Security Policy

for FIPS 140-2 Validation

Microsoft Windows 8

Microsoft Windows Server 2012

Microsoft Windows RT

Microsoft Surface Windows RT

Microsoft Surface Windows 8 Pro

Microsoft Windows Phone 8

Microsoft Windows Storage Server 2012

Enhanced DSS and Diffie-Hellman Cryptographic Provider (DSSENH.DLL)

DOCUMENT INFORMATION

Version Number	1.3
Updated On	December 17, 2014

TABLE OF CONTENTS

1 Introduction

Microsoft Corporation's Windows 8, Windows RT, Windows Server 2012, Windows Storage Server 2012, and Windows Phone 8 Enhanced DSS and Diffie-Hellman Cryptographic Provider, hereafter referred to by its short name of DSSENH, is a FIPS 140-2 Level 1 compliant, general-purpose, software-based, cryptographic module. Like other cryptographic providers that ship with Windows 8, Windows RT, Windows Server 2012, Windows Storage Server 2012, and Windows Phone 8, 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.

1.1 List of Cryptographic Module Binary Executables

DSSENH.DLL – Version 6.2.9200 for Windows 8, Windows RT, Windows Server 2012, Windows Storage Server 2012, and Windows Phone 8

1.2 Brief Module Description

DSSENH.DLL is a dynamically-linked library (DLL) providing Enhanced DSS and Diffie-Hellman cryptographic algorithms.

1.3 Validated Platforms

The Enhanced DSS and Diffie-Hellman Cryptographic Provider component listed in Section 1.1 was validated using the following machine configurations:

x86 Microsoft Windows 8 Enterprise – Dell Dimension C521 (AMD Athlon 64 X2 Dual Core)
x64 Microsoft Windows 8 Enterprise – Dell PowerEdge SC430 (Intel Pentium D without AES-NI)
x64-AES-NI Microsoft Windows 8 Enterprise – Intel Client Desktop (Intel Core i7 with AES-NI)
x64 Microsoft Windows Server 2012 – Dell PowerEdge SC430 (Intel Pentium D without AES-NI)
x64-AES-NI Microsoft Windows Server 2012 – Intel Client Desktop (Intel Core i7 with AES-NI)
ARMv7 Thumb-2 Microsoft Windows RT – NVIDIA Tegra 3 Tablet (NVIDIA Tegra 3 Quad-Core)
ARMv7 Thumb-2 Microsoft Windows RT – Qualcomm Tablet (Qualcomm Snapdragon S4)
ARMv7 Thumb-2 Microsoft Windows RT – Microsoft Surface Windows RT (NVIDIA Tegra 3 Quad-Core)
x64-AES-NI Microsoft Windows 8 Pro – Microsoft Surface Windows 8 Pro (Intel x64 Processor with AES-NI)
ARMv7 Thumb-2 Microsoft Windows Phone 8 – Windows Phone 8 (Qualcomm Snapdragon S4)
x64 Microsoft Windows Storage Server 2012 – Intel Maho Bay (Intel Core i7 without AES-NI)
x64-AES-NI Microsoft Windows Storage Server 2012 – Intel Maho Bay (Intel Core i7 with AES-NI)

The Enhanced DSS and Diffie-Hellman Cryptographic Provider maintains FIPS 140-2 validation compliance (according to FIPS 140-2 PUB Implementation Guidance G.5) on the following platforms:

x86 Microsoft Windows 8
x86 Microsoft Windows 8 Pro

x64 Microsoft Windows 8
x64 Microsoft Windows 8 Pro

x64 Microsoft Windows Server 2012 Datacenter

x64-AES-NI Microsoft Windows 8
x64-AES-NI Microsoft Windows 8 Pro
x64-AES-NI Microsoft Windows Server 2012 Datacenter

1.4 Cryptographic Boundary

The software binary that comprises the cryptographic boundary for Enhanced DSS and Diffie-Hellman Cryptographic Provider is DSSENH.DLL . The crypto boundary is also defined as the enclosure of the computer system on which DSSENH is to be executed. The physical configuration of DSSENH, as defined in FIPS-140-2, is multi-chip standalone.

It should be noted that the Data Protection API of Microsoft Windows 8, Windows RT, Windows Server 2012, Windows Storage Server 2012, and Windows Phone 8 is not part of the module and should be considered to be outside the boundary.

2 Security Policy

DSSENH operates under several rules that encapsulate its security policy.

- DSSENH is supported on Windows 8, Windows RT, Windows Server 2012, Windows Storage Server 2012, and Windows Phone 8 (in a single-user environment).
- DSSENH operates in FIPS mode of operation only when used with:
    - Windows 8, Windows RT, Windows Server 2012, Windows Storage Server 2012, and Windows Phone 8 Code Integrity (ci.dll) validated to FIPS 140-2 under Cert. #1897 for Windows 8, Windows RT, Windows Server 2012, Windows Storage Server 2012, and Windows Phone 8 operating in FIPS mode
    - Microsoft Windows 8, Windows RT, Windows Server 2012, Windows Storage Server 2012, and Windows Phone 8 Cryptographic Primitives Library (bcryptprimitives.dll) validated to FIPS 140-2 under Cert. #1892 for Windows 8, Windows RT, Windows Server 2012, Windows Storage Server 2012, and Windows Phone 8 operating in FIPS mode
    - Microsoft Windows 8, Windows RT, Windows Server 2012, Windows Storage Server 2012, and Windows Phone 8 Kernel Mode Cryptographic Primitives Library (cng.sys) validated to FIPS 140-2 under Cert. #1891 for Windows 8, Windows RT, Windows Server 2012, Windows Storage Server 2012, and Windows Phone 8 operating in FIPS mode.
- DSSENH provides no user authentication. Roles are assumed implicitly. The authentication provided by the Windows 8, Windows RT, Windows Server 2012, Windows Storage Server 2012, and Windows Phone 8 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
- The registry security policy settings can be observed with the regedit tool to determine whether the module is in FIPS mode.
- All services provided by DSSENH 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.

The following diagram illustrates the master components of the DSSENH module:

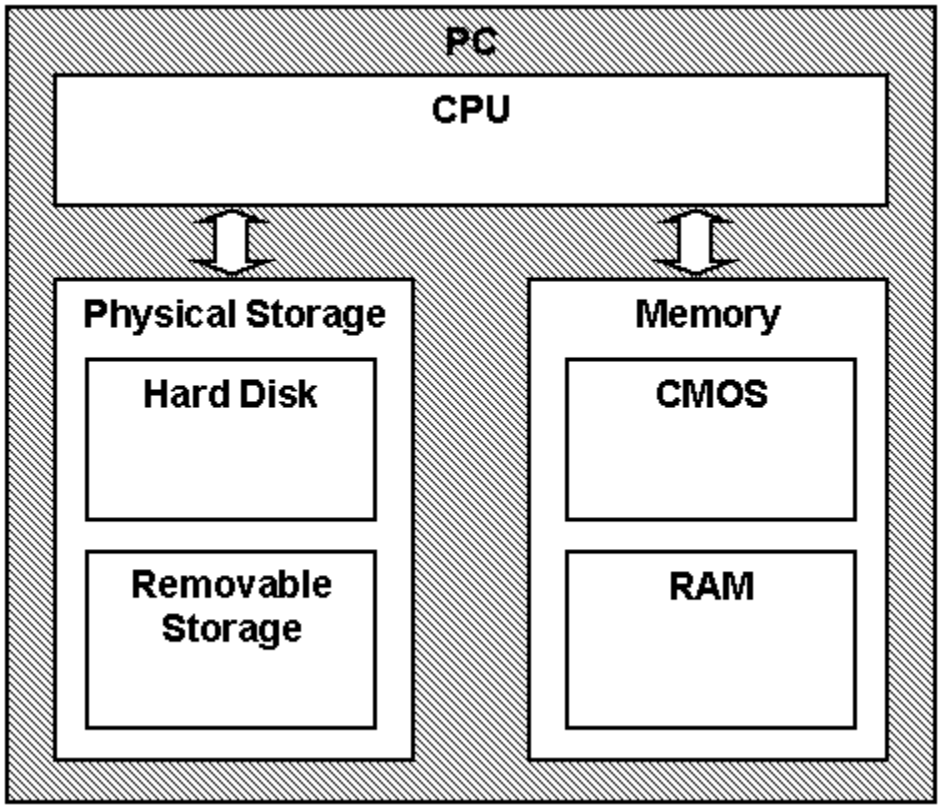

Figure 1 Master components of DSSENH module

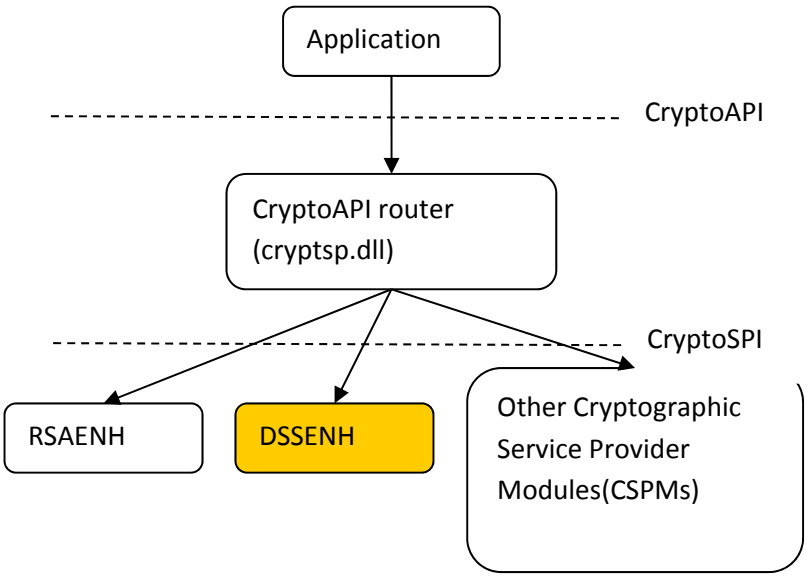

Figure 2 Relationship to other components in Windows CryptoAPI system – cryptographic module shown in gold

2.1 FIPS 140-2 Approved Algorithms

When operating DSSENH under Windows 8, Windows RT, Windows Server 2012, Windows Storage Server 2012, and Windows Phone 8, the following algorithms are Approved Security functions and can be used in FIPS mode:

- DSA (Cert. #686)
- Triple-DES[1] (Cert. #1386)
- Triple-DES MAC (Cert. #1386, vendor affirmed).
- SHA-1[2] (Cert. #1902)

2.2 Non-Approved Algorithms

- DSSENH supports the following FIPS allowed algorithms:
    - Diffie-Hellman[3] with the following key sizes for the cryptographic providers:

[1] Two-key Triple-DES is *restricted* and *legacy-use* according to NIST SP 800-131A. Users should start transitioning away from this algorithm to better, stronger choices.

[2] The SHA-1 hash is disallowed for use in digital signature generation after December 31, 2013. It can be used for digital signature verification legacy-use. Its use is Acceptable for non-digital signature generation applications.

[3] According to NIST SP 800-131A, Diffie-Hellman with 1024-bit key sizes are *deprecated* through the end of 2013 and 2048-bit key sizes will become the minimum. Users should start transitioning to the larger key sizes.

        - Microsoft Base DSS and Diffie-Hellman Cryptographic Provider 1024
        - Microsoft Enhanced DSS and Diffie-Hellman Cryptographic Provider 1024 – 4096
    - Triple-DES (Cert. #1386, key wrapping; key establishment methodology provides 80 or 112 bits of encryption strength)

DSSENH supports the following non-FIPS approved algorithms:

- DES
- DES MAC
- RC4
- RC2
- RC2 MAC
- MD5
- DES40
- DES40 MAC
- Diffie-Hellman with key sizes less than 1024 bits (primitives only)

Applications may not use any of these non-approved algorithms when operating the module in a FIPS mode. To operate the module in a FIPS mode, applications must only use FIPS 140-2 Approved algorithms.

2.3 Cryptographic Bypass

Cryptographic bypass is not supported in DSSENH.

2.4 Machine Configurations

DSSENH was tested using the machine configurations listed in Section 1.3 - Validated Platforms.

3 Operational Environment

The operational environment for DSSENH is Windows 8, Windows RT, Windows Server 2012, Windows Storage Server 2012, and Windows Phone 8 running on the hardware listed in Section 1.3 - Validated Platforms.

DSSENH is intended to run on Windows 8, Windows RT, Windows Server 2012, Windows Storage Server 2012, and Windows Phone 8 in Single User Mode, where there is only one interactive user during a logon session. 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, and DSSENH relies on the operational environment to maintain this isolation.

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.

4 Integrity Chain of Trust

The Windows Code Integrity module (CI.DLL) validates the integrity of DSSENH before DSSENH is loaded into memory. The validation procedure is documented in the Code Integrity Security Policy document in section 4.1.1.1 CiValidateImageHeader(). This section contains the algorithms and methods for validating the cryptographic module binaries. An RSA signature with a 2048-bit key and SHA-256 message digest are used.

5 Ports and Interfaces

As depicted in **Figure 2**, DSSENH is a Cryptographic Service Provider Module (CSPM) and implements a set of export functions known as the CryptoSPI. These CryptoSPI functions correspond directly, in a one-to-one manner, to functions in the CryptoAPI, which is the programming interface used by Windows applications to access DSSENH and other such cryptographic providers. More information about the CryptoSPI is available in the Windows Cryptographic Provider Development Kit, available from http://www.microsoft.com/download

5.1 Control Input Interface

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

5.2 Status Output Interface

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

5.3 Data Output Interface

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

5.4 Data Input Interface

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

6 Specification of Roles

The 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.

6.1 Maintenance Roles

Maintenance roles are not supported.

6.2 Multiple Concurrent Interactive Operators

There is only one interactive operator in Single User Mode. When run in this configuration, multiple concurrent interactive 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.

6.3 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.

6.4 Show Status Services

The User and Cryptographic Officer roles have the same Show Status functionality, which is, for each function, the status information is returned to the caller as the return value from the function.

6.5 Self-Test Services

The User and Cryptographic Officer roles have the same Self-Test functionality, which is described in Section 10 Self-Tests.

6.6 Service Inputs / Outputs

The User and Cryptographic Officer roles have service inputs and outputs as specified in Section 5 Ports and Interfaces and Section 7 Services.

7 Services

The following list contains all services available to an operator. All services are accessible to both the User and Crypto Officer roles.

Note that the functions named in this section are CryptoAPI functions; as mentioned in Section 5, these are called by the application and correspond directly to the CryptoSPI functions implemented by DSSENH.

7.1 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.

7.1.1 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.

7.1.2 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.

7.1.3 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.

7.1.4 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.

7.2 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.

7.2.1 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 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.

7.2.2 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.

7.2.3 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.

7.2.4 CryptGenKey

The CryptGenKey function generates a random cryptographic key in a FIPS Approved manner. 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 Cryptographic Primitives Library (bcryptprimitives.dll).

7.2.5 CryptGenRandom

The CryptGenRandom function fills a buffer with random bytes. This function merely forwards the call to a FIPS Approved RNG from the Cryptographic Primitives Library (bcryptprimitives.dll) with DRBG (Cert. #258).

7.2.6 CryptGetKeyParam

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

7.2.7 CryptGetUserKey

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

7.2.8 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.

7.2.9 CryptSetKeyParam

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

7.2.10 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.

7.3 Data Encryption and Decryption Services

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

7.3.1 CryptDecrypt

The CryptDecrypt function decrypts data previously encrypted using CryptEncrypt function.

7.3.2 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.

7.4 Hashing and Digital Signatures Services

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

7.4.1 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)[4].

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

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.

7.4.2 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.

7.4.3 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.

7.4.4 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.

7.4.5 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. This function only produces a hash and not a key.

7.4.6 CryptSetHashParam

The CryptSetHashParam function customizes the operations of a hash object.

7.4.7 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.

7.4.8 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.

7.4.9 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.

8 Authentication

The module does not provide authentication. Roles are implicitly assumed based on the services that are executed.

9 Security Relevant Data Items

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

Security Relevant Data Item	Description
Symmetric encryption/decryption keys	Keys used for 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.
Triple-DES MAC keys	Keys used for Triple-DES MAC.
DH Private and Public exponents	Private and public values used for Diffie-Hellman key establishment.

9.1 Access Control Policy

The DSSENH crypto module allows controlled access to the security relevant data items 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 item. The User and Cryptographic Officer roles have the same access to keys so roles are not distinguished in the table.

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		

9.2 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)

9.3 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)

9.4 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 8, Windows RT, Windows Server 2012, Windows Storage Server 2012, and Windows Phone 8 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 > Cryptography > Cryptography Reference > Key Generation and Exchange Functions)

9.5 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 8, Windows RT, Windows Server 2012, Windows Storage Server 2012, and Windows Phone 8. The DPAPI is a separate component of the operating system that is outside the boundaries of the crypto module. 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 8, Windows RT, Windows Server 2012, Windows Storage Server 2012, and Windows Phone 8 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 8, Windows RT, Windows Server 2012, Windows Storage Server 2012, and Windows Phone 8 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.

Please refer to the technical paper "Windows Data Protection" (http://msdn.microsoft.com/en-us/library/ms995355.aspx) for further detail of DPAPI.

9.6 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.

9.7 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 8, Windows RT, Windows Server 2012, Windows Storage Server 2012, and Windows Phone 8 DPAPI system portion of the OS) are destroyed when the operator calls CryptAcquireContext with the CRYPT_DELETE_KEYSET flag.

10 Self-Tests

10.1 Power-On Self-Tests

The following algorithm tests are initiated upon power-on (startup) without operator intervention:

- Triple-DES encrypt/decrypt KAT
- DSA sign/verify test

Note: TDES KAT covers TDES MAC and DSA KAT covers SHA-1 KAT.

If the self-test fails, the module will not load and status will be returned. If the status is not STATUS_SUCCESS, then that is the indicator a self-test failed.

10.2 Conditional Self-Tests

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

11 Design Assurance

The secure installation, generation, and startup procedures of this cryptographic module are part of the overall Windows 8, Windows RT, Windows Server 2012, and Windows Storage Server 2012 operating system secure installation, configuration, and startup procedures. After the operating system has been installed, it must be configured by enabling the "System cryptography: Use FIPS compliant algorithms for encryption, hashing, and signing" policy setting followed by restarting the system. This procedure is all the crypto officer and user behavior necessary for the secure operation of this cryptographic module.

Windows Phone 8 does not use the same installation, configuration, and startup procedures as the Windows operating system on a computer, but rather, is securely installed and configured by the cellular telephone carrier.

The procedures required for maintaining security while distributing and delivering versions of a cryptographic module to authorized operators are:

1. The secure distribution method is via the physical medium for product installation delivered by Microsoft Corporation, which is a DVD in the case of Windows 8 and Windows Server 2012. In the case of Windows RT, Surface Windows RT, Surface Windows 8 Pro, Windows Phone 8, and Windows Storage Server 2012, the cryptographic module is already installed at the factory and is only distributed with the hardware.
2. An inspection of authenticity of the physical medium can be made by following the guidance at this Microsoft web site: http://www.microsoft.com/en-us/howtotell/default.aspx
3. The installed version of Windows 8, Windows RT, Windows Server 2012, and Windows Storage Server 2012 must be verified to match the version that was validated. See Appendix A for details on how to do this.

For Windows Updates, the client only accepts binaries signed by Microsoft certificates. The Windows Update client only accepts content whose SHA-2 hash matches the SHA-2 hash specified in the metadata. All metadata communication is done over a Secure Sockets Layer (SSL) port. Using SSL ensures that the client is communicating with the real server and so prevents a spoof server from sending the client harmful requests. The version and digital signature of new cryptographic module releases must be verified to match the version that was validated. See Appendix A for details on how to do this.

12 Miscellaneous

The following items address requirements not addressed above.

12.1 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.

13 Mitigation of Other Attacks

The following table lists the mitigations of other attacks for this cryptographic module:

Algorithm	Protected Against	Mitigation	Comments
SHA1	Timing Analysis Attack	Constant Time Implementation	
	Cache Attack	Memory Access pattern is independent of any confidential data	
3DES	Timing Analysis Attack	Constant Time Implementation	

14 Additional Details

For the latest information on Microsoft Windows, check out the Microsoft web site at:

http://windows.microsoft.com

For more information about FIPS 140 evaluations of Microsoft products, please see:

http://technet.microsoft.com/en-us/library/cc750357.aspx

15 Appendix A – How to Verify Windows Versions and Digital Signatures

15.1 How to Verify Windows Versions

The installed version of Windows 8, Windows RT, Windows Server 2012, and Windows Storage Server 2012 must be verified to match the version that was validated using one of the following methods:

1. The ver command
   a. From Start, open the Search charm.
   b. In the search field type "cmd" and press the Enter key.
   c. The command window will open with a "C:\>" prompt.
   d. At the prompt, type "ver" and press the Enter key.
   e. You should see the answer "`Microsoft Windows [Version 6.2.9200]`".
2. The systeminfo command
   a. From Start, open the Search charm.
   b. In the search field type "cmd" and press the Enter key.
   c. The command window will open with a "C:\>" prompt.
   d. At the prompt, type "systeminfo" and press the Enter key.
   e. Wait for the information to be loaded by the tool.
   f. Near the top of the output, you should see:

```
OS Name:                   Microsoft Windows 8 Enterprise
OS Version:                6.2.9200 N/A Build 9200
OS Manufacturer:           Microsoft Corporation
```

If the version number reported by the utility matches the expected output, then the installed version has been validated to be correct.

15.2 How to Verify Windows Digital Signatures

After performing a Windows Update that includes changes to a cryptographic module, the digital signature and file version of the binary executable file must be verified. This is done like so:

1. Open a new window in Windows Explorer.
2. Type “C:\Windows\” in the file path field at the top of the window.
3. Type the cryptographic module binary executable file name (for example, “CNG.SYS”) in the search field at the top right of the window, then press the Enter key.
4. The file will appear in the window.
5. Right click on the file’s icon.
6. Select Properties from the menu and the Properties window opens.
7. Select the Details tab.
8. Note the File version Property and its value, which has a number in this format: x.x.xxxx.xxxxx.
9. If the file version number matches one of the version numbers that appear at the start of this security policy document, then the version number has been verified.
10. Select the Digital Signatures tab.
11. In the Signature list, select the Microsoft Windows signer.
12. Click the Details button.
13. Under the Digital Signature Information, you should see: “This digital signature is OK.” If that condition is true then the digital signature has been verified.