Microsoft

Windows NT®

*Operating System*

# Microsoft DSS/Diffie-Hellman Enhanced Cryptographic Provider

## FIPS 140-1 Documentation: Security Policy


**Abstract**

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 Microsoft Windows NT, 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.

This document specifies the security policy for the DSSENH as described in FIPS PUB 140-1.

CONTENTS

# 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 Microsoft Windows NT, 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.

## SECURITY POLICY

DSSENH operates under several rules that encapsulate its security policy.

- DSSENH is supported on Windows NT 4.0 with Service Pack 4 or later.
- DSSENH relies on Windows NT 4.0 for the authentication of users.
- DSSENH enforces a single role (i.e. Authenticated User) which is a combination of the User and Cryptographic Officer roles as defined in FIPS PUB 140-1.
- All users authenticated by Windows NT 4.0 employ the Authenticated User role.
- All services implemented within DSSENH are available to the Authenticated User role.
- Keys created within DSSENH by one user are not accessible to any other user via DSSENH.
- DSSENH relies on Windows NT 4.0 for the secure storage of keys.
- DSSENH performs the following self-tests upon power up:
  - RC4 encrypt/decrypt
  - RC2 ECB encrypt/decrypt
  - DES ECB encrypt/decrypt
  - 3DES ECB encrypt/decrypt
  - DES40 ECB encrypt/decrypt
  - 3DES 112 ECB 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.

## SPECIFICATION OF ROLES

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

An application requests the crypto module to generate keys for a user. Keys are generated, used and deleted as requested by applications. There are not implicit keys associated with a user. Each user may have numerous keys, signature and key exchange, and these keys are separate from other users' keys.

### Maintenance Roles

Maintenance roles are not supported by DSSENH.

### Multiple Concurrent Operators

DSSENH is intended to run on Windows NT 4.0 with Service Pack 4 or later in Single User Mode. When run in this configuration, multiple concurrent operators are not supported.

## SPECIFICATION OF SERVICES

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

### Key Storage

The following functions provide interfaces to the NT4.0 OS's Protect Storage key storage functions.

#### CryptAcquireContext

The CryptAcquireContext function is used to acquire a handle to a particular key container within 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 within the CSP matching the name specified by the *pszContainer* parameter.

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

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

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

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

### Key Generation and Exchange

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

#### CryptDeriveKey

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

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

#### CryptDestroyKey

The CryptDestroyKey function releases the handle referenced by the *hKey*

parameter. After a key handle has been released, it becomes invalid and cannot be used again.

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

### CryptExportKey

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

A handle to the key to be exported is passed to the function, and the function returns a key blob. This key blob can be sent over a nonsecure transport or stored in a nonsecure storage location. The 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 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).

### 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).

#### CryptSetKeyParam

The CryptSetKeyParam function customizes various aspects of a key's operations. This function is used to set session-specific values for symmetric keys.

### 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, RC2 or DES40).

#### 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. Currently, only a single value is defined for this function.

### 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.

## 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, RC2, RC4). When private keys are generated or imported from archival the are outputted to the NT4.0 OS protected storage in a covered form.

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

See MSDN Library\Platform SDK\Windows Base Services\Security\CryptoAPI 2.0\CryptoAPI Reference\CryptoAPI Functions\Base Cryptography Functions\Key Generation and Exchange Functions for more information.

### Key Storage

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

When an Authenticated User requests a keyed cryptographic operation from DSSENH, his/her keys are retrieved from the Microsoft Protected Storage System

(PSTORE) service of the NT4.0 OS. The PSTORE service, called via a local procedure call (LPC) from DSSENH, runs as Local System (analogous to the UNIX root account) and receives the caller's security token via the LPC call parameters. PSTORE then impersonates the user and retreives the Security Identity Descriptor (SID) from the user token. PSTORE uses the SID to navigate to the following System Registry location:

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

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

## Key Archival

DSSENH does not directly archive cryptographic keys. The Authenticated User may choose to export a cryptographic key labeled as exportable (cf. "Key Input and Output" above), but management of the secure archival of that key is the responsibility of the user.

## Key Destruction

All keys are destroyed and their memory location zeroized when the Authenticated User calls CryptDestroyKey on that key handle. Private keys (which are stored by the operating system in covered format in the protected storage system portion of the NT4.0 OS) are destroyed when the Authenticated User calls CryptAcquireContext with the CRYPT_DELETE_KEYSET flag.

## SELF-TESTS

### Mandatory

Software test 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
- DES40 ECB encrypt/decrypt KAT
- 3DES 112 ECB 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 DSSENH.

### Operation Authentication

DSSENH inherits all authentication from the Microsoft Windows NT operating system upon which it runs. Microsoft Windows NT 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 Users's) security token. Every user that has been authenticated by Microsoft Windows NT is naturally assigned the Authenticated User role when he/she accesses DSSENH.

#### Identity-based Authentication

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

### Operating System Security

The DSSENH cryptomodule is intended to run on Windows NT 4.0 with Service Pack 4 or later in Single User Mode.

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

Each operating system process creates an unique instance of the cryptomodule that is wholly dedicated to that process. The cryptomodule is not shared between processes.

## 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).