Microsoft®

Windows NT®

Operating System

Microsoft Base Cryptographic Provider

FIPS 140-1 Documentation: Security Policy

September 20, 2000 11:29 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 2000, 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 2000.
- RSABASE relies on Microsoft Windows 2000 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 2000 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 stores keys in the file system, but relies on Microsoft Windows 2000 for the covering of the keys prior to storage.
- 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 2000 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 stores keys in the file system. The task of covering the keys prior to storage in the file system is delegated to the Data Protection API of Microsoft Windows 2000, a separate component of the operating system, and outside the boundaries of the cryptomodule. When a key container is deleted, the file is zeroized before being deleted.

CryptAcquireContext

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

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

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

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

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

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

CryptGetProvParam

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

CryptSetProvParam

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

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 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.

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.

CryptGetKeyParam

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

CryptGetUserKey

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

CryptImportKey

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

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

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

Symmetric keys wrapped with other symmetric keys may also be imported using this function. The wrapped key blob is passed in along with a handle to a symmetric key which the module is supposed to use to unwrap the blob. If the function is successful then a handle to the unwrapped symmetric key is returned.

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).

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.

If the hash handle references a CALG_SCHANNEL_MASTER_HASH key then when CryptDestroyHash is called the associated key material is zeroized.

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. 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.

{ 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 covered with the Microsoft Windows 2000 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.

Key Storage

RSABASE offloads the key storage operations to the Microsoft Windows 2000 operating system. Keys are not stored in the cryptographic module, private keys are encrypted by the Microsoft Data Protection API (DPAPI) service, and then stored in the Microsoft Windows 2000 file system. 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 file system.

DPAPI uses a two-phase algorithm for shrouding the Secret Key (SK) used to encrypt data. Phase 2 occurs by default only if there is a Domain Controller associated with the user. Therefore in the local user case, the SK is protected by a local LSA secret. SYSKEY should be enabled to prevent access to this key. Refer to NT4/win2k documentation for info on SYSKEY.

Phase 1: Local Agent

In the first phase, the system shrouds the secret locally, relying on the service run as Local System to protect secrets. This protection shrouds the data both as it travels on the wire and also blinds the data from the DC. Thus, the shrouding ensures that no remote user (even a “phase 2” remote recovery agent) can decrypt the data independent from the local system.

Recovery setup

1. Agent has data D1 to shroud
2. Agent uses secret key SK encrypt D1
3. Agent stores SK in the user hive ACLed to local agent
4. Agent has shrouded E{D1}

Initiate recovery

1. Agent has E{D1} to unshroud
2. Agent retrieves secret key SK from user hive
3. Agent uses secret key SK to decrypt E{D1}
4. Agent has unshrouded D1

Phase 2: Remote Agent

In the second phase, if the machine is networked, the shrouded secret is sent to the domain controller (DC) for an identification stamp and second shrouding. This second shrouding will ensure that a roaming user profile is not self-contained, but needs an interactive logon to successfully recover the master key.

Recovery setup

5. User sends data D2 to remote agent
6. Agent uses secret monster key K, random R2, HMACs to derive SymKeyM.
7. Use SymKeyM to MAC {userid, D2} -> m{userid, D2}

8. Agent uses secret monster key K, random R3, HMACs to derive SymKeyK.

9. Use SymKeyK to encrypt { m{userid, D2} , R2 }

10. Agent returns recovery field E{ m{userid, D2}, R2 }, R3 to User

11. User stores recovery field E{ m{userid, D2}, R2 }, R3

Initiate recovery

5. User sends recovery field E{ m{userid, D2}, R2 }, R3 to remote agent

6. Agent uses secret monster key K, HMACs with R3 to re-derive SymKeyK.

7. SymKeyK used to decrypt m{userid, D2}, R2

8. Agent uses secret monster key K, HMACs with R2 to re-derive SymKeyM.

9. SymKeyM used to check MAC on {userid, D2}.

10. Agent returns D2 if userid matches current recovery requestor.

These phases can be nested such that D2 = E{D1}, which allows neither of the agents to recover the data barring collusion.

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 2000 operating system upon which it runs. Microsoft Windows 2000 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 2000 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.

ModularExpOffload

The ModularExpOffload function offloads modular exponentiation from a CSP to a hardware accelerator. The CSP will check in the registry for the value HKLM\Software\Microsoft\Cryptography\ExpoOffload that can be the name of a DLL. The CSP uses LoadLibrary to load that DLL and calls GetProcAddress to get the OffloadModExpo entry point in the DLL specified in the registry. The CSP 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.

Operating System Security

The RSABASE cryptomodule is intended to run on Windows 2000 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 11:29 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 2000, 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 2000.
- DSSBASE relies on Microsoft Windows 2000 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 2000 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 stores keys in the file system, but relies on Microsoft Windows 2000 for the covering of the keys prior to storage.
- 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 2000 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 stores keys in the file system. The task of covering the keys prior to storage in the file system is delegated to the Data Protection API of Microsoft Windows 2000, a separate component of the operating system, and outside the boundaries of the cryptomodule. When a key container is deleted, the file is zeroized before being deleted.

CryptAcquireContext

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

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

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

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

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

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

CryptGetProvParam

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

CryptSetProvParam

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

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 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.

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.

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.

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.

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).

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.

If the hash handle references a CALG_SCHANNEL_MASTER_HASH key then when CryptDestroyHash is called the associated key material is zeroized.

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 covered with the Microsoft Windows 2000 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.

Key Storage

DSSBASE offloads the key storage operations to the Microsoft Windows 2000 operating system. Keys are not stored in the cryptographic module, private keys are encrypted by the Microsoft Data Protection API (DPAPI) service, and then stored in the Microsoft Windows 2000 file system. 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 file system.

DPAPI uses a two-phase algorithm for shrouding the Secret Key (SK) used to encrypt data. Phase 2 occurs by default only if there is a Domain Controller associated with the user. Therefore in the local user case, the SK is protected by a local LSA secret. SYSKEY should be enabled to prevent access to this key. Refer to NT4/win2k documentation for info on SYSKEY.

Phase 1: Local Agent

In the first phase, the system shrouds the secret locally, relying on the service run as Local System to protect secrets. This protection shrouds the data both as it travels on the wire and also blinds the data from the DC. Thus, the shrouding ensures that no remote user (even a “phase 2” remote recovery agent) can decrypt the data independent from the local system.

Recovery setup

1. Agent has data D1 to shroud
2. Agent uses secret key SK encrypt D1
3. Agent stores SK in the user hive ACLed to local agent
4. Agent has shrouded E{D1}

Initiate recovery

1. Agent has E{D1} to unshroud
2. Agent retrieves secret key SK from user hive
3. Agent uses secret key SK to decrypt E{D1}
4. Agent has unshrouded D1

Phase 2: Remote Agent

In the second phase, if the machine is networked, the shrouded secret is sent to the domain controller (DC) for an identification stamp and second shrouding. This second shrouding will ensure that a roaming user profile is not self-contained, but needs an interactive logon to successfully recover the master key.

Recovery setup

5. User sends data D2 to remote agent

6. Agent uses secret monster key K, random R2, HMACs to derive SymKeyM.

7. Use SymKeyM to MAC {userid, D2} -> m{userid, D2}

8. Agent uses secret monster key K, random R3, HMACs to derive SymKeyK.

9. Use SymKeyK to encrypt { m{userid, D2} , R2 }

10. Agent returns recovery field E{ m{userid, D2}, R2 }, R3 to User

11. User stores recovery field E{ m{userid, D2}, R2 }, R3

Initiate recovery

5. User sends recovery field E{ m{userid, D2}, R2 }, R3 to remote agent

6. Agent uses secret monster key K, HMACs with R3 to re-derive SymKeyK.

7. SymKeyK used to decrypt m{userid, D2}, R2

8. Agent uses secret monster key K, HMACs with R2 to re-derive SymKeyM.

9. SymKeyM used to check MAC on {userid, D2}.

10. Agent returns D2 if userid matches current recovery requestor.

These phases can be nested such that D2 = E{D1}, which allows neither of the agents to recover the data barring collusion.

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 2000 operating system upon which it runs. Microsoft Windows 2000 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 2000 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.

ModularExpOffload

The ModularExpOffload function offloads modular exponentiation from a CSP to a hardware accelerator. The CSP will check in the registry for the value HKLM\Software\Microsoft\Cryptography\ExpoOffload that can be the name of a DLL. The CSP uses LoadLibrary to load that DLL and calls GetProcAddress to get the OffloadModExpo entry point in the DLL specified in the registry. The CSP 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.

Operating System Security

The DSSBASE cryptomodule is intended to run on Windows 2000 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 11:29 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 2000, 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 2000.
- RSAENH relies on Microsoft Windows 2000 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 2000 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 stores keys in the file system, but relies on Microsoft Windows 2000 for the covering of the keys prior to storage.
- 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 2000 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 stores keys in the file system. The task of covering the keys prior to storage in the file system is delegated to the Data Protection API of Microsoft Windows 2000, a separate component of the operating system, and outside the boundaries of the cryptomodule. When a key container is deleted, the file is zeroized before being deleted.

CryptAcquireContext

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

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

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

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

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

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

CryptGetProvParam

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

CryptSetProvParam

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

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 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.

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.

CryptGetKeyParam

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

CryptGetUserKey

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

CryptImportKey

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

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

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

Symmetric keys wrapped with other symmetric keys may also be imported using this function. The wrapped key blob is passed in along with a handle to a symmetric key which the module is supposed to use to unwrap the blob. If the function is successful then a handle to the unwrapped symmetric key is returned.

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, RC4 or RC2).

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.

If the hash handle references a CALG_SCHANNEL_MASTER_HASH key then when CryptDestroyHash is called the associated key material is zeroized.

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. 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.

{ 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 covered with the Microsoft Windows 2000 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.

Key Storage

RSAENH offloads the key storage operations to the Microsoft Windows 2000 operating system. Keys are not stored in the cryptographic module, private keys are encrypted by the Microsoft Data Protection API (DPAPI) service, and then stored in the Microsoft Windows 2000 file system. 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 file system.

DPAPI uses a two-phase algorithm for shrouding the Secret Key (SK) used to encrypt data. Phase 2 occurs by default only if there is a Domain Controller associated with the user. Therefore in the local user case, the SK is protected by a local LSA secret. SYSKEY should be enabled to prevent access to this key. Refer to NT4/win2k documentation for info on SYSKEY.

Phase 1: Local Agent

In the first phase, the system shrouds the secret locally, relying on the service run as Local System to protect secrets. This protection shrouds the data both as it travels on the wire and also blinds the data from the DC. Thus, the shrouding ensures that no remote user (even a “phase 2” remote recovery agent) can decrypt the data independent from the local system.

Recovery setup

1. Agent has data D1 to shroud
2. Agent uses secret key SK encrypt D1
3. Agent stores SK in the user hive ACLed to local agent
4. Agent has shrouded E{D1}

Initiate recovery

1. Agent has E{D1} to unshroud
2. Agent retrieves secret key SK from user hive
3. Agent uses secret key SK to decrypt E{D1}
4. Agent has unshrouded D1

Phase 2: Remote Agent

In the second phase, if the machine is networked, the shrouded secret is sent to the domain controller (DC) for an identification stamp and second shrouding. This second shrouding will ensure that a roaming user profile is not self-contained, but needs an interactive logon to successfully recover the master key.

Recovery setup

5. User sends data D2 to remote agent

6. Agent uses secret monster key K, random R2, HMACs to derive SymKeyM.

7. Use SymKeyM to MAC {userid, D2} -> m{userid, D2}

8. Agent uses secret monster key K, random R3, HMACs to derive SymKeyK.

9. Use SymKeyK to encrypt { m{userid, D2} , R2 }

10. Agent returns recovery field E{ m{userid, D2}, R2 }, R3 to User

11. User stores recovery field E{ m{userid, D2}, R2 }, R3

Initiate recovery

5. User sends recovery field E{ m{userid, D2}, R2 }, R3 to remote agent

6. Agent uses secret monster key K, HMACs with R3 to re-derive SymKeyK.

7. SymKeyK used to decrypt m{userid, D2}, R2

8. Agent uses secret monster key K, HMACs with R2 to re-derive SymKeyM.

9. SymKeyM used to check MAC on {userid, D2}.

10. Agent returns D2 if userid matches current recovery requestor.

These phases can be nested such that D2 = E{D1}, which allows neither of the agents to recover the data barring collusion.

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 2000 operating system upon which it runs. Microsoft Windows 2000 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 2000 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.

ModularExpOffload

The ModularExpOffload function offloads modular exponentiation from a CSP to a hardware accelerator. The CSP will check in the registry for the value HKLM\Software\Microsoft\Cryptography\ExpoOffload that can be the name of a DLL. The CSP uses LoadLibrary to load that DLL and calls GetProcAddress to get the OffloadModExpo entry point in the DLL specified in the registry. The CSP 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.

Operating System Security

The RSAENH cryptomodule is intended to run on Windows 2000 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)

Microsoft®

Windows NT®

Operating System

Microsoft DSS/Diffie-Hellman Enhanced Cryptographic Provider

FIPS 140-1 Documentation: Security Policy

September 20, 2000 11:29 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 2000, 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 2000.
- DSSENH relies on Microsoft Windows 2000 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 2000 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 stores keys in the file system, but relies on Microsoft Windows 2000 for the covering of the keys prior to storage.
- 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 2000 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 stores keys in the file system. The task of covering the keys prior to storage in the file system is delegated to the Data Protection API of Microsoft Windows 2000, a separate component of the operating system, and outside the boundaries of the cryptomodule. When a key container is deleted, the file is zeroized before being deleted.

CryptAcquireContext

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

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

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

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

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

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

CryptGetProvParam

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

CryptSetProvParam

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

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 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.

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.

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.

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.

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).

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.

If the hash handle references a CALG_SCHANNEL_MASTER_HASH key then when CryptDestroyHash is called the associated key material is zeroized.

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 covered with the Microsoft Windows 2000 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.

Key Storage

DSSENH offloads the key storage operations to the Microsoft Windows 2000 operating system. Keys are not stored in the cryptographic module, private keys are encrypted by the Microsoft Data Protection API (DPAPI) service, and then stored in the Microsoft Windows 2000 file system. 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 file system.

DPAPI uses a two-phase algorithm for shrouding the Secret Key (SK) used to encrypt data. Phase 2 occurs by default only if there is a Domain Controller associated with the user. Therefore in the local user case, the SK is protected by a local LSA secret. SYSKEY should be enabled to prevent access to this key. Refer to NT4/win2k documentation for info on SYSKEY.

Phase 1: Local Agent

In the first phase, the system shrouds the secret locally, relying on the service run as Local System to protect secrets. This protection shrouds the data both as it travels on the wire and also blinds the data from the DC. Thus, the shrouding ensures that no remote user (even a “phase 2” remote recovery agent) can decrypt the data independent from the local system.

Recovery setup

1. Agent has data D1 to shroud
2. Agent uses secret key SK encrypt D1
3. Agent stores SK in the user hive ACLed to local agent
4. Agent has shrouded E{D1}

Initiate recovery

1. Agent has E{D1} to unshroud
2. Agent retrieves secret key SK from user hive
3. Agent uses secret key SK to decrypt E{D1}
4. Agent has unshrouded D1

Phase 2: Remote Agent

In the second phase, if the machine is networked, the shrouded secret is sent to the domain controller (DC) for an identification stamp and second shrouding. This second shrouding will ensure that a roaming user profile is not self-contained, but needs an interactive logon to successfully recover the master key.

Recovery setup

5. User sends data D2 to remote agent

6. Agent uses secret monster key K, random R2, HMACs to derive SymKeyM.

7. Use SymKeyM to MAC {userid, D2} -> m{userid, D2}

8. Agent uses secret monster key K, random R3, HMACs to derive SymKeyK.

9. Use SymKeyK to encrypt { m{userid, D2} , R2 }

10. Agent returns recovery field E{ m{userid, D2}, R2 }, R3 to User

11. User stores recovery field E{ m{userid, D2}, R2 }, R3

Initiate recovery

5. User sends recovery field E{ m{userid, D2}, R2 }, R3 to remote agent

6. Agent uses secret monster key K, HMACs with R3 to re-derive SymKeyK.

7. SymKeyK used to decrypt m{userid, D2}, R2

8. Agent uses secret monster key K, HMACs with R2 to re-derive SymKeyM.

9. SymKeyM used to check MAC on {userid, D2}.

10. Agent returns D2 if userid matches current recovery requestor.

These phases can be nested such that D2 = E{D1}, which allows neither of the agents to recover the data barring collusion.

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 2000 operating system upon which it runs. Microsoft Windows 2000 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 2000 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.

ModularExpOffload

The ModularExpOffload function offloads modular exponentiation from a CSP to a hardware accelerator. The CSP will check in the registry for the value HKLM\Software\Microsoft\Cryptography\ExpoOffload that can be the name of a DLL. The CSP uses LoadLibrary to load that DLL and calls GetProcAddress to get the OffloadModExpo entry point in the DLL specified in the registry. The CSP 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.

Operating System Security

The DSSENH cryptomodule is intended to run on Windows 2000 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)