Microsoft Windows Vista Kernel Mode Security Support Provider Interface (ksecdd.sys) Security Policy Document

Microsoft Windows Vista Operating System

FIPS 140-2 Security Policy Document

This document specifies the security policy for the Microsoft Kernel Mode Security Support Cryptographic Module (KSECDD.SYS) as described in FIPS PUB 140-2.

May 22, 2009

Document Version: 1.8

1 Cryptographic Module Specification

Microsoft Kernel Security Support Provider Interface (KSECDD.SYS) is a FIPS 140-2 Level 1 compliant, general purpose, software-based, cryptographic module residing at kernel mode level of Windows Vista operating system. KSECDD.SYS (version 6.0.6001.22202 and 6.0.6002.18005) runs as a kernel mode export driver, and provides cryptographic services, through their documented interfaces, to Windows Vista kernel components.

The KSECDD.SYS encapsulates several different cryptographic algorithms in an easy-to-use cryptographic module accessible via the Microsoft CNG (Cryptography, Next Generation) API. It also supports several cryptographic algorithms accessible via a Fips function table request irp (I/O request packet). Windows Vista kernel mode components can use general-purpose FIPS 140-2 Level 1 compliant cryptography in KSECDD.SYS.

1.1 Cryptographic Boundary

The Windows Vista kernel mode KSECDD.SYS consists of a single kernel mode export driver (SYS). The cryptographic boundary for KSECDD.SYS is defined as the enclosure of the computer system, on which KSECDD.SYS is to be executed. The physical configuration of KSECDD.SYS, as defined in FIPS-140-2, is multi-chip standalone

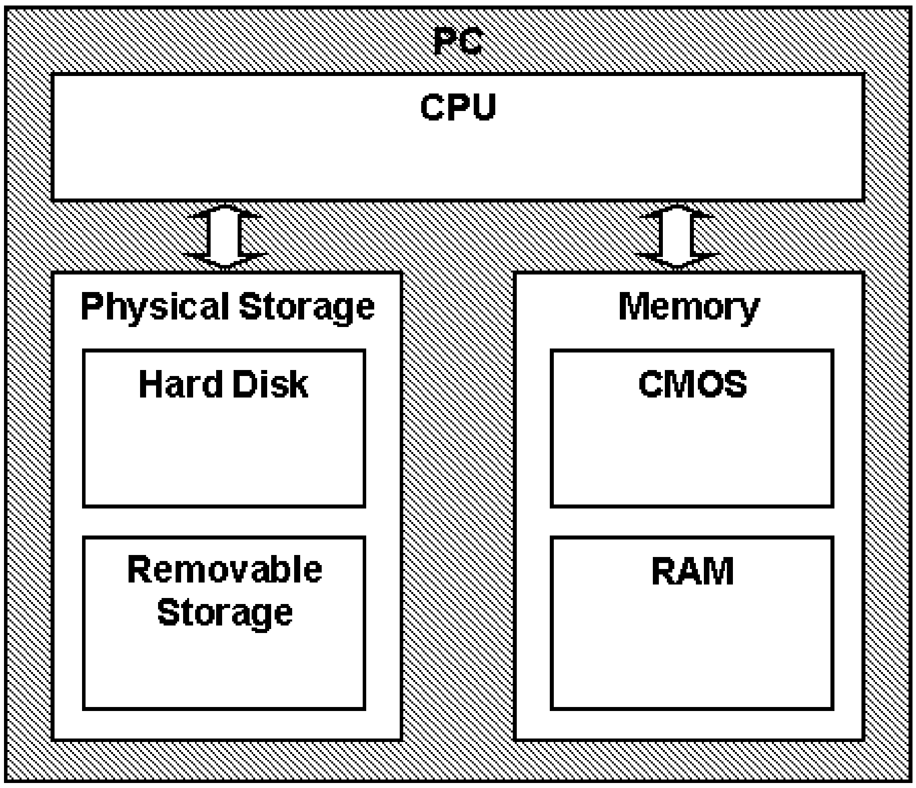
2 Security Policy

KSECDD.SYS operates under several rules that encapsulate its security policy.

- KSECDD.SYS is supported on Windows Vista Service Pack 1 and Service Pack 2.
- KSECDD.SYS operates in FIPS mode of operation only when used with the FIPS approved version of WINLOAD.EXE (FIPS 140-2 Cert. #979) operating in FIPS mode
- Windows Vista is an operating system supporting a "single user" mode where there is only one interactive user during a logon session.
- KSECDD.SYS is only in its Approved mode of operation when Windows is booted normally, meaning Debug mode is disabled and Driver Signing enforcement is enabled.
- All users assume either the User or Cryptographic Officer roles.
- KSECDD.SYS provides no authentication of users. Roles are assumed implicitly. The authentication provided by the Windows Vista operating system is not in the scope of the validation.
- All cryptographic services implemented within KSECDD.SYS are available to the User and Cryptographic Officer roles.
- In order to invoke the approved mode of operation, the user must call FIPS approved functions.
- KSECDD.SYS implements the following FIPS-140-2 Approved algorithms.
    - SHA-1, SHA-256, SHA-384, SHA-512 hash (Cert. #753)
    - SHA-1, SHA-256, SHA-384, SHA-512 HMAC (Cert. #412)
    - Triple-DES (2 key and 3 key) in ECB, CBC, and CFB with 8-bit feedback modes (Cert. #656)
    - AES-128, AES-192, AES-256 in ECB, CBC, and CFB with 8-bit feedback mode (Cert. #739)
    - AES-128, AES-192 and AES-256 in CCM (Cert. #756)
    - RSA (RSASSA-PKCS1-v1_5 and RSASSA-PSS) digital signatures (Cert. #353) and X9.31 RSA key-pair generation (Cert. #357)
    - ECDSA with the following NIST curves: P-256, P-384, P-521 (Cert. #82)
    - FIPS 186-2 general purpose random number generation algorithm per FIPS 186-2 (Cert. #435).
    - FIPS 186-2 regular random number generation algorithm per FIPS 186-2 (Cert. #435)
    - SP800-90 AES-256 based counter mode random generator algorithm (Vendor-Affirmed)
- KSECDD.SYS supports the following non-Approved algorithms allowed for use in FIPS mode.

  - Diffie-Hellman (DH) secret agreement (key agreement; key establishment methodology provides between 80 and 150 bits of encryption strength; non-compliant less than 80-bits of encryption strength).
  - ECDH with the following NIST curves: P-256, P-384, P-521 (key agreement; key establishment methodology provides between 128 and 256 bits of encryption strength)
  - RSA (key wrapping; key establishment methodology provides between 80 and 150 bits of encryption strength; non-compliant less than 80-bits of encryption strength).
  - TLS
  - SP800-90 Dual-EC DRBG random generator algorithm (non-compliant) – This RNG is non-approved and its output can only be used for generation of values which may be generated by non-approved RNGs (i.e. initialization vectors)
- KSECDD.SYS also supports the following non FIPS 140-2 approved algorithms, though these algorithms may not be used when operating the modules in a FIPS compliant manner.
  - AES-128, AES-192, and AES-256 GCM Mode (non-compliant)
  - AES-128, AES-192, and AES-256 GMAC mode (non-compliant)
  - RC2, RC4, MD2, MD4, MD5, HMAC MD5[1].
  - DES in ECB, CBC, and CFB with 8-bit feedback
  - IKEv1 Key Derivation Functions

The following diagram illustrates the master components of the module:



KSECDD.SYS was tested using the following machine configurations:

[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.

x86	Microsoft Windows Vista Ultimate Edition SP1 (x86 version) – Dell SC430 (Intel Pentium D 2.8GHz)
x64	Microsoft Windows Vista Ultimate Edition SP1 (x64 version) – Dell SC430 (Intel Pentium D 2.8GHz)

Microsoft also affirms that the module maintains validation conformance in the following configurations:

x86	Windows Vista Ultimate Edition SP2 (x86 version)
x64	Windows Vista Ultimate Edition SP2 (x64 version)

3 Cryptographic Module Ports and Interfaces

3.1 Export Functions

The following list contains the functions exported by KSECDD.SYS to its callers.

- BCryptCloseAlgorithmProvider
- BCryptCreateHash
- BCryptDecrypt
- BCryptDeriveKey
- BCryptDestroyHash
- BCryptDestroyKey
- BCryptDestroySecret
- BCryptDuplicateHash
- BCryptDuplicateKey
- BCryptEncrypt
- BCryptExportKey
- BCryptFinalizeKeyPair
- BCryptFinishHash
- BCryptFreeBuffer
- BCryptGenerateKeyPair
- BCryptGenerateSymmetricKey
- BCryptGenRandom
- BCryptGetProperty
- BCryptHashData
- BCryptImportKey
- BCryptImportKeyPair
- BCryptOpenAlgorithmProvider
- BCryptSecretAgreement
- BCryptSetProperty
- BCryptSignHash
- BCryptVerifySignature

KSECDD.SYS has additional export functions described in subsequent sections.

3.2 Data Input and Output Interfaces

The Data Input Interface for KSECDD.SYS consists of the KSECDD.SYS export functions. Data and options are passed to the interface as input parameters to the KSECDD.SYS 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 KSECDD.SYS also consists of the KSECDD.SYS export functions.

3.3 Control Input Interface

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

3.4 Status Output Interface

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

3.5 Cryptographic Bypass

Cryptographic bypass is not supported by KSECDD.SYS.

4 Roles and Authentication

4.1 Roles

KSECDD.SYS provides User and Cryptographic Officer roles (as defined in FIPS 140-2). These roles share all the services implemented in the cryptographic module.
When a kernel mode component requests the crypto module to generate keys, the keys are generated, used, and deleted as requested. There are no implicit keys associated with a kernel component. Each kernel component may have numerous keys.

4.2 Maintenance Roles

Maintenance roles are not supported by KSECDD.SYS.

4.3 Operator Authentication

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

The OS on which KSECDD.SYS executes (Microsoft Windows Vista) does authenticate users.
Microsoft Windows Vista requires authentication from the 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 for identification purpose. All subsequent processes and threads created by that Authenticated User are implicitly assigned the parent's (thus the Authenticated User's) security token.

5 Services

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

5.1 Cryptographic Module Power Up and Power Down

5.1.1 DriverEntry

Each Windows Vista driver must have a standard initialization routine DriverEntry in order to be loaded. The Windows Vista Loader is responsible to call the DriverEntry routine. The DriverEntry routine must have the following prototype.

```
NTSTATUS (*PDRIVER_INITIALIZE) (
        IN      PDRIVER_OBJECT      DriverObject,
        IN      PUNICODE_STRING     RegistryPath);
```

The input DriverObject represents the driver within the Windows Vista system. Its pointer allows the DriverEntry routine to set an appropriate entry point for its DriverUnload routine in the driver object. The RegistryPath input to the DriverEntry routine points to a counted Unicode string that specifies a path to the driver's registry key \Registry\Machine\System\CurrentControlSet\Services\KSECDD.

5.1.2 DriverUnload

It is the entry point for the driver's unload routine. The pointer to the routine is set by the DriverEntry routine in the DriverUnload field of the DriverObject when the driver initializes. An Unload routine is declared as follows:

```
        VOID (*PDRIVER_UNLOAD) (
                IN      PDRIVER_OBJECT      DriverObject);
```

When the driver is no longer needed, the Windows Vista Kernel is responsible to call the DriverUnload routine of the associated DriverObject.

5.2 Key Formatting

The following functions provide interfaces to the KSECDD.SYS module's key formatting functions.

5.2.1 FipsDesKey

```
        VOID FipsDesKey(
                DESTable *      pDesTable,
                UCHAR *         pbKey)
```

The FipsDesKey function formats a DES cryptographic session key into the form of a DESTable struct. It fills in the DESTable struct with the decrypt and encrypt key expansions. Its second parameter points to the DES key of DES_BLOCKLEN (8) bytes. FipsDesKey zeroizes its copy of the key before returning to the caller.

5.2.2 Fips3Des3Key

```
        VOID Fips3Des3Key(
                DES3TABLE *     pDES3Table,
                UCHAR *         pbKey)
```

The Fips3Des3Key function formats a Triple DES cryptographic session key into the form of a DES3Table struct. It fills in the DES3Table struct with the decrypt and encrypt key expansions. Its second parameter points to the Triple DES key of 3 * DES_BLOCKLEN (24) bytes. Fips3Des3Key zeroizes its copy of the key before returning to the caller.

5.3 Random Number Generation

5.3.1 FipsGenRandom

```
        BOOL FIPSGenRandom(
                IN OUT UCHAR *pb,
                IN      ULONG cb);
```

The FipsGenRandom function fills the buffer pb with cb random bytes produced using a FIPS 140-2 compliant pseudo random number generation algorithm. The algorithm is the SHS based RNG from FIPS 186-2. Internally, the function compares each 160 bits of the buffer with the next 160 bits. If they are the same, the function returns FALSE. The caller may optionally specify the initial 160 bits in the pb buffer for the initiation of the comparison. This initial 160 bit sequence is used only for the comparison algorithm and it is not intended as caller supplied random seed.
The seed sources are enumerated in the BCryptGenRandom() function description.

5.4 Data Encryption and Decryption

The following functions provide interfaces to the KSECDD.SYS module's data encryption and decryption functions.

5.4.1 FipsDes

```
VOID FipsDes(
        UCHAR *         pbOut,
        UCHAR *         pbIn,
        void *          pKey,
        int             iOp)
```

The FipsDes function encrypts or decrypts the input buffer pbIn using DES, putting the result into the output buffer pbOut. The operation (encryption or decryption) is specified with the iOp parameter. The pKey is a DESTable struct pointer returned by the FipsDesKey function. FipsDes zeroizes its copy of the DESTable struct before returning to the caller.

5.4.2 Fips3Des

```
VOID Fips3Des(
        UCHAR *         pbIn,
        UCHAR *         pbOut,
        void *          pKey,
        int             op)
```

The Fips3Des function encrypts or decrypts the input buffer pbIn using Triple DES, putting the result into the output buffer pbOut. The operation (encryption or decryption) is specified with the op parameter. The pkey is a DES3Table struct returned by the Fips3Des3Key function. Fips3Des zeroizes its copy of the DES3Table struct before returning to the caller.

5.4.3 FipsCBC

```
BOOL FipsCBC(
        ULONG EncryptionType,
        BYTE   *        output,
        BYTE   *        input,
        void   *        keyTable,
        int             op,
        BYTE   *        feedback)
```

The FipsCBC function encrypts or decrypts the input buffer input using CBC mode, putting the result into the output buffer output. The encryption algorithm (DES or Triple DES) to be used is specified with the EncryptionType parameter. The operation (encryption or decryption) is specified with the op parameter. If the EncryptionType parameter specifies Triple DES, the keyTable is a DES3Table struct returned by the Fips3Des3Key function. If the EncryptionType parameter specifies DES, the keyTable is a DESTable struct returned by the FipsDesKey function.
This function encrypts just one block at a time and assumes that the caller knows the algorithm block length and the buffers are of the correct length. Every time when the function is called, it zeroizes its copy of the DES3Table or DESTable struct before returning to the caller.

5.4.4 FipsBlockCBC

```
BOOL FipsBlockCBC(
        ULONG EncryptionType,
        BYTE   *        output,
        BYTE   *        input,
        ULONG           length,
        void   *        keyTable,
        int             op,
        BYTE   *        feedback)
```

Same as FipsCBC, the FipsBlockCBC function encrypts or decrypts the input buffer input using CBC mode, putting the result into the output buffer output. The encryption algorithm (DES or Triple DES) to be used

is specified with the EncryptionType parameter. The operation (encryption or decryption) is specified with the op parameter.
If the EncryptionType parameter specifies Triple DES, the keyTable is a DES3Table struct returned by the Fips3Des3Key function. If the EncryptionType parameter specifies DES, the keyTable is a DESTable struct returned by the FipsDesKey function.
This function can encrypt/decrypt more than one block at a time. The caller specifies the length in bytes of the input buffer in the "length" parameter. So the input/output buffer length is the arithmetic product of the number of blocks in the input/output buffer and the block length (8 bytes). When the length is 8 (i.e. one block of input buffer), FipsBlockCBC is the same as FipsCBC.
Every time when the function is called, it zeroizes its copy of the DES3Table or DESTable struct before returning to the caller.

5.5 Hashing

The following functions provide interfaces to the KSECDD.SYS module's hashing functions.

5.5.1 FipsSHAInit

```
void FipsSHAInit(
        A_SHA_CTX *    hash_context)
```

The FipsSHAInit function initiates the hashing of a stream of data. The output hash_context is used in subsequent hash functions.

5.5.2 FipsSHAUpdate

```
void FipsSHAUpdate(
        A_SHA_CTX *    hash_context,
        UCHAR *        pb,
        unsigned int   cb)
```

The FipsSHAUpdate function adds data pb of size cb to a specified hash object associated with the context hash_context. This function can be called multiple times to compute the hash on long data streams or discontinuous data streams. The FipsSHAFinal function must be called before retrieving the hash value.

5.5.3 FipsSHAFinal

```
void FipsSHAFinal (
        A_SHA_CTX *                       hash_context,
        unsigned char [A_SHA_DIGEST_LEN]  hash)
```

The FipsSHAFinal function computes the final hash of the data entered by the FipsSHAUpdate function. The hash is an array char of size A_SHA_DIGEST_LEN (20 bytes).

5.5.4 FipsHmacSHAInit

```
void FipsSHAInit(
        A_SHA_CTX *    pShaCtx
        UCHAR *        pKey,
        unsigned int   cbKey)
```

The FipsHmacSHAInit function initiates the HMAC hashing of a stream of data, with an input key provided via the pKey parameter. The size of the input key is specified in the cbKey parameter. If the key size is greater than 64 bytes, the key is hashed to a new key of size 20 bytes using SHA-1. The input key is EOR'ed with the ipad as required in the HMAC FIPS. The output pShaCtx is used in subsequent HMAC hashing functions. Every time when the function is called, it zeroizes its copy of the pKey before returning to the caller.

5.5.5 FipsHmacSHAUpdate

```
void FipsSHAUpdate(
        A_SHA_CTX *    pShaCtx,
        UCHAR *        pb,
        unsigned int   cb)
```

The FipsHmacSHAUpdate function adds data pb of size cb to a specified HMAC hashing object associated with the context pShaCtx. This function can be called multiple times to compute the HMAC hash on long data streams or discontinuous data streams. The FipsHmacSHAFinal function must be called before retrieving the final HMAC hash value.

5.5.6 FipsHmacSHAFinal

```
void FipsHmacSHAFinal (
A_SHA_CTX *   pShaCtx,
UCHAR *       pKey,
unsigned int  cbKey,
UCHAR  *      hash)
```

The FipsHmacSHAFinal function computes the final HMAC hash of the data entered by the FipsHmacSHAUpdate function, with an input key provided via the pKey parameter. The size of the input key is specified in the cbKey parameter. If the key size is greater than 64 bytes, the key is hashed to a new key of size 20 bytes using SHA-1. The input key is EOR'ed with the opad as required in the HMAC FIPS. It is the caller's responsibility to make sure that the input key used in FipsHmacSHAFinal is the same as the input key used in FipsHmacSHAInit. The final HMAC hash is an array char of size A_SHA_DIGEST_LEN (20 bytes). Every time when the function is called, it zeroizes its copy of the pKey before returning to the caller.

5.5.7 HmacMD5Init

```
void HmacMD5Init(
        MD5_CTX *      pMD5Ctx,
        UCHAR *        pKey,
        unsigned int   cbKey)
```

The HmacMD5Init function initiates the HMAC hashing of a stream of data, with an input key provided via the pKey parameter. The size of the input key is specified in the cbKey parameter. If the key size is greater than 64 bytes, the key is hashed to a new key of size 16 bytes using MD5 as required in the HMAC FIPS. The input key is EOR'ed with the ipad. The output pMD5Ctx is used in subsequent HMAC hashing functions. Every time when the function is called, it zeroizes its copy of the pKey before returning to the caller.

5.5.8 HmacMD5Update

```
void HmacMD5Update(
        MD5_CTX *      pMD5Ctx,
        UCHAR *        pb,
        unsigned int   cb)
```

The HmacMD5Update function adds data pb of size cb to a specified HMAC hashing object associated with the context pMD5Ctx. This function can be called multiple times to compute the HMAC hash on long data streams or discontinuous data streams. The HmacMD5Update function must be called before retrieving the final HMAC hash value.

5.5.9 HmacMD5Final

```
void HmacMD5Final(
        MD5_CTX *pMD5Ctx,
        UCHAR *pKey,
        unsigned int cbKey,
        UCHAR *pHash)
```

The HmacMD5Final function computes the final HMAC hash of the data entered by the HmacMD5Update function, with an input key provided via the pKey parameter. The size of the input key is specified in the cbKey parameter. If the key size is greater than 64 bytes, the key is hashed to a new key of size 16 bytes using MD5. The input key is EOR'ed with the opad as required in the HMAC FIPS. It is the caller's responsibility to make sure that the input key used in HmacMD5Final is the same as the input key used in HmacMD5Init. The final HMAC hash is an array char of size A_ MD5DIGESTLEN (16 bytes). Every time when the function is called, it zeroises its copy of the pKey before returning to the caller.

5.6 Acquiring a Table of Pointers to FipsXXX Functions

A kernel mode user of the KSECDD.SYS driver must be able to reference the FipsXXX functions before using them. The user needs to acquire the table of pointers to the FipsXXX functions from the KSECDD.SYS driver. The user accomplishes the table acquisition by building a Fips function table request irp (I/O request packet) and then sending the irp to the KSECDD.SYS diver via the IoCallDriver function. Further information on irp and IoCallDriver can be found on Microsoft Windows Vista Driver Development Kit.

5.7 Algorithm Providers and Properties

5.7.1 BCryptOpenAlgorithmProvider

```
NTSTATUS WINAPI BCryptOpenAlgorithmProvider(
        BCRYPT_ALG_HANDLE   *phAlgorithm,
        LPCWSTR pszAlgId,
        LPCWSTR pszImplementation,
        ULONG   dwFlags);
```

The BCryptOpenAlgorithmProvider() function has four parameters: algorithm handle output to the opened algorithm provider, desired algorithm ID input, an optional specific provider name input, and optional flags. This function loads and initializes a CNG provider for a given algorithm, and returns a handle to the opened algorithm provider on success.
Unless the calling function specifies the name of the provider, the default provider is used.
The calling function must pass the BCRYPT_ALG_HANDLE_HMAC_FLAG flag in order to use an HMAC function with a hash algorithm.

5.7.2 BCryptCloseAlgorithmProvider

```
NTSTATUS WINAPI BCryptCloseAlgorithmProvider(
        BCRYPT_ALG_HANDLE   hAlgorithm,
        ULONG   dwFlags);
```

This function closes an algorithm provider handle opened by a call to BCryptOpenAlgorithmProvider() function.

5.7.3 BCryptSetProperty

```
NTSTATUS WINAPI BCryptSetProperty(
        BCRYPT_HANDLE   hObject,
        LPCWSTR pszProperty,
        PUCHAR   pbInput,
        ULONG   cbInput,
        ULONG   dwFlags);
```

The BCryptSetProperty() function sets the value of a named property for a CNG object. The CNG object is a handle, the property name is a NULL terminated string, and the value of the property is a length-specified byte string.

5.7.4 BCryptGetProperty

```
NTSTATUS WINAPI BCryptGetProperty(
        BCRYPT_HANDLE   hObject,
        LPCWSTR pszProperty,
        PUCHAR   pbOutput,
        ULONG   cbOutput,
        ULONG   *pcbResult,
        ULONG   dwFlags);
```

The BCryptGetProperty() function retrieves the value of a named property for a CNG object. The CNG object is a handle, the property name is a NULL terminated string, and the value of the property is a length-specified byte string.

5.7.5 BCryptFreeBuffer

```
VOID WINAPI BCryptFreeBuffer(
        PVOID   pvBuffer);
```

Some of the CNG functions allocate memory on caller's behalf. The BCryptFreeBuffer() function frees memory that was allocated by such a CNG function.

5.8 Random Number Generation

5.8.1 BCryptGenRandom

```
NTSTATUS WINAPI BCryptGenRandom(
        BCRYPT_ALG_HANDLE   hAlgorithm,
        PUCHAR  pbBuffer,
        ULONG   cbBuffer,
        ULONG   dwFlags);
```

The BCryptGenRandom() function fills a buffer with random bytes. There are three random number generation algorithm:

- BCRYPT_RNG_ALGORITHM. This is the AES-256 counter mode based random generator as defined in SP800-90.
- BCRYPT_RNG_FIPS186_DSA_ALGORITHM. This is the FIPS 186-2 Regular random generator
- BCRYPT_RNG_DUAL_EC_ALGORITHM. This is the non-compliant Dual-EC DRBG based random generator as defined in SP800-90. The output from calls to this PRNG cannot be used for keys; however, it is available for generation of initialization vectors per FIPS 140-2.

During the function initialization, a seed is created from the output of an in-kernel random number generator. This RNG provides the necessary entropy for the RNGs available through this function.

5.9 Key and Key-Pair Generation

5.9.1 BCryptGenerateSymmetricKey

```
NTSTATUS WINAPI BCryptGenerateSymmetricKey(
        BCRYPT_ALG_HANDLE   hAlgorithm,
        BCRYPT_KEY_HANDLE   *phKey,
        PUCHAR   pbKeyObject,
        ULONG   cbKeyObject,
        PUCHAR   pbSecret,
        ULONG   cbSecret,
        ULONG   dwFlags);
```

The BCryptGenerateSymmetricKey() function generates a symmetric key object for use with a symmetric encryption algorithm from a supplied key value. The calling application must specify a handle to the algorithm provider created with the BCryptOpenAlgorithmProvider() function. The algorithm specified when the provider was created must support symmetric key encryption.

A key can also be generated by calling the FipsGenRandom() function in addition to the BCryptGenerateSymmetricKey() function.

5.9.2 BCryptGenerateKeyPair

```
NTSTATUS WINAPI BCryptGenerateKeyPair(
        BCRYPT_ALG_HANDLE   hAlgorithm,
        BCRYPT_KEY_HANDLE   *phKey,
        ULONG   dwLength,
        ULONG   dwFlags);
```

The BCryptGenerateKeyPair() function creates an empty public/private key pair. After creating a key using this function, call the BCryptSetProperty() function to set its properties. The key pair can be used only after BCryptFinalizeKeyPair() function is called.

5.9.3 BCryptFinalizeKeyPair

```
NTSTATUS WINAPI BCryptFinalizeKeyPair(
        BCRYPT_KEY_HANDLE   hKey,
        ULONG   dwFlags);
```

The BCryptFinalizeKeyPair() function completes a public/private key pair import or generation. The key pair cannot be used until this function has been called. After this function has been called, the BCryptSetProperty() function can no longer be used for this key.

5.9.4 BCryptDuplicateKey

```
NTSTATUS WINAPI BCryptDuplicateKey(
        BCRYPT_KEY_HANDLE   hKey,
        BCRYPT_KEY_HANDLE   *phNewKey,
        PUCHAR   pbKeyObject,
        ULONG   cbKeyObject,
        ULONG   dwFlags);
```

The BCryptDuplicateKey() function creates a duplicate of a symmetric key.

5.9.5 BCryptDestroyKey

```
NTSTATUS WINAPI BCryptDestroyKey(
        BCRYPT_KEY_HANDLE   hKey);
```

The BCryptDestroyKey() function destroys a key.

5.10 Key Entry and Output

5.10.1 BCryptImportKey

```
NTSTATUS WINAPI BCryptImportKey(
        BCRYPT_ALG_HANDLE hAlgorithm,
        BCRYPT_KEY_HANDLE hImportKey,
        LPCWSTR pszBlobType,
        BCRYPT_KEY_HANDLE *phKey,
        PUCHAR   pbKeyObject,
        ULONG   cbKeyObject,
        PUCHAR   pbInput,
        ULONG   cbInput,
        ULONG   dwFlags);
```

The BCryptImportKey() function imports a symmetric key from a key blob.

hAlgorithm [in] is the handle of the algorithm provider to import the key. This handle is obtained by calling the BCryptOpenAlgorithmProvider function.
hImportKey [in, out] is not currently used and should be NULL.

pszBlobType [in] is a null-terminated Unicode string that contains an identifier that specifies the type of BLOB that is contained in the *pbInput* buffer. *pszBlobType* can be one of BCRYPT_KEY_DATA_BLOB and BCRYPT_OPAQUE_KEY_BLOB.
phKey [out] is a pointer to a BCRYPT_KEY_HANDLE that receives the handle of the imported key that is used in subsequent functions that require a key, such as BCryptEncrypt. This handle must be released when it is no longer needed by passing it to the BCryptDestroyKey function.
pbKeyObject [out] is a pointer to a buffer that receives the imported key object. The *cbKeyObject* parameter contains the size of this buffer. The required size of this buffer can be obtained by calling the BCryptGetProperty function to get the BCRYPT_OBJECT_LENGTH property. This will provide the size of the key object for the specified algorithm. This memory can only be freed after the *phKey* key handle is destroyed.
cbKeyObject [in] is the size, in bytes, of the pbKeyObject buffer.
pbInput [in] is the address of a buffer that contains the key BLOB to import.
The *cbInput* parameter contains the size of this buffer.
The *pszBlobType* parameter specifies the type of key BLOB this buffer contains.
cbInput [in] is the size, in bytes, of the pbInput buffer.
dwFlags [in] is a set of flags that modify the behavior of this function. No flags are currently defined, so this parameter should be zero.

DES keys can also be imported into KSECDD.SYS via FipsDesKey(). DESTable struct can be exported out of KSECDD.SYS via FipsDesKey(). DESTable struct can be imported into KSECDD.SYS via FipsDes() or FipsCBC().
Triple DES keys can be imported into KSECDD.SYS via Fips3Des3Key(). DES3Table struct can be exported out of KSECDD.SYS via Fips3Des3Key(). DES3Table struct can be imported into KSECDD.SYS via Fips3Des() or FipsCBC().
HMAC keys can be imported into KSECDD.SYS via FipsHmacSHAInit and FipsHmacSHAFinal.

5.10.2 BCryptImportKeyPair

```
NTSTATUS WINAPI BCryptImportKeyPair(
        BCRYPT_ALG_HANDLE hAlgorithm,
        BCRYPT_KEY_HANDLE hImportKey,
        LPCWSTR pszBlobType,
        BCRYPT_KEY_HANDLE *phKey,
        PUCHAR   pbInput,
        ULONG    cbInput,
        ULONG    dwFlags);
```

The BCryptImportKeyPair() function is used to import a public/private key pair from a key blob.
hAlgorithm [in] is the handle of the algorithm provider to import the key. This handle is obtained by calling the BCryptOpenAlgorithmProvider function.
hImportKey [in, out] is not currently used and should be NULL.
pszBlobType [in] is a null-terminated Unicode string that contains an identifier that specifies the type of BLOB that is contained in the pbInput buffer. This can be one of the following values: BCRYPT_DH_PRIVATE_BLOB, BCRYPT_DH_PUBLIC_BLOB, BCRYPT_PUBLIC_KEY_BLOB, BCRYPT_PRIVATE_KEY_BLOB, BCRYPT_RSAPRIVATE_BLOB, BCRYPT_RSAPUBLIC_BLOB, LEGACY_DH_PUBLIC_BLOB, LEGACY_DH_PRIVATE_BLOB, LEGACY_RSAPRIVATE_BLOB, LEGACY_RSAPUBLIC_BLOB.
phKey [out] is a pointer to a BCRYPT_KEY_HANDLE that receives the handle of the imported key. This handle is used in subsequent functions that require a key, such as BCryptSignHash. This handle must be released when it is no longer needed by passing it to the BCryptDestroyKey function.
pbInput [in] is the address of a buffer that contains the key BLOB to import. The cbInput parameter contains the size of this buffer. The pszBlobType parameter specifies the type of key BLOB this buffer contains.
cbInput [in] contains the size, in bytes, of the pbInput buffer.

dwFlags [in] is a set of flags that modify the behavior of this function. This can be zero or the following value: BCRYPT_NO_KEY_VALIDATION.

5.10.3 BCryptExportKey

```
NTSTATUS WINAPI BCryptExportKey(
        BCRYPT_KEY_HANDLE   hKey,
        BCRYPT_KEY_HANDLE   hExportKey,
        LPCWSTR pszBlobType,
        PUCHAR   pbOutput,
        ULONG   cbOutput,
        ULONG   *pcbResult,
        ULONG   dwFlags);
```

The BCryptExportKey() function exports a key to a memory blob that can be persisted for later use.
hExportKey [in, out] is not currently used and should be set to NULL.
pszBlobType [in] is a null-terminated Unicode string that contains an identifier that specifies the type of BLOB to export. This can be one of the following values: BCRYPT_DH_PRIVATE_BLOB, BCRYPT_DH_PUBLIC_BLOB, BCRYPT_ECCPRIVATE_BLOB, BCRYPT_ECCPUBLIC_BLOB, BCRYPT_KEY_DATA_BLOB, BCRYPT_OPAQUE_KEY_BLOB, BCRYPT_PUBLIC_KEY_BLOB, BCRYPT_PRIVATE_KEY_BLOB, BCRYPT_RSAPUBLIC_BLOB, LEGACY_DH_PRIVATE_BLOB, LEGACY_DH_PUBLIC_BLOB, LEGACY_RSAPUBLIC_BLOB.
pbOutput is the address of a buffer that receives the key BLOB. The cbOutput parameter contains the size of this buffer. If this parameter is NULL, this function will place the required size, in bytes, in the ULONG pointed to by the pcbResult parameter.
cbOutput [in] contains the size, in bytes, of the pbOutput buffer.
pcbResult [out] is a pointer to a ULONG that receives the number of bytes that were copied to the pbOutput buffer. If the pbOutput parameter is NULL, this function will place the required size, in bytes, in the ULONG pointed to by this parameter.
dwFlags [in] is a set of flags that modify the behavior of this function. No flags are defined for this function.

5.11 Encryption and Decryption

5.11.1 BCryptEncrypt

```
NTSTATUS WINAPI BCryptEncrypt(
        BCRYPT_KEY_HANDLE hKey,
        PUCHAR   pbInput,
        ULONG   cbInput,
        VOID    *pPaddingInfo,
        PUCHAR   pbIV,
        ULONG   cbIV,
        PUCHAR   pbOutput,
        ULONG   cbOutput,
        ULONG   *pcbResult,
        ULONG   dwFlags);
```

The BCryptEncrypt() function encrypts a block of data of given length.
hKey [in, out] is the handle of the key to use to encrypt the data. This handle is obtained from one of the key creation functions, such as BCryptGenerateSymmetricKey, BCryptGenerateKeyPair, or BCryptImportKey.
pbInput [in] is the address of a buffer that contains the plaintext to be encrypted. The cbInput parameter contains the size of the plaintext to encrypt. For more information, see Remarks.
cbInput [in] is the number of bytes in the pbInput buffer to encrypt.

pPaddingInfo [in, optional] is a pointer to a structure that contains padding information. The actual type of structure this parameter points to depends on the value of the dwFlags parameter. This parameter is only used with asymmetric keys and must be NULL otherwise.
pbIV [in, out, optional] is the address of a buffer that contains the initialization vector (IV) to use during encryption. The cbIV parameter contains the size of this buffer. This function will modify the contents of this buffer. If you need to reuse the IV later, make sure you make a copy of this buffer before calling this function. This parameter is optional and can be NULL if no IV is used. The required size of the IV can be obtained by calling the BCryptGetProperty function to get the BCRYPT_BLOCK_LENGTH property. This will provide the size of a block for the algorithm, which is also the size of the IV.
cbIV [in] contains the size, in bytes, of the pbIV buffer.
pbOutput [out, optional] is the address of a buffer that will receive the ciphertext produced by this function. The cbOutput parameter contains the size of this buffer. For more information, see Remarks. If this parameter is NULL, this function will calculate the size needed for the ciphertext and return the size in the location pointed to by the pcbResult parameter.
cbOutput [in] contains the size, in bytes, of the pbOutput buffer. This parameter is ignored if the pbOutput parameter is NULL.
pcbResult [out] is a pointer to a ULONG variable that receives the number of bytes copied to the pbOutput buffer. If pbOutput is NULL, this receives the size, in bytes, required for the ciphertext.
dwFlags [in] is a set of flags that modify the behavior of this function. The allowed set of flags depends on the type of key specified by the hKey parameter. If the key is a symmetric key, this can be zero or the following value: BCRYPT_BLOCK_PADDING. If the key is an asymmetric key, this can be one of the following values: BCRYPT_PAD_NONE, BCRYPT_PAD_OAEP, BCRYPT_PAD_PKCS1.

5.11.2 BCryptDecrypt

```
NTSTATUS WINAPI BCryptDecrypt(
        BCRYPT_KEY_HANDLE   hKey,
        PUCHAR   pbInput,
        ULONG   cbInput,
        VOID    *pPaddingInfo,
        PUCHAR   pbIV,
        ULONG   cbIV,
        PUCHAR   pbOutput,
        ULONG   cbOutput,
        ULONG   *pcbResult,
        ULONG   dwFlags);
```

The BCryptDecrypt() function decrypts a block of data of given length.
hKey [in, out] is the handle of the key to use to decrypt the data. This handle is obtained from one of the key creation functions, such as BCryptGenerateSymmetricKey, BCryptGenerateKeyPair, or BCryptImportKey.
pbInput [in] is the address of a buffer that contains the ciphertext to be decrypted. The cbInput parameter contains the size of the ciphertext to decrypt. For more information, see Remarks.
cbInput [in] is the number of bytes in the pbInput buffer to decrypt.
pPaddingInfo [in, optional] is a pointer to a structure that contains padding information. The actual type of structure this parameter points to depends on the value of the dwFlags parameter. This parameter is only used with asymmetric keys and must be NULL otherwise.
pbIV [in, out, optional] is the address of a buffer that contains the initialization vector (IV) to use during decryption. The cbIV parameter contains the size of this buffer. This function will modify the contents of this buffer. If you need to reuse the IV later, make sure you make a copy of this buffer before calling this function. This parameter is optional and can be NULL if no IV is used. The required size of the IV can be obtained by calling the BCryptGetProperty function to get the BCRYPT_BLOCK_LENGTH property. This will provide the size of a block for the algorithm, which is also the size of the IV.
cbIV [in] contains the size, in bytes, of the pbIV buffer.

pbOutput [out, optional] is the address of a buffer to receive the plaintext produced by this function. The cbOutput parameter contains the size of this buffer. For more information, see Remarks.
If this parameter is NULL, this function will calculate the size required for the plaintext and return the size in the location pointed to by the pcbResult parameter.
cbOutput [in] is the size, in bytes, of the pbOutput buffer. This parameter is ignored if the pbOutput parameter is NULL.
pcbResult [out] is a pointer to a ULONG variable to receive the number of bytes copied to the pbOutput buffer. If pbOutput is NULL, this receives the size, in bytes, required for the plaintext.
dwFlags [in] is a set of flags that modify the behavior of this function. The allowed set of flags depends on the type of key specified by the hKey parameter. If the key is a symmetric key, this can be zero or the following value: BCRYPT_BLOCK_PADDING. If the key is an asymmetric key, this can be one of the following values: BCRYPT_PAD_NONE, BCRYPT_PAD_OAEP, BCRYPT_PAD_PKCS1.

5.12 Hashing and HMAC

5.12.1 BCryptCreateHash

```
NTSTATUS WINAPI BCryptCreateHash(
        BCRYPT_ALG_HANDLE   hAlgorithm,
        BCRYPT_HASH_HANDLE  *phHash,
        PUCHAR   pbHashObject,
        ULONG   cbHashObject,
        PUCHAR   pbSecret,
        ULONG   cbSecret,
        ULONG   dwFlags);
```

The BCryptCreateHash() function creates a hash object with an optional key. The optional key is used for HMAC type keyed-hash functions.
hAlgorithm [in, out] is the handle of an algorithm provider created by using the BCryptOpenAlgorithmProvider function. The algorithm that was specified when the provider was created must support the hash interface.
phHash [out] is a pointer to a BCRYPT_HASH_HANDLE value that receives a handle that represents the hash object. This handle is used in subsequent hashing functions, such as the BCryptHashData function. When you have finished using this handle, release it by passing it to the BCryptDestroyHash function.
pbHashObject [out] is a pointer to a buffer that receives the hash object. The cbHashObject parameter contains the size of this buffer. The required size of this buffer can be obtained by calling the BCryptGetProperty function to get the BCRYPT_OBJECT_LENGTH property. This will provide the size of the hash object for the specified algorithm. This memory can only be freed after the hash handle is destroyed.
cbHashObject [in] contains the size, in bytes, of the pbHashObject buffer.
pbSecret [in, optional] is a pointer to a buffer that contains the key to use for the hash. The cbSecret parameter contains the size of this buffer. If no key should be used with the hash, set this parameter to NULL. This key only applies to keyed hash algorithms, like Hash-Based Message Authentication Code (HMAC).
cbSecret [in, optional] contains the size, in bytes, of the pbSecret buffer. If no key should be used with the hash, set this parameter to zero.
dwFlags [in] is not currently used and must be zero.

5.12.2 BCryptHashData

```
NTSTATUS WINAPI BCryptHashData(
        BCRYPT_HASH_HANDLE  hHash,
        PUCHAR   pbInput,
        ULONG   cbInput,
        ULONG   dwFlags);
```

The BCryptHashData() function performs a one way hash on a data buffer. Call the BCryptFinishHash() function to finalize the hashing operation to get the hash result.

5.12.3 BCryptDuplicateHash

```
NTSTATUS WINAPI BCryptDuplicateHash(
        BCRYPT_HASH_HANDLE  hHash,
        BCRYPT_HASH_HANDLE  *phNewHash,
        PUCHAR   pbHashObject,
        ULONG   cbHashObject,
        ULONG   dwFlags);
```

The BCryptDuplicateHash()function duplicates an existing hash object. The duplicate hash object contains all state and data that was hashed to the point of duplication.

5.12.4 BCryptFinishHash

```
NTSTATUS WINAPI BCryptFinishHash(
        BCRYPT_HASH_HANDLE hHash,
        PUCHAR   pbOutput,
        ULONG   cbOutput,
        ULONG   dwFlags);
```

The BCryptFinishHash() function retrieves the hash value for the data accumulated from prior calls to BCryptHashData() function.

5.12.5 BCryptDestroyHash

```
NTSTATUS WINAPI BCryptDestroyHash(
        BCRYPT_HASH_HANDLE  hHash);
```

The BCryptDestroyHash() function destroys a hash object.

5.13 Signing and Verification

5.13.1 BCryptSignHash

```
NTSTATUS WINAPI BCryptSignHash(
        BCRYPT_KEY_HANDLE   hKey,
        VOID    *pPaddingInfo,
        PUCHAR   pbInput,
        ULONG   cbInput,
        PUCHAR   pbOutput,
        ULONG   cbOutput,
        ULONG   *pcbResult,
        ULONG   dwFlags);
```

The BCryptSignHash() function creates a signature of a hash value.
hKey [in] is the handle of the key to use to sign the hash.
pPaddingInfo [in, optional] is a pointer to a structure that contains padding information. The actual type of structure this parameter points to depends on the value of the dwFlags parameter. This parameter is only used with asymmetric keys and must be NULL otherwise.
pbInput [in] is a pointer to a buffer that contains the hash value to sign. The cbInput parameter contains the size of this buffer.
cbInput [in] is the number of bytes in the pbInput buffer to sign.
pbOutput [out] is the address of a buffer to receive the signature produced by this function. The cbOutput parameter contains the size of this buffer. If this parameter is NULL, this function will calculate the size required for the signature and return the size in the location pointed to by the pcbResult parameter.
cbOutput [in] is the size, in bytes, of the pbOutput buffer. This parameter is ignored if the pbOutput parameter is NULL.

pcbResult [out] is a pointer to a ULONG variable that receives the number of bytes copied to the pbOutput buffer. If pbOutput is NULL, this receives the size, in bytes, required for the signature.
dwFlags [in] is a set of flags that modify the behavior of this function. The allowed set of flags depends on the type of key specified by the hKey parameter. If the key is a symmetric key, this parameter is not used and should be set to zero. If the key is an asymmetric key, this can be one of the following values: BCRYPT_PAD_PKCS1, BCRYPT_PAD_PSS.

5.13.2 BCryptVerifySignature

```
NTSTATUS WINAPI BCryptVerifySignature(
        BCRYPT_KEY_HANDLE   hKey,
        VOID    *pPaddingInfo,
        PUCHAR   pbHash,
        ULONG   cbHash,
        PUCHAR   pbSignature,
        ULONG   cbSignature,
        ULONG   dwFlags);
```

The BCryptVerifySignature() function verifies that the specified signature matches the specified hash.
hKey [in] is the handle of the key to use to decrypt the signature. This must be an identical key or the public key portion of the key pair used to sign the data with the BCryptSignHash function.
pPaddingInfo [in, optional] is a pointer to a structure that contains padding information. The actual type of structure this parameter points to depends on the value of the *dwFlags* parameter. This parameter is only used with asymmetric keys and must be NULL otherwise.
pbHash [in] is the address of a buffer that contains the hash of the data. The cbHash parameter contains the size of this buffer.
cbHash [in] is the size, in bytes, of the pbHash buffer.
pbSignature [in] is the address of a buffer that contains the signed hash of the data. The BCryptSignHash function is used to create the signature. The *cbSignature* parameter contains the size of this buffer.
cbSignature [in] is the size, in bytes, of the pbSignature buffer. The BCryptSignHash function is used to create the signature.

5.14 Secret Agreement and Key Derivation

5.14.1 BCryptSecretAgreement

```
NTSTATUS WINAPI BCryptSecretAgreement(
        BCRYPT_KEY_HANDLE       hPrivKey,
        BCRYPT_KEY_HANDLE       hPubKey,
        BCRYPT_SECRET_HANDLE    *phAgreedSecret,
        ULONG                   dwFlags);
```

The BCryptSecretAgreement() function creates a secret agreement value from a private and a public key. This function is used with Diffie-Hellman (DH) and Elliptic Curve Diffie-Hellman (ECDH) algorithms.
hPrivKey [in] The handle of the private key to use to create the secret agreement value.
hPubKey [in] The handle of the public key to use to create the secret agreement value.
phSecret [out] A pointer to a BCRYPT_SECRET_HANDLE that receives a handle that represents the secret agreement value. This handle must be released by passing it to the BCryptDestroySecret function when it is no longer needed.
dwFlags [in] A set of flags that modify the behavior of this function. This can be zero or the following value: KDF_USE_SECRET_AS_HMAC_KEY_FLAG.

5.14.2 BCryptDeriveKey

```
NTSTATUS WINAPI BCryptDeriveKey(
        BCRYPT_SECRET_HANDLE hSharedSecret,
        LPCWSTR              pwszKDF,
        BCryptBufferDesc     *pParameterList,
        PUCHAR pbDerivedKey,
        ULONG           cbDerivedKey,
        ULONG           *pcbResult,
        ULONG           dwFlags);
```

The BCryptDeriveKey() function derives a key from a secret agreement value.
hSharedSecret [in, optional] is the secret agreement handle to create the key from. This handle is obtained from the BCryptSecretAgreement function.
pwszKDF [in] is a pointer to a null-terminated Unicode string that contains an object identifier (OID) that identifies the key derivation function (KDF) to use to derive the key. This can be one of the following strings: BCRYPT_KDF_HASH (parameters in pParameterList: KDF_HASH_ALGORITHM, KDF_SECRET_PREPEND, KDF_SECRET_APPEND), BCRYPT_KDF_HMAC (parameters in pParameterList: KDF_HASH_ALGORITHM, KDF_HMAC_KEY, KDF_SECRET_PREPEND, KDF_SECRET_APPEND), BCRYPT_KDF_TLS_PRF (parameters in pParameterList: KDF_TLS_PRF_LABEL, KDF_TLS_PRF_SEED).
pParameterList [in, optional] is the address of a BCryptBufferDesc structure that contains the KDF parameters. This parameter is optional and can be NULL if it is not needed.
pbDerivedKey [out, optional] is the address of a buffer that receives the key. The cbDerivedKey parameter contains the size of this buffer. If this parameter is NULL, this function will place the required size, in bytes, in the ULONG pointed to by the pcbResult parameter.
cbDerivedKey [in] contains the size, in bytes, of the pbDerivedKey buffer.
pcbResult [out] is a pointer to a ULONG that receives the number of bytes that were copied to the pbDerivedKey buffer. If the pbDerivedKey parameter is NULL, this function will place the required size, in bytes, in the ULONG pointed to by this parameter.
dwFlags [in] is a set of flags that modify the behavior of this function. This can be zero or the following value.

5.14.3 BCryptDestroySecret

```
        NTSTATUS WINAPI BCryptDestroySecret(
                BCRYPT_SECRET_HANDLE    hSecret);
```

The BCryptDestroySecret() function destroys a secret agreement handle that was created by using the BCryptSecretAgreement() function.

5.15 Configuration

These are not cryptographic functions. They are used to configure cryptographic providers on the system, and are provided for informational purposes. Please see http://msdn.microsoft.com for details.

Function Name	**Description**
BCryptAddContextFunction	Adds a function (algorithm or cipher-suite) to a context function list.
BCryptAddContextFunctionProvider	Adds a provider to a context function provider list.
BCryptConfigureContext	Configures a context.
BCryptConfigureContextFunction	Configures a context function.
BCryptConfigureContextFunction	Configures a context function.
BCryptCreateContext	Creates a new configuration context.
BCryptDeleteContext	Deletes a configuration context.
BCryptEnumAlgorithms	Enumerates the algorithms for a given set of operations.
BCryptEnumContextFunctionProviders	Enumerates the providers in a context function provider list.
BCryptEnumContextFunctions	Enumerates the functions (algorithms or suites) in a context function list.
BCryptEnumContexts	Enumerates the configuration contexts in the specified table.
BCryptEnumProviders	Returns a list of providers for a given algorithm.
BCryptEnumRegisteredProviders	Enumerates the providers currently registered on the local machine.
BCryptQueryContextConfiguration	Queries the current configuration of a context.

BCryptQueryContextFunctionConfiguration	Queries the current configuration of a context function.
BCryptQueryContextFunctionProperty	Queries the current value of a context function property.
BCryptQueryProviderRegistration	Retrieves registration information for a provider.
BCryptRegisterConfigChangeNotify	This API differs slightly between User-Mode and Kernel-Mode.
BCryptRegisterProvider	Registers a provider for usage on the local machine.
BCryptRemoveContextFunction	Removes a function (algorithm or cipher-suite) from a context function list.
BCryptRemoveContextFunctionProvider	Removes a provider from a context function provider list.
BCryptResolveProviders	This is the main API in Crypto configuration. It resolves queries against the set of providers currently registered on the local system and the configuration information specified in the machine and domain configuration tables, returning an ordered list of references to one or more providers matching the specified criteria.
BCryptSetContextFunctionProperty	Creates, modifies, or deletes a context function property.
BCryptUnregisterConfigChangeNotify	This API differs slightly between User-Mode and Kernel-Mode.
BCryptUnregisterProvider	Removes provider registration information from the local machine.

5.16 Other Interfaces

The following table lists other non-approved APIs exported from KSECDD.SYS crypto module.

Function Name	**Description**
InitSecurityInterfaceW	
AcquireCredentialsHandleW	
QueryCredentialsAttributesW	
SetCredentialsAttributesW	
AddCredentialsW	
FreeCredentialsHandle	
InitializeSecurityContextW	
AcceptSecurityContext	
ImpersonateSecurityContext	
RevertSecurityContext	
DeleteSecurityContext	
ApplyControlToken	
CompleteAuthToken	
QueryContextAttributesW	
FreeContextBuffer	
MakeSignature	
VerifySignature	
SealMessage = EncryptMessage	
UnsealMessage = DecryptMessage	
MapSecurityError	
GetSecurityUserInfo	
EnumerateSecurityPackagesW	
QuerySecurityContextToken	
QuerySecurityPackageInfoW	
ExportSecurityContext	
ImportSecurityContextW	
EfsGenerateKey	

GenerateDirEfs EfsDecryptFek GenerateSessionKey SecSetPagingMode SecMakeSPN SecMakeSPNEx SecMakeSPNEx2 SecLookupAccountName SecLookupAccountSid SecLookupWellKnownSid CredMarshalTargetInfo KSecValidateBuffer LsaEnumerateLogonSessions LsaGetLogonSessionData KSecRegisterSecurityProvider BCryptRegisterConfigChangeNotify BCryptUnregisterConfigChangeNotify BCryptResolveProviders BCryptGetFipsAlgorithmMode SslDecryptPacket SslEncryptPacket SslExportKey SslFreeObject SslImportKey SslLookupCipherSuiteInfo SslOpenProvider	

6 Operational Environment

KSECDD.SYS services are available to all kernel mode components, which are part of the TCB.

7 Cryptographic Key Management

KSECDD.SYS crypto module manages keys in the following manner.

7.1 Cryptographic Keys, CSPs, and SRDIs

The KSECDD.SYS crypto module contains the following security relevant data items:

Security Relevant Data Item	**SRDI Description**
Symmetric encryption/decryption keys	Keys used for AES or TDES encryption/decryption.
HMAC keys	Keys used for HMAC-SHA1, HMAC-SHA256, HMAC-SHA384, and HMAC-SHA512
ECDSA Public Keys	Keys used for the verification of ECDSA digital signatures
ECDSA Private Keys	Keys used for the calculation of ECDSA digital signatures

RSA Public Keys	Keys used for the verification of RSA digital signatures
RSA Private Keys	Keys used for the calculation of RSA digital signatures
DH Public and Private values	Public and private values used for Diffie-Hellman key establishment.
ECDH Public and Private values	Public and private values used for EC Diffie-Hellman key establishment.

7.2 Access Control Policy

The KSECDD.SYS 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.

KSECDD.SYS crypto module SRDI/Service Access Policy	Security Relevant Data Item	Symmetric encryption/decryption keys	HMAC keys	ECDSA public keys	ECDSA Private keys	RSA Public Keys	RSA Private Keys	DH Public and Private values	ECDH Public and Private values
Service									
Cryptographic Module Power Up and Power Down									
Key Formatting		w							
Random Number Generation									
Data Encryption and Decryption		x							
Hashing			x / w						
Acquiring a Table of Pointers to FipsXXX Functions									
Algorithm Providers and Properties									
Key and Key-Pair Generation		w / d	w / d	w / d	w / d	w / d	w / d	w / d	w / d
Key Entry and Output		r / w	r / w	r / w	r / w	r / w	r / w	r / w	r / w
Signing and Verification				x	x	x	x		
Secret Agreement and Key Derivation								x	x

7.3 Key Material

When KSECDD.SYS is loaded in the Windows Vista Operating System kernel, no keys exist within it. A kernel module is responsible for importing keys into KSECDD.SYS or using KSECDD.SYS's functions to generate keys.

7.4 Key Generation

KSECDD.SYS can create and use keys for the following algorithms: RSA, DH, ECDH, ECDSA, RC2, RC4, DES, Triple-DES, AES, and HMAC.
Random keys can be generated by calling the BCryptGenerateSymmetricKey() and BCryptGenerateKeyPair() functions. Random data generated by the BCryptGenRandom() function is provided to BCryptGenerateSymmetricKey() function to generate symmetric keys. DES, Triple-DES, AES, RSA, ECDSA, DH, and ECDH keys and key-pairs are generated following the techniques given in 5.8.1. FipsGenRandom() function can also generate DES, Triple-DES, and HMAC keys.

7.5 Key Establishment

KSECDD.SYS can use FIPS approved Diffie-Hellman key agreement (DH), Elliptic Curve Diffie-Hellman key agreement (ECDH), and manual methods to establish keys.
KSECDD.SYS can use the following FIPS approved key derivation functions (KDF) from the common secret that is established during the execution of DH and ECDH key agreement algorithms:

- BCRYPT_KDF_HMAC. This KDF supports FIPS approved IPsec IKE v1 key derivation as specified in FIPS 140-2 Implementation Guidance.
- BCRYPT_KDF_TLS_PRF. This KDF supports FIPS approved SSLv3.1 and TLS v1.0 key derivation as specified in FIPS 140-2 Implementation Guidance.

7.6 Key Entry and Output

Keys can be both exported and imported out of and into KSECDD.SYS via BCryptExportKey(), BCryptImportKey(), and BCryptImportKeyPair() functions.
Symmetric key entry and output can also be done by exchanging keys using the recipient's asymmetric public key via BCryptSecretAgreement() and BCryptDeriveKey() functions.
DES keys can also be imported into KSECDD.SYS via FipsDesKey(). DESTable struct can be exported out of KSECDD.SYS via FipsDesKey(). DESTable struct can be imported into KSECDD.SYS via FipsDes() or FipsCBC().
Triple DES keys can be imported into KSECDD.SYS via Fips3Des3Key(). DES3Table struct can be exported out of KSECDD.SYS via Fips3Des3Key(). DES3Table struct can be imported into KSECDD.SYS via Fips3Des() or FipsCBC().
HMAC keys can be imported into KSECDD.SYS via FipsHmacSHAInit and FipsHmacSHAFinal.

Exporting the RSA private key by supplying a blob type of BCRYPT_PRIVATE_KEY_BLOB, BCRYPT_RSAFULLPRIVATE_BLOB, or BCRYPT_RSAPRIVATE_BLOB to BCryptExportKey() is not allowed in FIPS mode.

7.7 Key Storage

KSECDD.SYS does not provide persistent storage of keys.

7.8 Key Archival

KSECDD.SYS does not directly archive cryptographic keys. A user may choose to export a cryptographic key (cf. "Key Entry and Output" above), but management of the secure archival of that key is the responsibility of the user. All key copies inside KSECDD.SYS are destroyed and their memory location zeroized after used. It is the caller's responsibility to maintain the security of DES, Triple DES and HMAC keys when the keys are outside KSECDD.SYS.

7.9 Key Zeroization

All keys are destroyed and their memory location zeroized when the Authenticated User calls BCryptDestroyKey() or BCryptDestroySecret() on that key handle.
All DES and Triple DES key copies, their associated DESTable and DES3Table struct copies, and HMAC key copies inside KSECDD.SYS are destroyed and their memory location zeroized after they have been used in FipsDes, Fips3Des, or FipsCBC.

8 Self-Tests

KSECDD.SYS performs the following power-on (start up) self-tests when a caller calls its DriverEntry.

- SHA-1 hash Known Answer Test
- HMAC-SHA-1, HMAC-SHA-256, HMAC-SHA-384, and HMAC-SHA-512 Known Answer Test .
- Triple-DES encrypt/decrypt EBC Known Answer Test
- Triple-DES encrypt/decrypt CBC Known Answer Test
- AES-128, AES-192, AES-256 encrypt/decrypt EBC Known Answer Test
- AES-128, AES-192, AES-256 encrypt/decrypt CBC Known Answer Test
- AES-128, AES-192, AES-256 encrypt/decrypt CFB with 8-bit feedback Known Answer Test
- AES-128, AES-192, AES-256 GMAC Known Answer Test
- AES-128, AES-192, AES-256 encrypt/decrypt CCM Known Answer Test
- RSA sign/verify test
- DH secret agreement Known Answer Test
- ECDSA sign/verify test
- ECDH secret agreement Known Answer Test
- FIPS 186-2 General Purpose Known Answer Test
- SP800-90 AES-256 based counter mode random generator Known Answer Test

In all cases for any failure of a power-on (start up) self-test, KSECDD.SYS DriverEntry will fail to return the STATUS_SUCCESS status to its caller. The only way to recover from the failure of a power-on (start up) self-test is to attempt to invoke DriverEntry, which will rerun the self-tests, and will only succeed if the self-tests passes.

KSECDD.SYS performs pair-wise consistency checks upon each invocation of RSA, ECDH, and ECDSA key-pair generation and import as defined in FIPS 140-2. KSECDD.SYS also performs a continuous RNG test on its implemented FIPS 186-2 General Purpose RNG.

9 Design Assurance

The KSECDD.SYS crypto module is part of the overall Windows Vista operating system, which is a product family that has gone through and is continuously going through the Common Criteria or equivalent Certification under US NIAP CCEVS since Windows NT 3.5. The certification provides the necessary design assurance.
The KSECDD.SYS is installed and started as part of the Windows Vista operating system.

10 Additional Details

For the latest information on Windows Vista, check out the Microsoft web site at http://www.microsoft.com.

CHANGE HISTORY			
AUTHOR	**DATE**	**VERSION**	**COMMENT**
Tolga Acar	8/8/2006	1.0	First Draft, based on combined BCRYPT.DLL and KSECDD.SYS security policy document.
Tolga Acar	8/11/2006	1.1	Updated software integrity check, adding bootmgr.
Tolga Acar	9/19/2006	1.2	Added function prototypes. Updated pseudo random number entropy sources.
Tolga Acar	10/11/2006	1.3	Seed sources updated in BCryptGenRandom. Updated function descriptions. Other editorial updates.
Shivaram Mysore	5/2/2007	1.4	Updated self test information, Security Policy section for FIPS approved/non-approved algorithms, fixed typos and language
Stefan Santesson	2/15/2008	1.5	Added technical updates related to SP1 and WS2K8 and merged CMVP review comments