



Operating System

Microsoft DSS/Diffie-Hellman Enhanced Cryptographic Provider

FIPS 140-1 Documentation: Security Policy

6/28/2002 5:00 PM

Abstract

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

CONTENTS

INTRODUCTION	1
SECURITY POLICY.....	2
SPECIFICATION OF ROLES.....	3
SPECIFICATION OF SERVICES.....	4
CRYPTOGRAPHIC KEY MANAGEMENT.....	11
SELF-TESTS.....	15
MISCELLANEOUS.....	16
FOR MORE INFORMATION	18

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 XP, DSSENH encapsulates several different cryptographic algorithms in an easy-to-use cryptographic module accessible via the Microsoft CryptoAPI. Software developers can dynamically link the Microsoft DSSENH module into their applications to provide FIPS 140-1 compliant cryptographic support.

Windows XP does not ship the previously FIPS-140-1 validated Microsoft Base DSS and Diffie-Hellman Cryptographic Provider (DSSBASE.DLL) anymore. There is no loss of functionality as the DSSENH functionality has always been a subset of the DSSBASE functionality.

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. It should be noted that the Data Protection API of Microsoft Windows XP is not part of the module and should be considered to be outside the boundary.

SECURITY POLICY

DSENH operates under several rules that encapsulate its security policy.

- DSENH is supported on Windows XP .
- DSENH provides no user authentication; however, it relies on Microsoft Windows XP for the authentication of users.
- DSENH 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 XP employ the Authenticated User role.
- All services provided by the DSENH.DLL are available to the Authenticated User role.
- Keys created within DSENH by one user are not accessible to any other user via DSENH.
- DSENH stores keys in the file system, but relies on Microsoft Windows XP for the encryption of the keys prior to storage.
- DSENH supports the following FIPS-approved algorithms: DES, 3DES, SHA-1, and DSA.
- DSENH supports the following FIPS allowed algorithms: Diffie-Hellman
- DSENH supports the following non-FIPS approved algorithms: RC4, RC2, and MD5¹.
- DSENH 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
 - DSA pairwise consistency test
 - Diffie-Hellman pairwise consistency test
- DSENH performs a pair-wise consistency test upon each invocation of DSA key generation as defined in FIPS PUB 140-1 and FIPS PUB 186.

¹ Applications may not use any of these non-FIPS algorithms if they need to be FIPS compliant. To operate the module in a FIPS compliant manner, applications must only use FIPS-approved algorithms.

SPECIFICATION OF ROLES

DSENH combines the User and Cryptographic Officer roles (as defined in FIPS PUB 140-1) into a single role called the Authenticated User role. The Authenticated User may access all services implemented in the cryptographic module.

When an application requests the crypto module to generate keys for a user, the keys are generated, used, and deleted as requested by applications. There are no implicit keys associated with a user, and each user may have numerous keys, both signature and key exchange, and these keys are separate from other users' keys.

Maintenance Roles

Maintenance roles are not supported by DSENH.

Multiple Concurrent Operators

DSENH is intended to run on Windows XP in Single User Mode. When run in this configuration, multiple concurrent operators are not supported.

Because the module is a DLL, each process requesting access is provided its own instance of the module. As such, each process has full access to all information and keys within the module. Note that no keys or other information are maintained upon detachment from the DLL, thus an instantiation of the module will only contain keys or information that the process has placed in the module.

Data Access

Because an operator acting in the Authenticated User role is provided a separate instance of the module (a separate instantiation of the DLL), the Authenticated User role has complete access to all of the security data items within the module.

SPECIFICATION OF SERVICES

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

Key Storage Services

The following functions provide interfaces to the cryptomodule's key container functions. Please see the Key Storage description under the Cryptographic Key Management section for more information.

CryptAcquireContext

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

This function performs two operations. It first attempts to find a 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 Services

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

CryptDeriveKey

The CryptDeriveKey function creates cryptographic session keys derived from a hash value. This function guarantees that when the same CSP and algorithms are used, the keys created from the same hash value are identical. The hash value is typically a cryptographic hash (SHA-1 must be used when operating in FIPS-mode) of a password or similar secret user data.

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

If keys are being derived from a CALG_SCHANNEL_MASTER_HASH, then the appropriate key derivation process is used to derive the key. In this case the process used is from either the SSL 2.0, SSL 3.0, PCT or TLS specification of deriving client and server side encryption and MAC keys. This function will cause the key block to be derived from the master secret and the requested key is then derived from the key block. Which process is used is determined by which protocol is associated with the hash object. For more information see the SSL 2.0, SSL 3.0, PCT and TLS specifications.

CryptDestroyKey

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

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

`CryptExportKey`

The `CryptExportKey` function exports cryptographic keys from a cryptographic service provider (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 algorithms may be used to encrypt the private key blob (DES, 3DES, DES40, RC4 or RC2²).

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.

² Note that RC2 and RC4 may not be used while operating DSSSENH in a FIPS compliant manner.

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 Services

The following functions provide interfaces to the cryptomodule's data encryption and decryption functions.

CryptDecrypt

The CryptDecrypt function decrypts data previously encrypted using CryptEncrypt function.

CryptEncrypt

The CryptEncrypt function encrypts data. The algorithm used to encrypt the data is designated by the key held by the CSP module and is referenced by the hKey parameter.

Hashing and Digital Signatures Services

The following functions provide interfaces to the cryptomodule's hashing and digital signature functions.

CryptCreateHash

The CryptCreateHash function initiates the hashing of a stream of data. It returns to the calling application a handle to a 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, and 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. When a hash object is destroyed, the crypto module zeroizes the memory within the module where the hash object was held. The memory is then freed.

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

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

CryptGetHashParam

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

CryptHashData

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

CryptHashSessionKey

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

CryptSetHashParam

The CryptSetHashParam function customizes the operations of a hash object.

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.

CRYPTOGRAPHIC KEY MANAGEMENT

The DSSSENH cryptomodule manages keys in the following manner.

Key Material

DSSSENH can create and use keys for the following algorithms: DSS, Diffie-Hellman, RC2, RC4, DES, DES40, and 3DES. Each time an application links with DSSSENH, the DLL is instantiated and no keys exist within. The user application is responsible for importing keys into DSSSENH or using DSSSENH's functions to generate keys.

See MSDN Library\Platform SDK\Windows Base Services\Security\CryptoAPI 2.0\CryptoAPI Reference\CryptoAPI Structures\Cryptography Structures for more information about key formats and structures.

Key Generation

Random keys can be generated by calling the `CryptGenKey()` function. Keys can also be created from known values via the `CryptDeriveKey()` function. Keys are generated following the techniques given in FIPS PUB 186-2, Appendix 3, Random Number Generation.

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 DSSSENH 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/protected with the Microsoft Windows 2000 Data Protection API (DPAPI) and then outputted to the file system in the covered/protected 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

DSSSENH does not provide persistent storage of keys. While it is possible to store keys in the file system, this functionality is outside the scope of this validation. The task of protecting (or encrypting) the keys prior to storage in the file system is delegated to the Data Protection API (DPAPI) of Microsoft Windows XP. The DPAPI is a separate component of the operating system that is outside the boundaries of the cryptomodule but relies upon DSSSENH for all cryptographic functionality. This section describes this functionality for information purposes only.

When a key container is deleted, the file is zeroized before being deleted. DSSSENH offloads the key storage operations to the Microsoft Windows XP operating system, which is outside the cryptographic boundary. Because keys are not persistently stored inside the cryptographic module, private keys are instead encrypted by the Microsoft Data Protection API (DPAPI) service, and then stored in the Microsoft Windows XP file system. Keys are zeroized from memory after use. As an exception, the key used for power up self-testing is stored in the cryptographic module.

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

If the MasterKeyLegacyCompliance registry key value is set to the non-default non-zero value, Windows XP DPAPI uses a two-phase algorithm for encrypting the Secret Key (SK) used to encrypt data. Therefore in the local user case of the MasterKeyLegacyCompliance mode, 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. If there is a Windows 2000 Domain Controller associated with the user, then Phase 2 occurs by default regardless of the MasterKeyLegacyCompliance.

MasterKeyLegacyCompliance Phase 1: Local Agent

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

MasterKeyLegacyCompliance Recovery setup

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

MasterKeyLegacyCompliance Initiate recovery

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

Phase 2: Remote Agent

In the second phase, the encrypted secret is sent from the networked Windows XP machine to a Windows 2000 domain controller (DC) for an identification stamp and second encryption, if another Windows DC which is more version-compatible with the Windows XP machine is not available. This second encryption will ensure that a roaming user profile is not self-contained, but needs an interactive logon to successfully recover the master key.

Recovery setup with the cooperation of a Windows 2000 DC

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} $\rightarrow m\{userid, D2\}$
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 with the cooperation of a Windows 2000 DC

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.

By default, Windows XP DPAPI does not run in the MasterKeyLegacyCompliance mode. The Windows XP SK protection does not depend on a LSA secret. As in the case Windows 2000 DPAPI, Windows XP DPAPI uses a hash of the user's logon password to protect the SK. Windows XP has the local user account logon password backup and recovery support to address the unlikely situation where the user forgets his/her password and therefore is unable to gain access to the SK. The Windows XP local user account logon password backup and recovery support allows a local user to use a public key to encrypt the user's logon password hash. The private key corresponding to the public key is stored off-line on a floppy disk in a physically secure manner. During the logon password recovery, Windows XP uses the private key to recover the forgotten password and the SK, while asking the user to supply a new password for logon password resetting and re-encrypting the SK.

Key Archival

DSENH 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 that reside outside the cryptographic boundary (ones stored by the operating system in encrypted format in the Windows XP DPAPI system portion of the OS) are destroyed when the Authenticated User calls CryptAcquireContext with the CRYPT_DELETE_KEYSET flag.

SELF-TESTS

Power up

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

MISCELLANEOUS

The following items address requirements not addressed above.

Cryptographic Bypass

Cryptographic bypass is not support in DSSSENH.

Operator Authentication

DSSSENH provides no authentication of operators. However, the Microsoft Windows XP operating system upon which it runs does provide authentication, but this is outside the scope of DSSSENH's FIPS validation. The information about the authentication provided by Microsoft Windows XP is for informational purposes only. Microsoft Windows XP requires authentication from a trusted computer base (TCB³) 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 XP is naturally assigned the Authenticated User role when he/she accesses DSSSENH.

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. Note that to use DSSSENH in a FIPS compliant manner, this function should only be used if the hardware accelerator is FIPS validated.

Operating System Security

The DSSSENH cryptomodule is intended to run on Windows XP in Single User Mode.

³ The TCB is the part of the operating system that is designed to meet the security functional requirements of the Controlled Access Protection Profile, which can be found at http://www.radium.ncsc.mil/tepp/library/protection_profiles/index.html. At this time, Windows XP has not been evaluated.

When an operating system process loads the cryptomodule into memory, the cryptomodule runs a DES MAC on the cryptomodule's disk image of DSSSENH.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.

Each process requesting access is provided its own instance of the module. As such, each process has full access to all information and keys within the module. Note that no keys or other information are maintained upon detachment from the DLL, thus an instantiation of the module will only contain keys or information that the process has placed in the module.

MORE INFORMATION

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



Operating System

Microsoft DSS/Diffie-Hellman Enhanced Cryptographic Provider

FIPS 140-1 Documentation: Finite State Machine

Abstract

This document specifies the finite state machine for the DSSENH as described in FIPS PUB 140-1.

CONTENTS

INTRODUCTION

FINITE STATE MACHINE

APPENDIX A

APPENDIX B

FOR MORE INFORMATION

INTRODUCTION

The 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 Windows XP, 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.

FINITE STATE MACHINE

The DSSSENH cryptomodule can be in exactly one of the following states at any given moment. Transitions between states can be automatic or result from user intervention.

States

See Appendix A and B for more information.

Power Up

The Power Up state is entered when a process thread calls the Microsoft CryptoAPI function `CryptAcquireContext()` (encapsulated in `ADVAPI32.DLL`) in the following manner:

```
CryptAcquireContext(&hProv, pszContainer, MS_ENH_DSS_DH_PROV,  
PROV_DSS_DH, dwFlags)
```

This `ADVAPI32.DLL` function locates DSSSENH on the user's system, verifies its export compliance signature, and attempts to load DSSSENH via `LoadLibrary()` and run its `DLLInitialize()` function.

Power Down

The Power Down state is entered when DSSSENH library is unloaded either explicitly (e.g. a process thread calls `FreeLibrary()`) or implicitly (e.g. the process exits or is killed.)

Init Error

The Init Error State is entered when DSSSENH's `DLLInitialize()` fails as a result of either configuration errors (i.e. provider could not be found, not enough memory, etc.) or errors resulting from the power up self-tests.

Un-Initialized

The Un-Initialized state is entered when `ADVAPI32.DLL` successfully loads DSSSENH and calls its `CPAcquireContext()` function. If `CryptAcquireContext()` was called with any valid `dwFlags` other than `CRYPT_VERIFY_CONTEXT` or `CRYPT_DELETE_CONTEXT`, DSSSENH attempts to load the requested key container.

Initialized

The Initialized state is entered when CPACquireContext() completes successfully and a cryptographic provider handle (hProv) is returned to the client through the original ADVAPI32.DLL CryptAcquireContext() call. While a key container has been found, no keys have yet been loaded. Keyless cryptographic operations occur from the Initialized state until such time a keyed cryptographic operation is requested.

Key Entry

The Key Entry state is entered when a keyed cryptographic operation is requested such as CryptImportKey(), CryptSignHash(), CryptSetKeyParam (when the private key is generated with KP_X), or CryptGenKey() (when a DSS or DH private key is being generated). Keys are uncovered/unprotected using the Data Protection APIs (DPAPI). If keys are successfully uncovered/unprotected, DSSSENH will automatically transition to the Key Initialized state.

Key Initialized

The Key Initialized state is entered after keys have been loaded. This state is identical to the Initialized state except both keyless and keyed cryptographic operations can occur within this state.

Operation Error

The Operation Error state is entered whenever an error occurs as a result of a cryptographic operation. DSSSENH will automatically transition back to either the Initialized or Key Initialized depending on whether or not keys have been successfully loaded.

State Transitions

See Appendix A.

State Diagrams

See Appendix B.

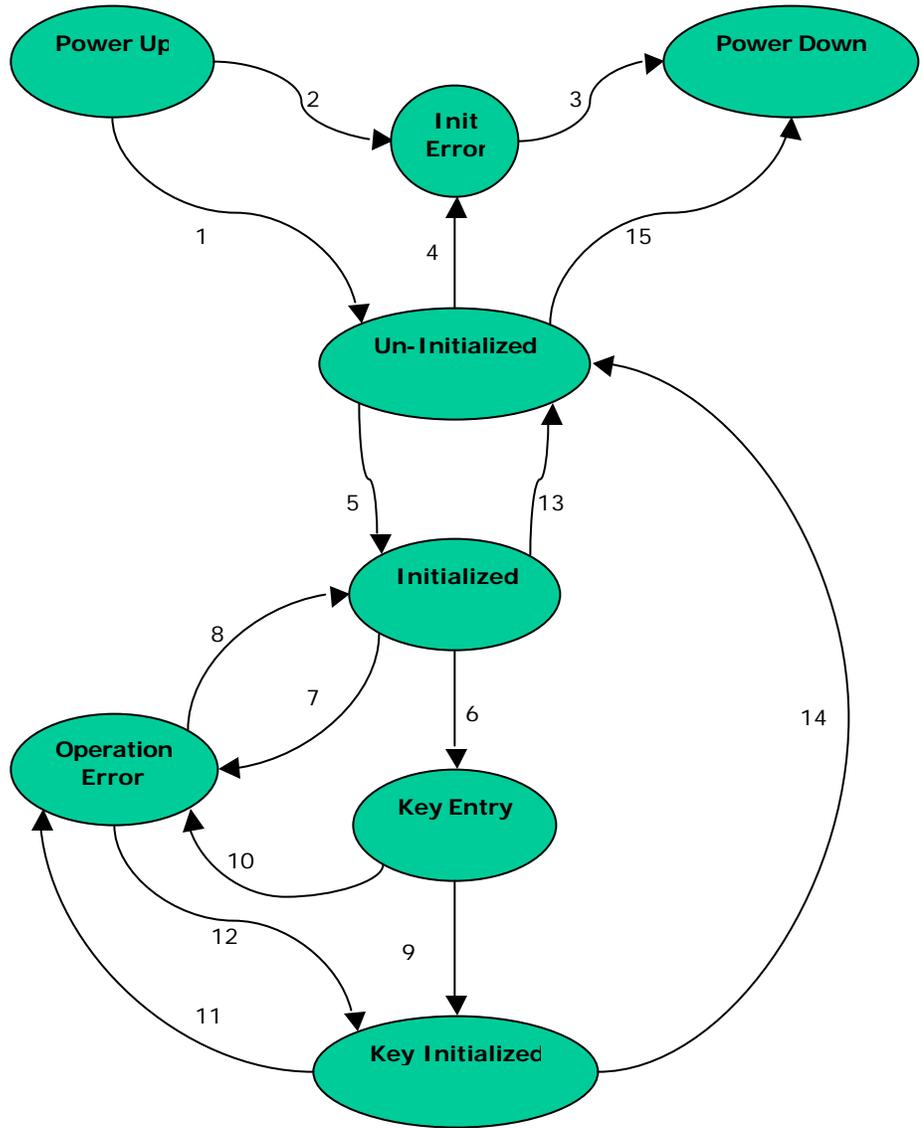
APPENDIX A

The following table describes the state transitions possible within the DSSSENH cryptomodule during operation.

	Current State	Input	Output	Next State
1	Power Up	DSSSENH loads	NO_ERROR	Un-Initialized
2	Power Up	DSSSENH.DLL not found	NTE_PROV_DLL_NOT_FOUND	Init Error
2	Power Up	Bad export compliance signature	NTE_BAD_SIGNATURE	Init Error
2	Power Up	DES MAC check on cryptographic provider fails	NTE_PROVIDER_DLL_FAIL	Init Error
2	Power Up	One or more power-on cryptographic self-tests fail	NTE_PROVIDER_DLL_FAIL	Init Error
2	Power Up	System error	System error message	Init Error
3	Init Error	Automatic transition	No output	Power Down
4	Un-Initialized	Cannot load key container	NTE_BAD_KEYSET	Init Error
4	Un-Initialized	dwFlags is CRYPT_DELETEKEYSET or CRYPT_VERIFYCONTEXT but operation could not be completed	NTE_BAD_KEYSET or NTE_FAIL	Init Error
5	Un-Initialized	dwFlags is not either CRYPT_DELETEKEYSET or CRYPT_VERIFYCONTEXT	NO_ERROR and valid provider handle (hProv)	Initialized
6	Initialized	Keyed cryptographic operation requested (i.e. CryptImportKey(), CryptSignHash(), CryptSetKeyParam (when the private key is generated with KP_X), or CryptGenKey() (when a DSS or DH private key is being generated))	No output	Key Entry
7	Initialized	Generic cryptographic operation failure	Operation specific error message	Operation Error
8	Operation Error	Automatic transition when keys have not yet been loaded	No output	Initialized
9	Key Entry	Keys uncovered/unprotected with DPAPI and loaded	No output	Key Initialized
10	Key Entry	Keys could not be uncovered/unprotected with DPAPI	NTE_FAIL	Operation Error
11	Key Initialized	Generic cryptographic operation failure	Operation specific error message	Operation Error
12	Operation Error	Automatic transition when keys have already been loaded	No output	Key Initialized
13	Initialized	CryptReleaseContext() called	NO_ERROR	Un-Initialized
14	Key Initialized	CryptReleaseContext() called	NO_ERROR	Un-Initialized
15	Un-Initialized	Automatic transition when no other outstanding provider handles exist	NO_ERROR	Power Down
15	Un-Initialized	Automatic transition when dwFlags is CRYPT_DELETEKEYSET or CRYPT_VERIFYCONTEXT and operation successfully completes	NO_ERROR	Power Down

APPENDIX B

The following diagram illustrates the finite state machine of the DSSSENH cryptomodule.



FOR MORE
INFORMATION

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



Operating System

Microsoft DSS/Diffie-Hellman Enhanced Cryptographic Provider

FIPS 140-1 Documentation: Master Component List

Abstract

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

This document specifies the master component list for the DSSNH as described in FIPS PUB 140-1.

CONTENTS

MASTER COMPONENT LIST

APPENDIX A

FOR MORE INFORMATION

MASTER COMPONENT LIST

The DSSSENH cryptomodule is a software cryptomodule and is intended to operate on a PC running Windows XP. Several components of the base PC are also to be considered components of the cryptomodule.

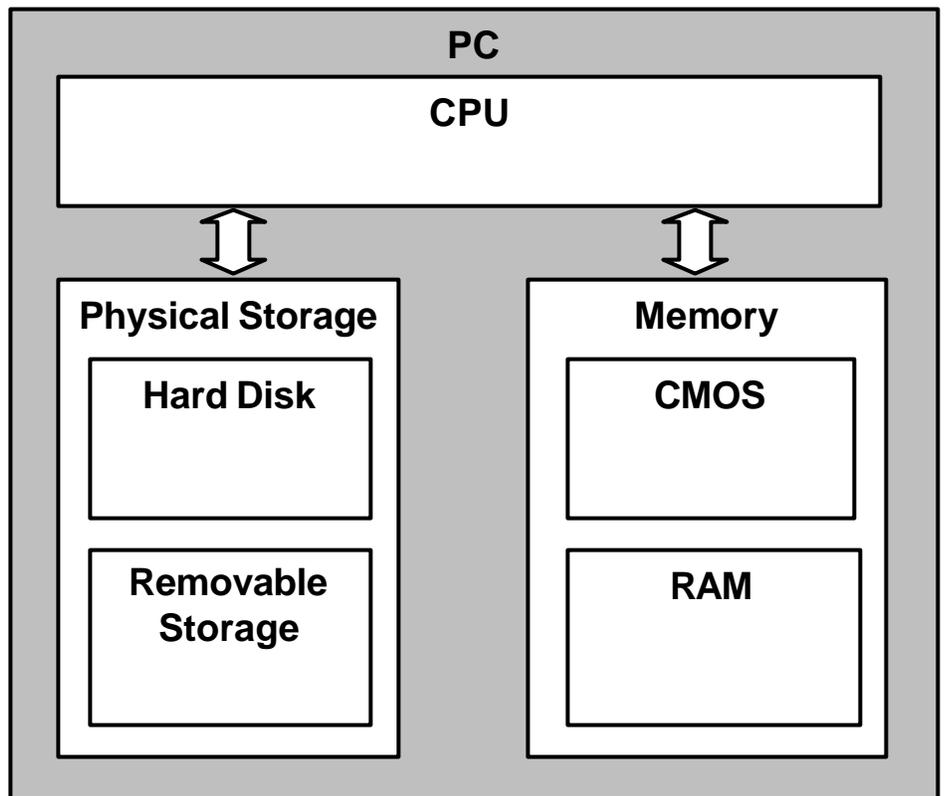
Components

The following components are to be considered components of the cryptomodule (see Appendix A below):

- PC Enclosure
- Central Processing Unit (CPU)
- Physical Storage (Hard Drives and Removable Storage)
- Memory (RAM and CMOS)

APPENDIX A

The following diagram illustrates the master components of the DSSSENH cryptomodule.



FOR MORE
INFORMATION

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