the recipient’s asymmetric public 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.

Key Storage

DSSENH offloads the key storage operations to the Microsoft Windows NT 4.0 operating system. Keys are not stored in the cryptographic module, private keys are stored in the Microsoft Protected Storage System (PSTORE) service of the Microsoft Windows NT 4.0 in the manner described below. Keys are zeroized from memory after use.   Only the key used for power up self-testing is stored in the cryptographic module.

When an Authenticated User requests a keyed cryptographic operation from DSSENH his/her keys are retrieved from the Microsoft Protected Storage System (PSTORE) service of the Microsoft Windows NT 4.0. The PSTORE service, called via a local procedure call (LPC) from DSSENH, runs as Local System (analogous to the UNIX root account) and receives the caller's security token via the LPC call parameters. PSTORE then impersonates the user and retrieves the Security Identity Descriptor (SID) from the user token. PSTORE uses the SID to navigate to the following System Registry location:

```
HKEY_CURRENT_USER\Software\Microsoft\Protected Storage System Provider\<retrieved
user SID>Data\<data GUID>\
```

The <data GUID> is a PSTORE specific identifier that describes the type of data being protected. This System Registry location contains the private keys in a covered format. The System Registry folders containing cryptographic keys are additionally protected by Access Control Lists (ACLs) that limit access to only Local System or BUILTIN\Administrator. The PSTORE services, after retrieving the covered key, uncovers the key and returns it via LPC back to the caller.

Key Archival

DSSENH does not directly archive cryptographic keys. The Authenticated User 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 Authenticated User calls CryptDestroyKey on that key handle. Private keys (which are stored by the operating system in covered format in the protected storage system portion of the NT4.0 OS) are destroyed when the Authenticated User calls CryptAcquireContext with the CRYPT_DELETE_KEYSET flag.

{ TC "SELF-TESTS" \F SP }SELF-TESTS

Mandatory

Software tests via a DES MAC of library image

- RC4 encrypt/decrypt KAT
- RC2 ECB encrypt/decrypt KAT
- DES ECB encrypt/decrypt KAT
- DES40 ECB encrypt/decrypt KAT
- 3DES ECB encrypt/decrypt KAT
- 3DES 112 ECB encrypt/decrypt KAT
- RC2 CBC encrypt/decrypt KAT
- DES CBC encrypt/decrypt KAT
- DES40 CBC encrypt/decrypt KAT
- 3DES CBC encrypt/decrypt KAT
- 3DES 112 CBC encrypt/decrypt KAT
- MD5 hash KAT
- SHA-1 hash KAT
- DSS pairwise consistency test
- Diffie-Hellman pairwise consistency test

Conditional

The following are initiated at key generation:

- DSS pairwise consistency test
- Diffie-Hellman pairwise consistency test

{ TC "MISCELLANEOUS" \F SP }MISCELLANEOUS

The following items address requirements not addressed above.

Cryptographic Bypass

Cryptographic bypass is not support in DSSENH.

Operation Authentication

DSSENH inherits all authentication from the Microsoft Windows NT 4.0 operating system upon which it runs. Microsoft Windows NT 4.0 requires authentication from a trusted control 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 Authenticated User’s security token. All subsequent processes and threads created by that Authenticated User are implicitly assigned the parent’s (thus the Authenticated User’s) security token. Every user that has been authenticated by Microsoft Windows NT 4.0 is naturally assigned the Authenticated User role when he/she accesses DSSENH.

Identity-based Authentication

While all Authenticated Users are assigned the same role and thus have access to the same complete set of services, individual Authenticated Users may only access key containers which they themselves have created. DSSENH assumes the authentication of the user and enforces it by running in a thread with the Authenticated User’s security token.

Operating System Security

The DSSENH cryptomodule is intended to run on Windows NT 4.0 with Service Pack 6 or later in Single User Mode.

When an operating system process loads the cryptomodule into memory, the cryptomodule runs a DES MAC on the cryptomodule’s disk image of DSSENH.DLL, excluding the DES MAC, checksum, and export signature resources. This MAC is compared to the value stored in the DES MAC 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.

{ TC "FOR MORE INFORMATION" \F SP }FOR MORE INFORMATION

For the latest information on Windows NT Server, check out our World Wide Web site at http://www.microsoft.com/ntserver or the Windows NT Server Forum on the MSN™ network of Internet services (GO WORD: MSNTS).