Windows Server 2008 R2 Enhanced DSS and Diffie-Hellman Cryptographic Provider (DSSENH)

Document Version 1.9 (Windows Server 2008 R2)

FIPS 140-2 Documentation: Security Policy

June 8, 2011

Abstract

This document specifies the security policy for the Windows Server 2008 R2 Enhanced DSS and Diffie-Hellman Cryptographic Provider (DSSENH) as described in FIPS PUB 140-2.

CONTENTS

Introduction

Microsoft Corporation's Windows Server 2008 R2 Enhanced DSS and Diffie-Hellman Cryptographic Provider (DSSENH) is a FIPS 140-2 Level 1 compliant, general-purpose, software-based, cryptographic module. Like other cryptographic providers that ship with Microsoft Windows Server 2008 R2, 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 140-2 compliant cryptographic support.

Windows Server 2008 R2 does not ship the previously FIPS-140-1 validated Microsoft Base DSS and Diffie-Hellman Cryptographic Provider (DSSBASE.DLL) anymore. There is no lost of functionality as the DSSENH functionality has always been a superset of the DSSBASE functionality.

Cryptographic Boundary

Windows Server 2008 R2 Enhanced DSS and Diffie-Hellman Cryptographic Provider (DSSENH) (Software version 6.1.7600.16385) consists of a single dynamically-linked library (DLL) named DSSENH.DLL, which comprises the modules logical boundary. 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 140-2, is Multi-Chip Standalone. It should be noted that the Data Protection API of Microsoft Windows Server 2008 R2 is not part of the module and should be considered to be outside the boundary.

SECURITY POLICY

DSSENH operates under several rules that encapsulate its security policy.

- DSSENH is supported on Windows Server 2008 R2 and Windows Server 2008 R2 SP1 (in a single-user environment).
- DSSENH operates in FIPS mode of operation only when used with Windows Server 2008 R2 Code Integrity (ci.dll) validated to FIPS 140-2 under Cert. #1334 operating in FIPS mode and Microsoft Windows Server 2008 R2 Kernel Mode Cryptographic Primitives Library (cng.sys) validated to FIPS 140-2 under Cert. #1335 operating in FIPS mode.
- DSSENH provides no user authentication. Roles are assumed implicitly. The authentication provided by the Windows Server 2008 R2 operating system is not in the scope of the validation.
- DSSENH is only in its Approved mode of operation when FIPS approved security functions are used and Windows is booted normally, meaning Debug mode is disabled and Driver Signing enforcement is enabled.
- DSSENH operates in its FIPS mode of operation only when one of the following DWORD registry values is set to 1:
    - HKLM\SYSTEM\CurrentControlSet\Control\Lsa\FIPSAlgorithmPolicy\Enabled
    - HKLM\SYSTEM\CurrentControlSet\Policies\Microsoft\Cryptography\Configuration\SelfTestAlgorithms
- All services provided by the DSSENH.DLL are available to the User and Crypto-officer roles.
- Keys created within DSSENH by one user are not accessible to any other user via DSSENH.
- When operating this module under Window 7 the following algorithms are Approved Security functions and can be used in FIPS mode:
    - DSA (Cert. #390), RNG (Cert. #649), SHA-1 (Cert. #1081), Triple-DES (Cert. #846), and Triple-DES MAC (Cert. #846, vendor affirmed).
- DSSENH supports the following FIPS allowed algorithms: Diffie-Hellman
- DSSENH supports the following non-FIPS approved algorithms[1]: DES, DES MAC, RC4, RC2, RC2 MAC, MD5, DES40, and DES40 MAC.

DSSENH was tested using the following machine configurations:

x64	Windows Server 2008 R2– HP Compaq dc7600
IA64	Windows Server 2008 R2– HP zx2000
x64	Windows Server 2008 R2 SP1 – HP Compaq dc7600
IA64	Windows Server 2008 R2 SP1 – HP zx2000

The following diagrams illustrate the master components of the module and how it relates to the rest of the Windows CryptoAPI system:

---

[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.
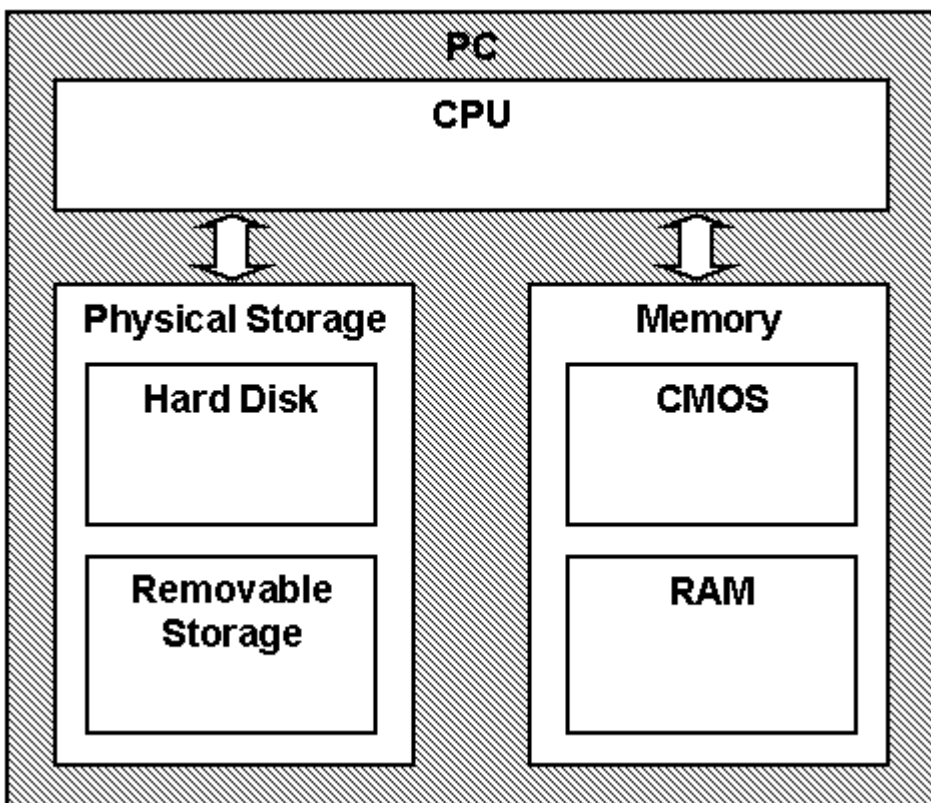
Figure 1 Master components of DSSENH module
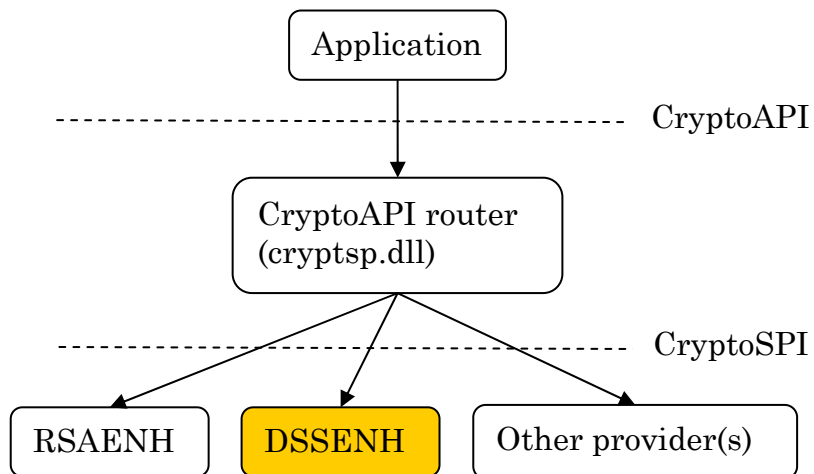
Figure 2 Relationship to other components in Windows CryptoAPI system – cryptographic module shown in gold

SPECIFICATION OF ROLES

DSSENH module supports both the User and Cryptographic Officer roles (as defined in FIPS PUB 140-2). Both roles 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 DSSENH.

Multiple Concurrent Operators

DSSENH is intended to run on Windows Server 2008 R2 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 is provided a separate instance of the module (a separate instantiation of the DLL), the operator 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 the User and Crypto-officer roles.

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 module (CSPM). This returned handle can then be used to make calls to the selected CSPM. 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 CSPM with the characteristics described in the *dwProvType* and *pszProvider* parameters. If the CSPM 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 CSPM.

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

Approved Random Number Generators are used for all Key Generation. 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 CSPM 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. A calling application can make use of this function as the pseudo-random function (PRF) of TLS v1.0; however, the use of this function to derive keys is not allowed in FIPS mode.

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 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 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 CSPM 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 module (CSPM) 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 CSPM. 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. Only Triple-DES may be used to encrypt private keys for export.

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 encrypting the keys with another symmetric key. The encrypted 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.

Operators have two options while generating Diffie-Hellman keys for key exchange purposes — having CryptoAPI generate all new values for G, P, and X or by using existing values for G and P, and generating a new value for X. Generating completely new keys requires the operator to call CryptGenKey passing either CALG_DH_SF or CALG_DH_EPHEM in the Algid parameter. The key will be generated, using new, random values for G and P, a newly calculated value for X, and its handle will be returned in the phKey parameter. The process for generating keys using pre-defined G & P values is more involved. Refer to http://msdn.microsoft.com/library/default.asp?url=/library/en-us/security/security/diffie_hellman_keys.asp for detailed directions on key generation and the key establishment process.

DSS keys and parameters are generated using the SHA-1 based RNG from FIPS 186-2 DSA random generator.

CryptGenRandom

The CryptGenRandom function fills a buffer with random bytes. The random number generation algorithm is the SP800-90 AES-256 based counter mode random number generation algorithm. During the function initialization, a seed is read from an in-kernel RBG, which exists outside the cryptographic boundary. CryptGenRandom accepts caller supplied data through its in/out pbBuffer parameter. This data is mixed with the seed.

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 module (CSPM).

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

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

To import a Diffie-Hellman (DH) key into the CSPM, call CryptImportKey, passing a pointer to the public key BLOB in the pbData parameter, the length of the BLOB in the dwDataLen parameter, and the handle to a DIFFIE-HELLMAN key in the hImpKey parameter. This call to CryptImportKey causes the calculation, (Y^X) mod P, to be performed thus creating the shared, secret key and completing the key exchange. This function call returns a handle to the new, secret, bulk-encryption key in the hKey parameter.

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 CSPM 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 CSPM 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 is the cryptographic hashing algorithm 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, Triple-DES, DES40, and RC2)[2].

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, 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, 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.

---

[2] Only Triple-DES MAC can be used in the FIPS mode of operation

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. 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.

Data Input and Output Interfaces

The Data Input Interface for DSSENH.DLL consists of the DSSENH export functions. Data and options are passed to the interface as input parameters to the DSSENH export functions. Data Input is kept separate from Control Input

by passing Data Input in separate parameters from Control Input.

The Data Output Interface for DSSENH.DLL also consists of the DSSENH export functions.

Control Input Interface
The Control Input Interface for DSSENH.DLL also consists of the DSSENH export functions. Options for control operations are passed as input parameters to the DSSENH export functions.

Status Ouput Interface
The Status Output Interface for DSSENH.DLL also consists of the DSSENH export functions. For each function, the status information is returned to the caller as the return value from the function.

Cryptographic Key Management

The DSSENH cryptomodule manages keys in the following manner.

Cryptographic Keys, CSPs, and SRDIs

The DSSENH crypto module contains the following security relevant data items:

Security Relevant Data Item	SRDI Description
Symmetric encryption/decryption keys	Keys used for AES or TDEA encryption/decryption.
DSA Public Keys	Keys used for the verification of DSA digital signatures.
DSA Private Keys	Keys used for the calculation of DSA digital signatures.
DH Public and Private exponents	Public and private values used for Diffie-Hellman key establishment.
Triple-DES MAC keys	Keys used for Triple-DES MAC.

Access Control Policy

The DSSENH crypto module allows controlled access to the SRDIs contained within it. The following table defines the access that a service has to each. The permissions are categorized as a set of four separate permissions: read (r), write (w), execute (x), delete (d). If no permission is listed, the service has no access to the SRDI.

DSSENH crypto module SRDI/Service Access Policy	Symmetric encryption and decryption keys	DSA Public Keys	DSA Private Keys	DH Public and Private exponents	Triple-DES MAC keys
Key Storage Services	r/x	r/x	r/x	r/x	r/x
Key Generation and Exchange Services	r/w/d	r/w/d	r/w/d	r/w/d	r/w/d
Data Encryption and Decryption Services	x				
Hashing and Digital Signature Services		x	x		

Key Material

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

(MSDN Home > MSDN Library > Win32 and COM Development > Security > Cryptography > Cryptography Reference > General Cryptography Structures)

Key Generation

Random keys can be generated by calling the CryptGenKey() function. DSA 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.

(MSDN Home > MSDN Library > Win32 and COM Development > Security > Cryptography > Cryptography Reference > Cryptography Functions > Key Generation and Exchange Functions)

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 or the values may be exported in plaintext. Only Triple-DES may be used to encrypt private keys for export.   When private keys are generated or imported from archival, they are covered/protected with the Microsoft Windows Server 2008 R2 Data Protection API (DPAPI) and then output 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.

(MSDN Home > MSDN Library > Win32 and COM Development > Security >

Key Storage

DSSENH 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 Server 2008 R2. The DPAPI is a separate component of the operating system that is outside the boundaries of the crypto module but relies upon DSSENH 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. DSSENH offloads the key storage operations to the Microsoft Windows Server 2008 R2 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 stored in the Microsoft Windows Server 2008 R2 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 operator requests a keyed cryptographic operation from DSSENH, his/her keys are retrieved from the file system by DSSENH with the support of DPAPI.

The readers may refer to the technical paper "Windows Data Protection" (http://msdn.microsoft.com/library/en-us/dnsecure/html/windataprotection-dpapi.asp) for further detail of DPAPI.

Key Archival

DSSENH does not directly archive cryptographic keys. The operator 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 operator 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 Server 2008 R2 DPAPI system portion of the OS) are destroyed when the operator calls CryptAcquireContext with the CRYPT_DELETE_KEYSET flag.

Self-Tests

Power up

The following algorithm tests are initiated upon power-up without operator intervention:

- Triple-DES encrypt/decrypt KAT
- DSA sign/verify test
- FIPS 186-2 x-Change Notice Regular RNG KAT

Conditional

The following are initiated at key generation:

- DSSENH performs a pair-wise consistency test upon each invocation of DSA key generation.
- Continuous random number generator test

Miscellaneous

The following items address requirements not addressed above.

Cryptographic Bypass

Cryptographic bypass is not support in DSSENH.

Operator Authentication

DSSENH provides no authentication of operators. However, the Microsoft Windows7operating system upon which it runs does provide authentication, but this is outside the scope of DSSENH's FIPS validation. The information about the authentication provided by Microsoft Windows Server 2008 R2 is for informational purposes only. Microsoft Windows Server 2008 R2 requires authentication from a trusted computer base (TCB[3]) before a user is able to access system services. Once a user is authenticated from the TCB, a process is created bearing the operator's security token. All subsequent processes and threads created by that operator are implicitly assigned the parent's (thus the operator's) security token. Every user that has been authenticated by Microsoft Windows Server 2008 R2 is naturally assigned the operator role when he/she accesses DSSENH.

ModularExpOffload

The ModularExpOffload function offloads modular exponentiation from a CSPM to a hardware accelerator. The CSPM will check in the registry for the value HKLM\Software\Microsoft\Cryptography\ExpoOffload that can be the name of a DLL. The CSPM uses LoadLibrary to load that DLL and calls GetProcAddress to get the OffloadModExpo entry point in the DLL specified in the registry. The CSPM 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 DSSENH in a FIPS compliant manner, this function should only be used if the hardware accelerator is FIPS validated.

Operating System Security

The DSSENH crypto module is intended to run on Windows Server 2008 R2 in Single User Mode, where there is only one interactive user during a logon session.

---

[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/tpep/library/protection_profiles/index.html >. At this time, Windows Server 2008 R2 has not been evaluated.

Each operating system process creates a unique instance of the crypto module that is wholly dedicated to that process. The crypto module 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 Server 2008 R2, check out our World Wide Web site at http://www.microsoft.com/windows.