Microsoft Windows 7 Kernel Mode Cryptographic Primitives Library (cng.sys) Security Policy Document

Microsoft Windows 7 Operating System

FIPS 140-2 Security Policy Document

This document specifies the security policy for the Microsoft Kernel Mode Cryptographic Primitives Library (CNG.SYS) as described in FIPS PUB 140-2.

January 16, 2013

Document Version: 2.2

1 Cryptographic Module Specification

Microsoft Kernel Mode Cryptographic Primitives Library (CNG.SYS) is a FIPS 140-2 Level 1 compliant, general purpose, software-based, cryptographic module residing at kernel mode level of Windows 7 operating system. CNG.SYS (versions 6.1.7600.16385, 6.1.7600.16915, 6.1.7600.21092, 6.1.7601.17514, 6.1.7601.17725, 6.1.7601.17919, 6.1.7601.21861, and 6.1.7601.22076) runs as a kernel mode export driver, and provides cryptographic services, through their documented interfaces, to Windows 7 kernel components.

The CNG.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 7 kernel mode components can use general-purpose FIPS 140-2 Level 1 compliant cryptography in CNG.SYS.

1.1 Cryptographic Boundary

The Windows 7 kernel mode CNG.SYS consists of a single kernel mode export driver (SYS). The cryptographic boundary for CNG.SYS is defined as the enclosure of the computer system, on which CNG.SYS is to be executed. The physical configuration of CNG.SYS, as defined in FIPS-140-2, is multi-chip standalone

2 Security Policy

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

- CNG.SYS is supported on Windows 7 and Windows 7 SP1.
- CNG.SYS operates in FIPS mode of operation only when used with the FIPS approved version of Windows 7 Winload OS Loader (winload.exe) validated to FIPS 140-2 under Cert. #1326 operating in FIPS mode
- Windows 7 is an operating system supporting a “single user” mode where there is only one interactive user during a logon session.
- CNG.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.
- CNG.SYS operates in its FIPS mode of operation only when one of the following DWORD registry values is set to 1:
    - HKLM\SYSTEM\CurrentControlSet\Control\Lsa\FIPSAlgorithmPolicy\Enabled
    - HKLM\SYSTEM\CurrentControlSet\Policies\Microsoft\Cryptography\Configuration\SelfTestAlgorithms
- All users assume either the User or Cryptographic Officer roles.
- CNG.SYS provides no authentication of users. Roles are assumed implicitly. The authentication provided by the Windows 7 operating system is not in the scope of the validation.
- All cryptographic services implemented within CNG.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.
- CNG.SYS implements the following FIPS-140-2 Approved algorithms.
    - SHA-1, SHA-256, SHA-384, SHA-512 hash (Cert. #1081)
    - SHA-1, SHA-256, SHA-384, SHA-512 HMAC (Cert. #677)
    - Triple-DES (2 key and 3 key) in ECB, CBC, and CFB8 modes (Cert. #846)
    - AES-128, AES-192, AES-256 in ECB, CBC, and CFB8 modes (Cert. #1168)
    - AES-128, AES-192 and AES-256 in CCM mode (Cert. #1178)
    - AES-128, AES-192 and AES-256 in GCM mode (AES Cert. #1168, vendor-affirmed)
    - AES-128, AES-192 and AES-256 in GMAC mode (AES Cert. #1168, vendor-affirmed)
    - RSA (RSASSA-PKCS1-v1_5 and RSASSA-PSS) digital signatures (Cert. #560) and X9.31 RSA key-pair generation (Cert. #559)
    - ECDSA with the following NIST curves: P-256, P-384, P-521 (Cert. #141)

  - FIPS 186-2 x-Change Notice General Purpose RNG (Cert. #649)
  - FIPS 186-2 x-Change Notice Regular RNG (Cert. #649)
  - SP800-90 AES-256 counter mode DRBG (Cert. #23)
  - SP800-90 Dual-EC DRBG (Cert. #24)
  - KAS – SP800-56A (vendor-affirmed) EC Diffie-Hellman Key Agreement; key establishment methodology provides between 128 and 256-bits of encryption strength
- CNG.SYS supports the following non-Approved algorithms allowed for use in FIPS mode.
  - AES Key Wrap (AES Cert #1168; key establishment methodology provides between 128 and 256 bits of encryption strength)
  - Diffie-Hellman (DH) secret agreement
  - TLS and EAP-TLS
  - IKEv1 Key Derivation Functions
- CNG.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.
  - RSA encrypt/decrypt
  - RC2, RC4, MD2, MD4, MD5, HMAC MD5[1].
  - DES in ECB, CBC, and CFB with 8-bit feedback

The following diagram illustrates the master components of the module:

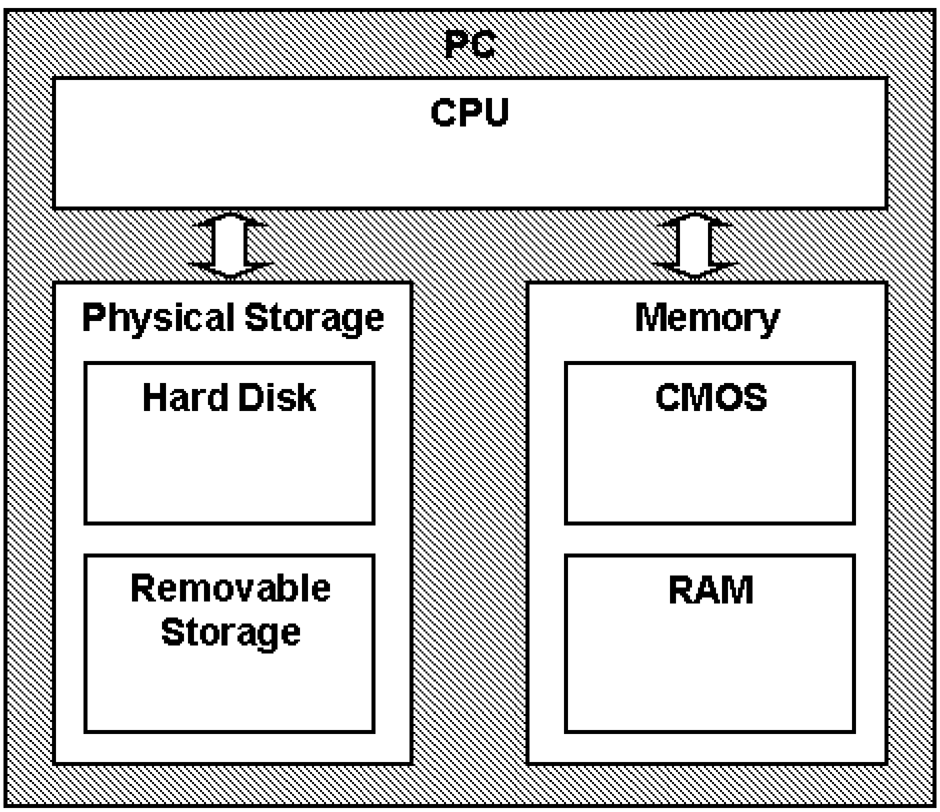

Figure 1 Master components of cng.sys crypto module

[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.

CNG.SYS (versions 6.1.7600.16385, 6.1.7600.16915, and 6.1.7600.21092) were tested using the following machine configurations:

x86	Windows 7 Ultimate – HP Compaq dc7600
x64	Windows 7 Ultimate – HP Compaq dc7600

CNG.SYS (versions 6.1.7600.17514, 6.1.7601.17725, 6.1.7601.17919, 6.1.7601.21861, and 6.1.7601.22076) were tested using the following machine configurations:

x86	Windows 7 Ultimate SP1 – HP Compaq dc7600
x64	Windows 7 Ultimate SP1 – HP Compaq dc7600

3 Cryptographic Module Ports and Interfaces

As shown in Figure 2, the CNG.SYS module is accessed through one of four logical interfaces. Kernel applications requiring cryptographic services use the BCrypt or legacy Fips APIs detailed in Section 5. Entropy sources supply random bits to the random number generator through the entropy APIs. Finally, both kernel mode and user mode random number generators use the SystemPrng interface to obtain seed material for their PRNGs.

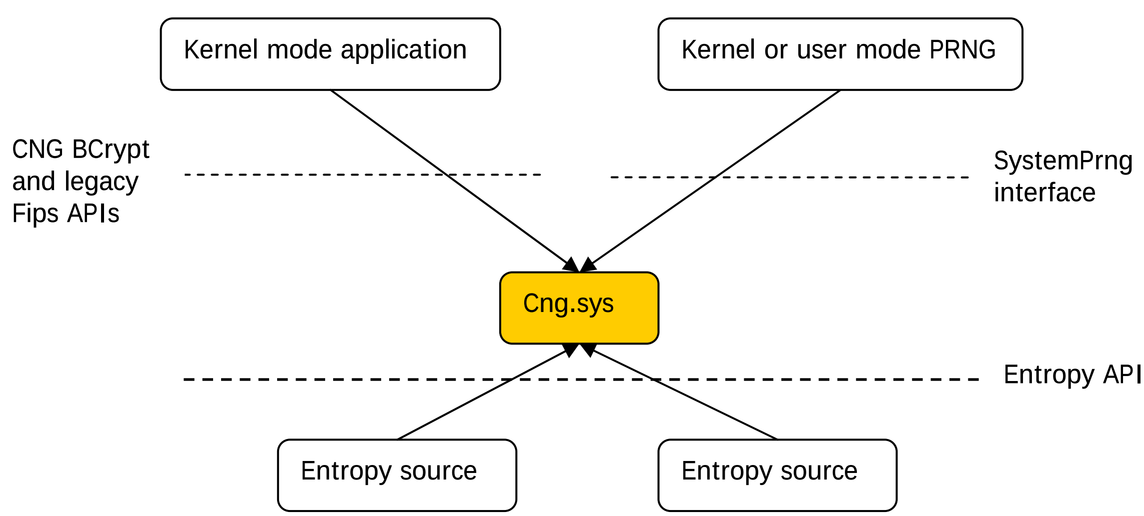

Figure 2 Relationship of cng.sys to other system components – cryptographic boundary shown in gold

3.1 Exported Functions

The following list contains the functions exported by CNG.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
- SystemPrng
- EntropyRegisterSource
- EntropyUnregisterSource
- EntropyProvideData

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

3.2 Data Input and Output Interfaces

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

3.3 Control Input Interface

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

3.4 Status Output Interface

The Status Output Interface for CNG.SYS also consists of the CNG.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 CNG.SYS.

4 Roles and Authentication

4.1 Roles

CNG.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 CNG.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 CNG.SYS executes (Microsoft Windows 7) does authenticate users.
Microsoft Windows 7 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 7 driver must have a standard initialization routine DriverEntry in order to be loaded. The Windows 7 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 7 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\CNG.

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 7 Kernel is responsible to call the DriverUnload routine of the associated DriverObject.

5.2 Algorithm Providers and Properties

5.2.1 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.

5.2.2 BCryptCloseAlgorithmProvider

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

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

5.2.3 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.

5.2.4 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.

5.2.5 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.

5.3 Random Number Generation

5.3.1 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 Dual-EC DRBG based random generator as defined in SP800-90.

During the function initialization, a seed is obtained from the output of the SystemPrng function. This provides the necessary entropy for the RNGs available through this function.

5.3.2 SystemPrng

```
BOOL SystemPrng(
        unsigned char   *pbRandomData,
        size_t          cbRandomData );
```

The SystemPrng() function fills a buffer with random bytes. It generates these bytes by taking the output of a cascade of two SP800-90 AES-256 counter mode based PRNGs, seeded from the Windows entropy pool. The Windows entropy pool is populated by periodically gathering random bits from the Trusted Platform Module (TPM) when present, as well as by periodically querying the values of the following OS variables:

- The process ID of the currently running process
- The thread ID of the currently running thread
- A 32-bit tick count since the system boot
- The current local date and time
- The current system time of day information consisting of the boot time, current time, time zone bias, time zone ID, boot time bias, and sleep time bias
- The current hardware-platform-dependent high-resolution performance-counter value
- The information about the system's current usage of both physical and virtual memory, and page file, Zero Page Count, Free Page Count, Modified Page Count, Modified No Write Page Count, Bad Page Count, Page Count By Priority, Repurposed Pages By Priority
- The system device information consisting of Number Of Disks, Number Of Floppies, Number Of CD Roms, Number Of Tapes, Number Of Serial Ports, Number Of Parallel Ports
- The local disk information including the numbers of sectors per cluster, bytes per sector, free clusters, and clusters that are available to the user associated with the calling thread
- A hash of the environment block for the current process
- Some hardware CPU-specific cycle counters
- The system file cache information consisting of Current Size, Peak Size, Page Fault Count, Minimum Working Set, Maximum Working Set, Current Size Including Transition In Pages, Peak Size Including Transition In Pages, Transition Repurpose Count, Flags
- The system processor power information consisting of Current Frequency, Thermal Limit Frequency, Constant Throttle Frequency, Degraded Throttle Frequency, Last Busy Frequency, Last C3 Frequency, Last Adjusted Busy Frequency, Processor Min Throttle, Processor Max Throttle, Number Of Frequencies, Promotion Count, Demotion Count, Error Count, Retry Count, Current Frequency Time, Current Processor Time, Current Processor Idle Time, Last Processor Time, Last Processor Idle Time

- The system page file information consisting of Next Entry Offset, Total Size, Total In-Use, Peak Usage, Page File Name
- The system processor idle information consisting of Idle Time
- The system processor performance information consisting of Idle Process Time, Io Read Transfer Count, Io Write Transfer Count, Io Other Transfer Count, Io Read Operation Count, Io Write Operation Count, Io Other Operation Count, Available Pages, Committed Pages, Commit Limit, Peak Commitment, Page Fault Count, Copy On Write Count, Transition Count, Cache Transition Count, Demand Zero Count, Page Read Count, Page Read Io Count, Cache Read Count, Cache Io Count, Dirty Pages Write Count, Dirty Write Io Count, Mapped Pages Write Count, Mapped Write Io Count, Paged Pool Pages, Non Paged Pool Pages, Paged Pool Allocated space, Paged Pool Free space, Non Paged Pool Allocated space, Non Paged Pool Free space, Free System page table entry, Resident System Code Page, Total System Driver Pages, Total System Code Pages, Non Paged Pool Look aside Hits, Paged Pool Lookaside Hits, Available Paged Pool Pages, Resident System Cache Page, Resident Paged Pool Page, Resident System Driver Page, Cache manager Fast Read with No Wait, Cache manager Fast Read with Wait, Cache manager Fast Read Resource Missed, Cache manager Fast Read Not Possible, Cache manager Fast Memory Descriptor List Read with No Wait, Cache manager Fast Memory Descriptor List Read with Wait, Cache manager Fast Memory Descriptor List Read Resource Missed, Cache manager Fast Memory Descriptor List Read Not Possible, Cache manager Map Data with No Wait, Cache manager Map Data with Wait, Cache manager Map Data with No Wait Miss, Cache manager Map Data Wait Miss, Cache manager Pin-Mapped Data Count, Cache manager Pin-Read with No Wait, Cache manager Pin Read with Wait, Cache manager Pin-Read with No Wait Miss, Cache manager Pin-Read Wait Miss, Cache manager Copy-Read with No Wait, Cache manager Copy-Read with Wait, Cache manager Copy-Read with No Wait Miss, Cache manager Copy-Read with Wait Miss, Cache manager Memory Descriptor List Read with No Wait, Cache manager Memory Descriptor List Read with Wait, Cache manager Memory Descriptor List Read with No Wait Miss, Cache manager Memory Descriptor List Read with Wait Miss, Cache manager Read Ahead IOs, Cache manager Lazy-Write IOs, Cache manager Lazy-Write Pages, Cache manager Data Flushes, Cache manager Data Pages, Context Switches, First Level Translation buffer Fills, Second Level Translation buffer Fills, and System Calls
- The system exception information consisting of Alignment Fix up Count, Exception Dispatch Count, Floating Emulation Count, and Byte Word Emulation Count
- The system look-aside information consisting of Current Depth, Maximum Depth, Total Allocates, Allocate Misses, Total Frees, Free Misses, Type, Tag, and Size
- The system processor performance information consisting of Idle Time, Kernel Time, User Time, Deferred Process Call Time, Interrupt Time Interrupt Count
- The system interrupt information consisting of context switches, deferred procedure call count, deferred procedure call rate, time increment, deferred procedure call bypass count, and asynchronous procedure call bypass count
- The system process information consisting of Next Entry Offset, Number Of Threads, Working Set Private Size, Create Time, User Time, Kernel Time, Image Name, Base Priority, Unique Process Id, Inherited From Unique Process Id, Handle Count, Session Id, Unique Process Key, Peak Virtual Size, Virtual Size, Page Fault Count, Peak Working Set Size, Working Set Size, Quota Peak Paged Pool Usage, Quota Paged Pool Usage, Quota Peak Non Paged Pool Usage, Quota Non Paged Pool Usage, Pagefile Usage, Peak Pagefile Usage, Private Page Count, Read Operation Count, Write Operation Count, Other Operation Count, Read Transfer Count, Write Transfer Count, Other Transfer Count

5.3.3 EntropyRegisterSource

```
NTSTATUS EntropyRegisterSource(
        ENTROPY_SOURCE_HANDLE * phEntropySource,
        ENTROPY_SOURCE_TYPE     entropySourceType,
```

```
    PCWSTR              entropySourceName );
```

This function is used to obtain a handle that can be used to contribute randomness to the Windows entropy pool. The handle is returned in the phEntropySource parameter. For this function, entropySource must be set to ENTROPY_SOURCE_TYPE_HIGH_PUSH, and entropySourceName must be a Unicode string describing the entropy source.

5.3.4 EntropyUnregisterSource

```
    NTSTATUS EntropyRegisterSource(
    ENTROPY_SOURCE_HANDLE hEntropySource);
```

This function is used to destroy a handle created with EntropyRegisterSource().

5.3.5 EntropyProvideData

```
    NTSTATUS EntropyProvideData(
        ENTROPY_SOURCE_HANDLE   hEntropySource,
        PCBYTE                  pbData,
        SIZE_T                  cbData,
        ULONG                   entropyEstimateInMilliBits );
```

This function is used to contribute entropy to the Windows entropy pool. hEntropySource must be a handle returned by an earlier call to EntropyRegisterSource. The caller provides cbData bytes in the buffer pointed to by pbData, as well as an estimate (in the entropyEstiamteInMilliBits parameter) of how many millibits of entropy are contained in these bytes.

5.4 Key and Key-Pair Generation

5.4.1 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.

5.4.2 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.

5.4.3 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.

5.4.4 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.

5.4.5 BCryptDestroyKey

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

The BCryptDestroyKey() function destroys a key.

5.5 Key Entry and Output

5.5.1 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_AES_WRAP_KEY_BLOB, 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.

5.5.2 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_ECCPRIVATE_BLOB, BCRYPT_ECCPUBLIC_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.

5.5.3 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_AES_WRAP_KEY_BLOB, 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.

5.6 Encryption and Decryption

5.6.1 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 authenticated encryption modes (i.e. AES-CCM and AES-GCM). It 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.

5.6.2 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 authenticated encryption modes (i.e. AES-CCM and AES-GCM). It 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.

5.7 Hashing and Message Authentication

5.7.1 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 and AES GMAC.

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 the HMAC and AES GMAC algorithms.

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.

5.7.2 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.

5.7.3 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.

5.7.4 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.

5.7.5 BCryptDestroyHash

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

The BCryptDestroyHash() function destroys a hash object.

5.8 Signing and Verification

5.8.1 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.

5.8.2 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.

5.9 Secret Agreement and Key Derivation

5.9.1 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 must be zero.

5.9.2 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) , BCRYPT_KDF_SP80056A_CONCAT (parameters in pParameterList: KDF_ALGORITHMID, KDF_PARTYUINFO, KDF_PARTYVINFO, KDF_SUPPPUBINFO, KDF_SUPPPRIVINFO).
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 KDF_USE_SECRET_AS_HMAC_KEY_FLAG. The KDF_USE_SECRET_AS_HMAC_KEY_FLAG value must only be used when pwszKDF is equal to BCRYPT_KDF_HMAC. It indicates that the secret will also be used as the HMAC key. If this flag is used, the KDF_HMAC_KEY parameter must not be specified in pParameterList.

5.9.3 BCryptDestroySecret

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

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

5.10 Legacy Compatibility Interfaces

The CNG.SYS driver provides an additional set of interfaces for compatibility with legacy software written for previous versions of Windows. These interfaces are described in this section.

These legacy interfaces are not exported by the CNG.SYS driver. A kernel mode user of the CNG.SYS driver must be able to reference these functions before using them. The user needs to acquire the table of pointers to the legacy functions from the CNG.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 CNG.SYS diver via the IoCallDriver function. Further information on irp and IoCallDriver can be found on Microsoft Windows 7 Driver Development Kit.

5.10.1 Key Formatting

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

5.10.1.1 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.

5.10.1.2 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.

5.10.2 Random Number Generation

5.10.2.1 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 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.

5.10.3 Data Encryption and Decryption

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

5.10.3.1 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.

5.10.3.2 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.

5.10.3.3 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.

5.10.3.4 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.

5.10.4 Hashing

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

5.10.4.1 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.

5.10.4.2 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.

5.10.4.3 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).

5.10.4.4 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.

5.10.4.5 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.

5.10.4.6 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.

5.10.4.7 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.

5.10.4.8 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.

5.10.4.9 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.

5.11Configuration

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**
BCryptEnumAlgorithms	Enumerates the algorithms for a given set of operations.
BCryptEnumProviders	Returns a list of providers for a given algorithm.
BCryptRegisterConfigChangeNotify	This API differs slightly between User-Mode and Kernel-Mode.
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.
BCryptUnregisterConfigChangeNotify	This API differs slightly between User-Mode and Kernel-Mode.
BCryptGetFipsAlgorithmMode	Used by applications to determine whether CNG.SYS is operating in FIPS mode. Some applications use the value returned by this API to alter their own behavior, such as blocking the use of some SSL versions.

5.12Other Interfaces

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

Function Name	**Description**
BCryptDeriveKeyCapi BCryptDeriveKeyPBKDF2 SslDecryptPacket SslEncryptPacket SslExportKey SslFreeObject SslImportKey SslLookupCipherLengths SslLookupCipherSuiteInfo SslOpenProvider SslIncrementProviderReferenceCount SslDecrementProviderReferenceCount AppHashComputeFileAttributes	

6 Operational Environment

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

7 Cryptographic Key Management

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

7.1 Cryptographic Keys, CSPs, and SRDIs

The CNG.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 TDEA 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.
RNG & DRBG Seeds and Seed Keys	Secret values maintained internal to the module that provide the necessary seeding/entropy material to the approved RNG and DRBGs.

7.2 Access Control Policy

The CNG.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.

CNG.SYS crypto module SRDI/Service Access Policy	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	RNG & DRBG Seeds/Seed Keys
Cryptographic Module Power Up and Power Down									
Key Formatting	w								
Random Number Generation									x
Data Encryption and Decryption	x								
Hashing		wx							
Acquiring a Table of Pointers to FipsXXX Functions									
Algorithm Providers and Properties									
Key and Key-Pair Generation	wd	wd	wd	wd	wd	wd	wd	wd	x
Key Entry and Output	rw	rw	rw	rw	rw	rw	rw	rw	
Signing and Verification			x	x	x	x			x
Secret Agreement and Key Derivation							x	x	x

7.3 Key Material

When CNG.SYS is loaded in the Windows 7 Operating System kernel, no keys exist within it. A kernel module is responsible for importing keys into CNG.SYS or using CNG.SYS's functions to generate keys.

7.4 Key Generation

CNG.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

CNG.SYS can use FIPS approved Diffie-Hellman key agreement (DH), Elliptic Curve Diffie-Hellman key agreement (ECDH), RSA key transport and manual methods to establish keys.
CNG.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_SP80056A_CONCAT. This KDF supports the Concatenation KDF as specified in SP 800-56A (Section 5.8.1).
- BCRYPT_KDF_HASH. This KDF supports FIPS approved SP800-56A (Section 5.8), X9.63, and X9.42 key derivation.
- BCRYPT_KDF_HMAC. This KDF supports the IPsec IKEv1 key derivation that is allowed in FIPS mode when used to establish keys for IKEv1 as specified in FIPS 140-2 Implementation Guidance 7.1.
- BCRYPT_KDF_TLS_PRF. This KDF supports the SSLv3.1 and TLSv1.0 key derivation that is allowed in FIPS mode when used to establish keys for SSLv3.1 or TLSv1.0 as specified in FIPS 140-2 Implementation Guidance 7.1.

7.6 Key Entry and Output

Keys can be both exported and imported out of and into CNG.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 CNG.SYS via FipsDesKey(). DESTable struct can be exported out of CNG.SYS via FipsDesKey(). DESTable struct can be imported into CNG.SYS via FipsDes() or FipsCBC().
Triple DES keys can be imported into CNG.SYS via Fips3Des3Key(). DES3Table struct can be exported out of CNG.SYS via Fips3Des3Key(). DES3Table struct can be imported into CNG.SYS via Fips3Des() or FipsCBC().
HMAC keys can be imported into CNG.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

CNG.SYS does not provide persistent storage of keys.

7.8 Key Archival

CNG.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 CNG.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 CNG.SYS.

7.9 Key Zeroization

All keys are destroyed and their memory location zeroized when the operator 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 CNG.SYS are destroyed and their memory location zeroized after they have been used in FipsDes, Fips3Des, or FipsCBC.

8 Self-Tests

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

- HMAC-SHA-1 Known Answer Test
- SHA-256 and SHA-512 Known Answer Tests

- Triple-DES encrypt/decrypt ECB Known Answer Test
- AES-128 encrypt/decrypt EBC Known Answer Test
- AES-128 encrypt/decrypt CBC Known Answer Test
- AES-128 encrypt/decrypt CCM Known Answer Test
- AES-128 encrypt/decrypt GCM Known Answer Test
- RSA sign/verify test with 2048-bit key
- ECDSA sign/verify test on P256 curve
- ECDH secret agreement Known Answer Test on P256 curve
- SP800-56A concatenation KDF Known Answer Tests
- FIPS 186-2 x-Change Notice Regular Known Answer Test
- FIPS 186-2 x-Change Notice General Purpose Known Answer Test
- SP800-90 AES-256 counter mode DRBG Known Answer Tests (instantiate, generate and reseed)
- SP800-90 Dual-EC DRBG Known Answer Tests (instantiate, generate and reseed)

In all cases for any failure of a power-on (start up) self-test, CNG.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.

CNG.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. CNG.SYS also performs a continuous RNG test on each implemented RNG as prescribed in FIPS 140-2.

9 Design Assurance

The CNG.SYS crypto module is part of the overall Windows 7 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 CNG.SYS is installed and started as part of the Windows 7 operating system.

10 Additional Details

For the latest information on Windows 7, please see 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
Vijay	2/28/2008	1.6	SP1/WS08 edits

Bharadwaj			