Microsoft®

Microsoft Outlook®

Microsoft Outlook Cryptographic Provider

FIPS 140-1 Documentation: Security Policy

September 20, 2000 11:26 AM

Abstract

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

CONTENTS

{ TC "INTRODUCTION" \F SP }INTRODUCTION

Microsoft Outlook Cryptographic Provider (EXCHCSP) is a FIPS 140-1 Level 1 compliant, general-purpose, software-based, cryptographic module. Like the cryptographic providers that ship with Microsoft Windows 2000, EXCHCSP 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 Outlook Cryptographic Provider (EXCHCSP) consists of a single dynamically-linked library (DLL) named EXCHCSP.DLL. The cryptographic boundary for EXCHCSP 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

EXCHCSP operates under several rules that encapsulate its security policy.

- EXCHCSP is supported on Windows 2000.
- EXCHCSP relies on Microsoft Windows 2000 for the authentication of users.
- EXCHCSP 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 EXCHCSP are available to the Authenticated User role.
- Keys created within EXCHCSP by one user are not accessible to any other user via EXCHCSP.
- EXCHCSP stores keys in the Microsoft Windows 2000 registry.
- EXCHCSP performs the following self-tests upon power up:
  - RC2 ECB encrypt/decrypt
  - DES ECB encrypt/decrypt
  - 3DES ECB encrypt/decrypt
  - RC2 CBC encrypt/decrypt
  - DES CBC encrypt/decrypt
  - 3DES CBC encrypt/decrypt
  - MD2 hash
  - MD5 hash
  - SHA-1 hash
  - RSA self-test
- EXCHCSP performs a pair wise 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

EXCHCSP 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 EXCHCSP.

Multiple Concurrent Operators

EXCHCSP 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 EXCHCSP.

Key Storage

EXCHCSP stores keys in the system registry. EXCHCSP covers the keys using a user-supplied password prior to storing them in the system registry. When a key container is deleted, the registry entries are 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.

Unlike most CSPs, new key containers cannot be created or destroyed using the CryptAcquireContext function. Creation of new key containers can only be done by a) enrollment is performed only through the Microsoft Exchange Key Management Server or b) importing an Exchange Protection File (EPF) saved file.

Key containers can only be deleted during the export of a key container to an EPF file.

CryptGetProvParam

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

CryptSetProvParam

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

CryptReleaseContext

The CryptReleaseContext function releases the handle referenced by the *hProv* parameter. After a provider handle has been released, it becomes invalid and cannot be used again. In addition, key and hash handles associated with that provider handle may not be used after CryptReleaseContext has been called.

BuildEnrollmentMessage

The BuildEnrollmentMessage function is one of two ways to create key containers in EXCHCSP. The function builds the enrollment message that is to be sent to the Microsoft Exchange Key Management Server (KMS). This function takes as parameters:

- hWnd - a handle to a window for dialogs to display against.
- pdata -- a structure containing data needed for the process. Some of the contents of the data includes:
  - The X500 Distinguished Name for the mailbox.
  - The enrollment token supplied to the user by the Microsoft Exchange KMS.
  - The name to be used for the key container to be created
  - The display name to be used for the mailbox
  - The RFC822 email name to be used for the mailbox.
- fOverWrite -- Should an existing key container be overwriting?
- ppbData -- location to return the string containing the enrollment message. Message is allocated with LocalAlloc.

A key container is then created with the requested name (a flag exists if an already existing container of the same name is to be deleted and a new one created). A new signing key is then created for the user. The DN, RFC822 name, display name and signing key are then used to create a PKCS#10 certificate request. The message to be transmitted to the Microsoft Exchange KMS is then created. This contains the certificate request and is authenticated using the supplied enrollment token and the mailbox DN.

The enrollment request is returned to the calling function for transmission to the Microsoft Exchange KMS. The function returns S_OK (0) on success and a non-zero error code on failure.

ProcessEnrollmentMessage

The ProcessEnrollmentMessage function is used to finish the enrollment process. This function takes as inputs the response message from the Microsoft Exchange Key Management Server (KMS) and the name of the keyset created in the BuildEnrollmentMessage function. The contents of the enrollment message are parsed and stored in the keyset. The parameters for this function are:

- hWnd - a window handle to bring up dialogs against.
- pdata - a structure of data about operation. This structure includes:
  - The response message from the enrollment message
  - Return location for the end-entity and root certificates.
  - Keyset Name

- ppsz - return location for the acknowledgement message is one is produced by the protocol. The message is a string allocated by LocalAlloc.

The enrollment response message includes the following elements:

- Private key exchange keys generated on the server
- Certificates for key exchange keys
- Certificate for the signing key in the request message.
- Root certificates for trust anchor points

The private keys are protected using an RC2/128-bit key. The public signing key transported in the enrollment request message is used as a key-exchange key to encrypt and return the content encryption key. The function will return a conformation message to be sent to the KMS for some of the enrollment protocols and an error code of S_OK (0) for success and non-zero for failure.

BuildRenewalMessage

EXCHCSP has built in the ability to do automatic turn over of key material, allow for more than one key exchange key to be kept in a key container. The BuildRenewalMessage function allows for the CSP to get a new set of key exchange keys from the Microsoft Exchange Key Management Server. The function builds an enrollment message that is sent to the Microsoft Exchange Key Management Server

The parameters for the function are:

- hWnd - The window that all dialogs should be modal to.
- pData - A structure containing parameters about the operation. This structure includes:
    - The key set name
    - The mailbox display name
    - The mailbox RFC822 name
    - The mailbox distinguished name
- fOverwirte - Overwrite any existing renewal operation currently in progress
- ppbData - Return location for the renewal message string. The string is allocated using LocalAlloc.

The function creates a new signing key, creates a certificate request for the signing key and signs the request using the current signing key. This provides for authentication of the packet based on existing data. The function returns S_OK (0) on success and an error code (non-zero) on failure. The caller then sends the returned renewal message to the Microsoft Exchange Key Management Server and when the response is obtained calls ProcessEnrollmentMessage function passing in the response message.

UpgradeEpfToPStore

The UpgradeEpfToPStore function is used to restore a previously saved set of keys and create a new key container. EPF (Exchange Private key Files) provide for users to archive their own key material and to transport keys between different machines. The function reads the keys from the file and save them into a key container in the CSP. The file is not deleted on completion of the operation. The function takes the following parameters:

- szEpfFileName - Contains the name of the EPF File for the keys to be imported into a key set. The file contains the entire history of key exchange keys and the current signing key along with all relevant certificates.
- szPassword - Contains the password for access to the EPF file.
- szKeySetName - The name of the key set to be created when importing the key material.
- dwFlags - not used
- hwnd - The window to bring all dialogs up modal with.
- fOverWrite - Overwrite any existing key set with the same name.
- rgpccert - Returns the current end-entity certificates and root certificates.
- pfDomestic - returns if this is the North American version of the EXCHCSP. This is used by the to determine the set of symmetric algorithms supported.

The function returns S_OK on success and non-zero on failure.

PStoreToEPF

The PStoreToEPF function is used to save the private key material (and related certificates to a disk file. A key derived from a user-supplied password protects the saved file. This function provides the only method to delete a key container.

The parameters to this function are:

- szFileName - the name of the EPF file to be created.
- szPassword - the password to be used in protecting the contents of the EPF file.
- szKeySetName - the name of the key set to be saved into the EPF file.
- dwFlags -
    - 2 - Overwrite the EFP file if it already exists.
    - 1 - Delete the key set after it has been saved to the EPF file.
- hwnd - the handle to the window that all dialogs are to be made modal to.

The function returns S_OK on success and non-zero on failure.

ChangeStorePassword

The ChangeStorePassword function provides the ability to change the password used to protect the private key material in a key container. The function brings up a dialog box to get the old and new password for the key container. The function takes as parameters:

- hWnd - the window to make the dialog box modal against.
- wszKeySet - The name of the key set to change the password for.

The function returns S_OK (0) on success and non-zero on failure.

Key Generation and Exchange

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

CryptDeriveKey

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

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

CryptDestroyKey

The CryptDestroyKey function releases the handle referenced by the *hKey* parameter. After a key handle has been released, it becomes invalid and cannot be used again.

If the handle refers to a session key, or to a public key that has been imported into the CSP through CryptImportKey, this function zeroizes the key in memory and frees the memory that the key occupied. The underlying public/private key pair is not destroyed by this function. Only the handle is destroyed.

CryptExportKey

The CryptExportKey function exports cryptographic keys from a cryptographic service provider (CSP) in a secure manner for key archival purposes.

EXCHCSP does not permit the private portion of an RSA key to be exported except via the PStoreToEPF function.

Public RSA keys are 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 SHA-1 based RNG from FIPS 186-1 Appendix 3.1.

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. Imported private keys cannot be persisted to permanent key storage.

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

Symmetric keys wrapped with other symmetric keys may also be imported using this function. The wrapped key blob is passed in along with a handle to a symmetric key that 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. Two keyed hash algorithms; MAC and HMAC are also created using this function. When creating a MAC hash object, a symmetric key object is passed in. The symmetric key object may be any of the symmetric block ciphers supported by the module (DES, 3DES, RC2).

CryptDestroyHash

The CryptDestroyHash function destroys the hash object referenced by the *hHash* parameter. After a hash object has been destroyed, it can no longer be used.

All hash objects should be destroyed with the CryptDestroyHash function when the application is finished with them.

CryptGetHashParam

The CryptGetHashParam function retrieves data that governs the operations of a hash object. The actual hash value can also be retrieved by using this function.

CryptHashData

The CryptHashData function adds data to a specified hash object. This function and CryptHashSessionKey can be called multiple times to compute the hash on long data streams or discontinuous data streams. Before calling this function, the CryptCreateHash function must be called to create a handle of a hash object.

CryptHashSessionKey

The CryptHashSessionKey function computes the cryptographic hash of a key object. This function can be called multiple times with the same hash handle to compute the hash of multiple keys. Calls to CryptHashSessionKey can be interspersed with calls to CryptHashData. Before calling this function, the CryptCreateHash function must be called to create the handle of a hash object.

CryptSetHashParam

The CryptSetHashParam function customizes the operations of a hash object.

CryptSignHash

The CryptSignHash function signs data. Because all signature algorithms are asymmetric and thus slow, the CryptoAPI does not allow data be signed directly. Instead, data is first hashed and CryptSignHash is used to sign the hash. The crypto module supports signing with RSA

After this function has been completed, only CryptDestroyHash can be called using the hHash handle.

CryptVerifySignature

The CryptVerifySignature function verifies the signature of a hash object. Before calling this function, the CryptCreateHash function must be called to create the handle of a hash object. CryptHashData or CryptHashSessionKey is then used to add data or session keys to the hash object. The crypto module supports verifying RSA signatures

After this function has been completed, only CryptDestroyHash can be called using the hHash handle.

CryptDuplicateHash

The CryptDuplicateHash function is used to duplicate, make a copy of, the state of a hash and returns a handle to this new hash. The CryptDestroyHash function must be used on both the handle to the original hash and the newly duplicated hash.

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

The EXCHCSP cryptomodule manages keys in the following manner.

Key Material

EXCHCSP can create and use keys for the following algorithms: RSA Signature, RSA Key Exchange, RC2, 3DES, 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. Asymmetric keys cannot be generated using this function call.

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

Symmetric keys can be exported from EXCHCSP via CryptExportKey. Symmetric keys can be imported into EXCHCSP via CryptImportKey().

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.

The public portion of asymmetric keys can be exported from EXCHCSP via CryptExportKey and imported via CryptImportKey. The private portion key asymmetric keys can only be exported via the PStoreToEPF function and imported via the UpgradeEPFToPStore function. These functions move the asymmetric keys into/out of a file format where the private portion of the asymmetric key is shrouded by a user-supplied password. No provisions exist for exporting the private portion of an asymmetric key in the clear.

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

EXCHCSP offloads the key storage operations to the Microsoft Windows 2000 system registry. Keys are not stored in the cryptographic module; private keys are encrypted by the a 128-bit RC2 key derived from a user supplied password, and then stored in the Microsoft Windows 2000 system registry. Keys are zeroized from memory after use. Only the keys used for power up self-testing is stored in the cryptographic module.

When an Authenticated User requests a keyed cryptographic operation from EXCHCSP his/her keys are retrieved from the system registry.

EXCHCSP uses the following algorithm for shrouding the private key material.

The following process derives the key used to shroud the data:

1. A salt value generated at creation time is appended to the user's supplied pass phrase.
2. The value is hashed using SHA1.
3. The output is padded to 512 bits and then hashed by SHA1.
4. Step 3 is repeated 99 times (resulting in 100 hash operations).
5. The first 128-bits of the resultant value is the RC2 key used. The 64-bit initialization vector is derived in a similar manner.
6. The key and IV are used to encrypt an array of eight zero bytes and the value is compared to a known saved value in the message. If this matches then the correct key is assumed to be derived and the correct pass phrase supplied.

Each of the private key values to be shrouded is encoded in ASN.1. The values are then appended together, compressed using a variant of Huffman encoding. After compression, the result is encrypted and stored in the system registry.

Key Archival

EXCHCSP supports the archival of key encryption keys, but does not directly support archival of signing keys. The Microsoft Exchange Key Management server archives all key exchange keys when it generates those keys during the enrollment process. A new signing key is always generated during the enrollment process. Additionally, Authenticated User may choose to export their cryptographic keys (key encryption and signing) into a disk file where the key material is protected by a user-supplied passphrase (see PStoreToEPF).

Key Destruction

All ephemeral keys are destroyed and their memory location zeroized when the Authenticated User calls CryptDestroyKey on that key handle. Keys stored in a keyset can only be removed and destroyed from the system by calling PStoreToEPF and passing in the delete flag, or by directly deleting the item in the registry.

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

Mandatory

Software tests via a DES MAC of library image

- 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 EXCHCSP.

Operation Authentication

EXCHCSP 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 EXCHCSP.

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 that they themselves have created. EXCHCSP assumes the authentication of the user and enforces it by running in a thread with the Authenticated User's security token.

OffloadModExpo

No offloading of modular exponentation from the CSP to a hardware accelerator is provided for.

Operating System Security

The EXCHCSP 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 EXCHCSP.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 Microsoft Outlook, check out our World Wide Web site at http://www.microsoft.com/outlook.