Microsoft Corporation Windows Embedded Compact Cryptographic Primitives Library (bcrypt.dll) Non-Proprietary Security Policy Document

Microsoft Corporation Windows Embedded Compact 2013 Operating System Cryptographic Primitives Library (BCRYPT.DLL) FIPS 140-2 Security Policy Document

This document specifies the security policy for the Microsoft Corporation Windows Embedded Compact Cryptographic Primitives Library (BCRYPT.DLL) as described in FIPS PUB 140-2.

November 08, 2019

Document Version: 1.9

The information contained in this document represents the current view of Microsoft Corporation on the issues discussed as of the date of publication. Because Microsoft must respond to changing market conditions, it should not be interpreted to be a commitment on the part of Microsoft, and Microsoft cannot guarantee the accuracy of any information presented after the date of publication.

This document is for informational purposes only. MICROSOFT MAKES NO WARRANTIES, EXPRESS OR IMPLIED, AS TO THE INFORMATION IN THIS DOCUMENT.

Contents

1 Cryptographic Module Specification

The Microsoft Corporation Windows Cryptographic Primitives Library is a general purpose, software-based, cryptographic module. The primitive provider functionality is offered through one cryptographic module, BCRYPT.DLL (version 8.00.6273 – Windows Embedded Compact 2013), subject to FIPS-140-2 validation. BCRYPT.DLL provides cryptographic services, through its documented interfaces, to Windows Embedded Compact 2013 components and applications running on Windows Embedded Compact 2013. This cryptographic module is referred to as BCRYPT.DLL or BCRYPT in this document. The cryptographic module, BCRYPT.DLL, encapsulates several different cryptographic algorithms in an easy to-use cryptographic module accessible via the Microsoft CNG (Cryptography, Next Generation) API. It can be dynamically linked into applications by software developers to permit the use of general- purpose FIPS 140-2 Level 1 compliant cryptography as provided in the table.

Anytime the term 'Cryptographic Primitive Library' is used through out this document, regardless of the preceding qualifiers , it refers to the module.

Section	Section Title	Level
1	Cryptographic Module specification	1
2	Cryptographic Module Ports and Interfaces	1
3	Roles, Services, and Authentication	1
4	Finite State Model	1
5	Physical Security	NA
6	Operational Environment	1
7	Cryptographic Key Management	1
8	EMI/EMC	1
9	Self-Tests	1
10	Design Assurance	1
11	Mitigation of Other Attacks	NA

1.1 Cryptographic Boundary

The Windows Embedded Compact 2013 BCRYPT.DLL consists of a dynamically-linked library (DLL). The cryptographic boundary for BCRYPT.DLL is defined as the enclosure of the computer system, on which BCRYPT.DLL is to be executed. The logical boundary for BCRYPT.DLL is the BCRYPT.DLL file itself. The physical configuration of BCRYPT.DLL, as defined in FIPS-140-2, is multi-chip standalone.

2 Security Policy

BCRYPT.DLL operates under several rules that encapsulate its security policy.

- BCRYPT.DLL is supported on Windows Embedded Compact 2013
- Windows Embedded Compact 2013 is an operating system supporting a “single user” mode where there is only one interactive user during a logon session.
- BCRYPT.DLL is only in its Approved mode of operation when Windows Embedded Compact 2013 is booted normally, meaning Debug mode is disabled.
- Also for BCRYPT.DLL to be in approved mode of operation the scavenge interval for gathering random data from different sources should be set to 1 sec. This ensures even in the worst case after boot-up the device is guaranteed to have 256 bits of entropy.
  The scavenge interval can be set by modifying the registry at:
  [HKEY_LOCAL_MACHINE\Comm\Security\Crypto]
  ScavengeIntervalInSeconds=dword:1
- Every time a Scavenge is performed it's expected to provide a minimum of 50 bits of entropy to the DRBG.
- All users assume either the User or Cryptographic Officer roles.
- BCRYPT.DLL provides no authentication of users. Roles are assumed implicitly. The authentication provided by the Windows Embedded Compact 2013 operating system is not in the scope of the validation.
- All cryptographic services implemented within BCRYPT.DLL are available to the User and Cryptographic Officer roles.
- BCRYPT.DLL implements the following FIPS-140-2 Approved algorithms:

  Note - Not all CAVP-tested modes and options are implemented by BCRYPT.dll. Only the modes and options listed below are implemented.

  - FIPS 180-4 SHA-1, SHA-256, SHA-384, SHA-512 hash (Cert #3648)
  - FIPS 198-1 SHA-1, SHA-256, SHA-384, SHA-512 HMAC (Cert #2942).

    Note - HMAC Keys used in HMAC-SHA1 must be 112 bits in length (or longer), and that any key length shorter than that is not allowed as per SP800-131A
  - SP 800-67r1 Triple-DES (2 key legacy-use decryption and 3 key encryption/decryption) in ECB and CBC modes (Certs #2381).

    Note - Triple-DES 2 key is restricted to use for decryption only (legacy use) as per SP 800-131A.

    - As per FIPS 140-2 IG A.13 there is a limit of 2^16 data block encryptions with the same Triple-DES key. The operator of the module is responsible for enforcing this limit
  - FIPS 197 AES-128, AES-192, AES-256 in ECB, CBC, CCM, CFB8 modes and AES-128 GCM modes (Certs #4430)

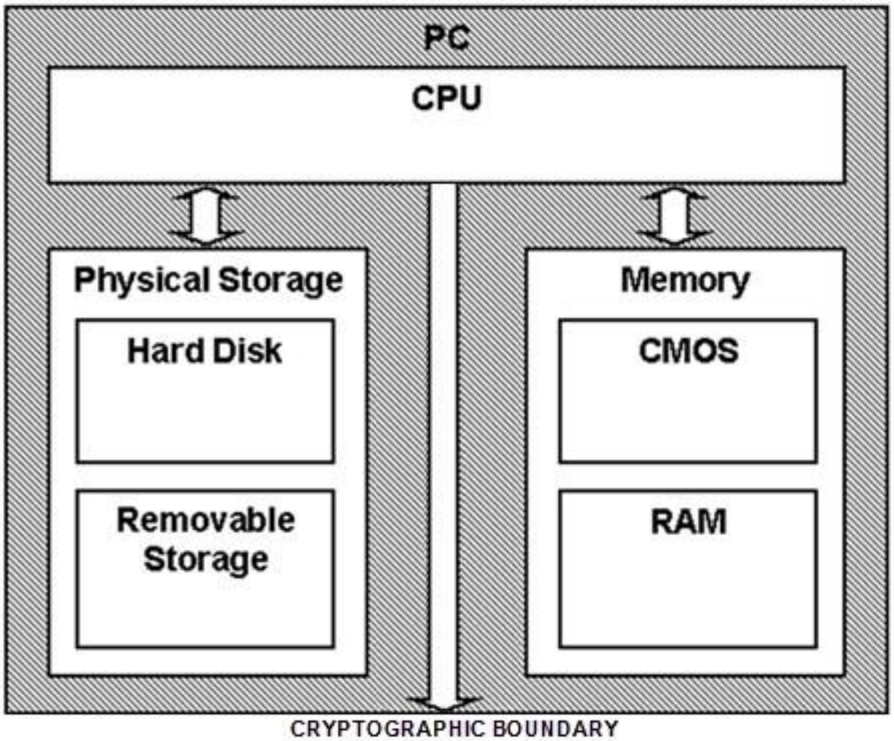
    Note - Only GCM decryption is considered Approved as per IG A.5
  - FIPS 186-4 RSA (RSASSA-PKCS1-v1_5) digital signatures (Cert #2411)
    - RSA key sizes up to 4096 bits are supported. A 1024-bit or 1536-bit modulus and/or SHA-1 are only allowed for legacy digital signature verification as per SP800-131A.
  - FIPS 186-4 ECDSA with the following NIST curves: P-256, P-384, P-521 (Cert #1072). ECDSA with SHA-1 is only allowed for legacy digital signature verification as per SP800-131A.
  - FIPS 186-2 DSA PQG Verification and Signature Verification for legacy use only (Cert #1187, L=1024, N=160)
  - SP 800-90A Deterministic Random Bit Generator (DRBG) with AES-256-CTR (Cert #1429)

- SP 800-56A Diffie-Hellman Key Agreement; Finite Field Cryptography (FFC) with parameter FB (p=2048, q=224) and FC (p=2048, q=256); key establishment methodology provides 112 bits of encryption strength (KAS Cert #114)
- SP 800-56A EC Diffie-Hellman Key Agreement; Elliptic Curve Cryptography (ECC) with parameter EC (P-256 w/ SHA-256), ED (P-384 w/ SHA-384), and EE (P-521 w/ SHA-512); key establishment methodology provides between 128 and 256 bits of encryption strength (KAS Cert #114)
- SP800-135 IKEv1 and TLS(1.0/1.1 & 1.2) KDF primitives (CVL Cert#C705)
- SP800-133 Cryptographic Key Generation(vendor affirmed)

- BCRYPT.DLL supports the following non-Approved (but allowed) algorithms:
  - MD5 is allowed in the approved mode of operation when used as part of an approved key transport scheme where no security is provided by the algorithm (as per FIPS 140-2 IG G.13)
  - NDRNG is allowed for usage in FIPS mode in order to seed the Approved DRBG
  - RSA Key wrapping (key wrapping; key establishment methodology provides between 112 and 150 bits of encryption strength. Keys can be entered by using the recipient's public key, per Section 7.6).

- BCRYPT.DLL also supports the following non-Approved algorithms that may not be used at all when operating the module in a FIPS compliant manner:
  - AES-128 GCM mode for encryption
  - FIPS 186-2 DSA Key Generation and Signature Generation (L=1024, N=160)
  - RC2, RC4, MD2, MD4
  - DES in ECB, CBC, and CFB with 8-bit feedback
  - Dual-EC DRBG non-Approved implementation
  - FIPS 186-2 DSA RNG non-Approved implementation.

The following diagram illustrates the master components of the BCRYPT.DLL module



BCRYPT.DLL was tested using the following machine configurations:

Windows Embedded Compact 2013:

ARMV7	Microsoft Corporation Windows Embedded Compact 2013 – TI OMAP EVM3730 DM3730 ARM Cortex A8 Processor
x86	Microsoft Corporation Windows Embedded Compact 2013 – i586 (MSTIPDX-600)

3 Cryptographic Module Ports and Interfaces

3.1 Ports and Interfaces

3.1.1 Export Functions

The following list contains the functions exported by BCRYPT.DLL to its callers.

- BCryptCloseAlgorithmProvider
- BCryptCreateHash
- BCryptDecrypt
- BCryptDeriveKey
- BCryptDestroyHash
- BCryptDestroyKey
- BCryptDestroySecret
- BCryptDuplicateHash
- BCryptDuplicateKey
- BCryptEncrypt
- BCryptEnumAlgorithms
- BCryptEnumProviders
- BCryptExportKey
- BCryptFinalizeKeyPair
- BCryptFinishHash

- BCryptFreeBuffer
- BCryptGenerateKeyPair
- BCryptGenerateSymmetricKey
- BCryptGenRandom
- BCryptGetProperty
- BCryptHashData
- BCryptImportKey
- BCryptImportKeyPair
- BCryptOpenAlgorithmProvider
- BCryptSecretAgreement
- BCryptSetProperty
- BCryptSignHash
- BCryptVerifySignature
- BCryptQueryProviderRegistration
- BCryptEnumRegisteredProviders
- BCryptCreateContext
- BCryptDeleteContext
- BCryptEnumContexts
- BCryptConfigureContext
- BCryptQueryContextConfiguration
- BCryptAddContextFunction
- BCryptRemoveContextFunction
- BCryptEnumContextFunctions
- BCryptConfigureContextFunction
- BCryptQueryContextFunctionConfiguration
- BCryptEnumContextFunctionProviders
- BCryptSetContextFunctionProperty
- BCryptQueryContextFunctionProperty
- BCryptRegisterConfigChangeNotify
- BCryptUnregisterConfigChangeNotify
- BCryptResolveProviders
- BCryptGetFipsAlgorithmMode

All these functions are available in the approved and non-approved modes. Furthermore, these are the only functions that this module can perform.

Additionally, BCRYPT.DLL exports crypto configuration functions. They are described in a separate section 5.9 below for informational purposes.

3.1.2 Data Input and Output Interfaces

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

3.1.3 Control Input Interface

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

3.1.4 Status Output Interface

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

3.2 Cryptographic Bypass

Cryptographic bypass is not supported by BCRYPT.DLL.

4 Roles and Authentication

4.1 Roles

BCRYPT.DLL provides User and Cryptographic Officer roles (as defined in FIPS 140-2). These roles share all the services implemented in the cryptographic module.

When an application requests the crypto module to generate keys for a user, the keys are generated, used, and deleted as requested by applications. There are no implicit keys associated with a user. Each user may have numerous keys, and each user’s keys are separate from other users’ keys.

4.2 Maintenance Roles

Maintenance roles are not supported by BCRYPT.DLL.

4.3 Operator Authentication

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

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 Algorithm Providers and Properties

5.1.1 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. See http://msdn.microsoft.com for CNG providers. Unless the calling function specifies the name of the provider, the default provider is used. The default provider is the first provider listed for a given algorithm. The calling function must pass the BCRYPT_ALG_HANDLE_HMAC_FLAG flag in order to use an HMAC function with a hash algorithm.

5.1.2 BCryptCloseAlgorithmProvider

NTSTATUS WINAPI BCryptCloseAlgorithmProvider(BCRYPT_ALG_HANDLE hAlgorithm, ULONG dwFlags);

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

5.1.3 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, e.g., a cryptographic key. The CNG object is referenced by a handle, the property name is a NULL terminated string, and the value of the property is a length-specified byte string.

User can pass BCRYPT_INTERNAL_AESCTR_RNG_SELF_TEST to pass pbInput (as pbEntropy) to AesCtrRng_Instantiate. However, BCryptSetProperty does not support pbPersonalizationString.

5.1.4 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, e.g., a cryptographic key. The CNG object is referenced by a handle, the property name is a NULL terminated string, and the value of the property is a length-specified byte string.

5.1.5 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.

5.2 Random Number Generation

5.2.1 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 algorithms:

- BCRYPT_RNG_ALGORITHM. The DRBG based on the AES counter mode specified in the NIST SP 800-90A standard.
- BCRYPT_RNG_FIPS186_DSA_ALGORITHM. This is Non-Approved RNG algorithm suitable for DSA (Digital Signature Algorithm) as defined in FIPS 186-2 which is not allowed in FIPS mode.
- BCRYPT_RNG_DUAL_EC_ALGORITHM. This is the dual elliptic curve Non-Approved RNG algorithm specified in the NIST SP 800-90A standard, currently which is not allowed in FIPS mode.

When BCRYPT_RNG_USE_ENTROPY_IN_BUFFER is specified in the dwFlags parameter, this function will use the number in the pbBuffer buffer as additional entropy for the random number. If this flag is not specified, this function will use a random number for the entropy.

5.3 Key and Key-Pair Generation

The following list of Services for Key and Key-Pair Generation all use the unmodified output from the module's SP800-90A AES-CTR DRBG to produce symmetric keys and generated seeds when the module is being operated in the Approved mode.

5.3.1 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 cbSecret bytes long key value provided in the pbSecret memory location. The calling application must specify a handle to the algorithm provider opened with the BCryptOpenAlgorithmProvider() function. The algorithm specified when the provider was opened must support symmetric key encryption.

5.3.2 BCryptGenerateKeyPair

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

The BCryptGenerateKeyPair() function creates a public/private key pair object without any cryptographic keys in it. After creating such an empty key pair object using this function, call the BCryptSetProperty() function to set its properties. The key pair can be used only after BCryptFinalizeKeyPair() function is called.

5.3.3 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 pair.

5.3.4 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 object.

5.3.5 BCryptDestroyKey

NTSTATUS WINAPI BCryptDestroyKey(BCRYPT_KEY_HANDLE hKey);

The BCryptDestroyKey() function destroys a key.

5.4 Key Entry and Output

5.4.1 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.

5.4.2 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_DSA_PRIVATE_BLOB, BCRYPT_DSA_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_DSA_PRIVATE_BLOB, LEGACY_DSA_PUBLIC_BLOB, LEGACY_DSA_V2_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.

5.4.3 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. hKey [in] is

the handle of the key to export.

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_DSA_PRIVATE_BLOB, BCRYPT_DSA_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_RSAPRIVATE_BLOB, BCRYPT_RSAPUBLIC_BLOB, LEGACY_DH_PRIVATE_BLOB, LEGACY_DH_PUBLIC_BLOB, LEGACY_DSA_PRIVATE_BLOB, LEGACY_DSA_PUBLIC_BLOB, LEGACY_DSA_V2_PRIVATE_BLOB, LEGACY_RSAPRIVATE_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.

5.5 Encryption and Decryption

5.5.1 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.

5.5.2 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.

5.6 Hashing and HMAC

5.6.1 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.

5.6.2 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.

5.6.3 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.

5.6.4 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.

5.6.5 BCryptDestroyHash

NTSTATUS WINAPI BCryptDestroyHash(BCRYPT_HASH_HANDLE hHash);

The BCryptDestroyHash() function destroys a hash object.

5.7 Signing and Verification

5.7.1 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.

Note: According to SP 800-131A, SHA-1 hash signing should no longer be used, and is disallowed as of 12/2013. This is for legacy use only for signature verification.

5.7.2 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.

Note: According to SP 800-131A, SHA-1 hash signing should no longer be used, and is disallowed as of 12/2013. This is for legacy use only for signature verification.

5.8 Secret Agreement and Key Derivation

5.8.1 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.

5.8.2 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.

5.8.3 BCryptDestroySecret

NTSTATUS WINAPI BCryptDestroySecret(BCRYPT_SECRET_HANDLE hSecret);

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

5.9 Configuration

The below list of approved functions are used to configure cryptographic providers on the system. Please see http://msdn.microsoft.com for details.

Function Name	Description
BCryptAddContextFunction	Adds a function (algorithm or cipher-suite) to a context function list.
BCryptConfigureContext	Configures a context.
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.
BCryptRemoveContextFunction	Removes a function (algorithm or cipher-suite) from a context function 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	Unregisters Config Change notification request.
BCryptGetFipsAlgorithmMode	Retrieve whether the FIPS algorithm mode is enabled or not.

6 Operational Environment

BCRYPT.DLL is intended to run on Windows Embedded Compact 2013 operating systems and in Single User mode, on the hardware as defined in Section 2, and is tested on the below operational environments. When run in these configurations, multiple concurrent operators are not supported.

Windows Embedded Compact 2013:

- Windows Embedded Compact 2013 running on a TI OMAP TMDSEVM3730 with Texas Instruments EVM3730 CPU
- Windows Embedded Compact 2013 running on an eBox-330-A with MSTI PDX-600 CPU

BCRYPT.DLL is also compliant on platforms that are not listed above. Please see FIPS PUB 140-2 Implementation Guidance G.5 for more details on portability rules and requirements.

Because BCRYPT.DLL module is a DLL, each process requesting access is provided its own instance of the module. As such, each process has full access to all information and keys within the module. Note that no keys or other information are maintained upon detachment from the DLL, thus an instantiation of the module will only contain keys or information that the process has placed in the module.

7 Cryptographic Key Management

BCRYPT.DLL crypto module manages keys in the following manner.

7.1 Cryptographic Keys, CSPs, and SRDIs

The BCRYPT.DLL crypto module contains the following security relevant data items:

Security Relevant Data	**Item SRDI Description**
Symmetric encryption/decryption keys	Keys used for AES or Triple-DES encryption/decryption. Key sizes for AES are 128, 192, and 256 bits and key sizes for Triple-DES are 168 and 112 bits.
HMAC keys	Keys used for HMAC-SHA1, HMAC-SHA256, HMAC- SHA384, and HMAC-SHA512
Hard coded CSP cert	Ce_csp_root - Cert used for self-check. This is a 2048-bit RSA Public Key.
DSA Public Keys	Keys used for the legacy verification of DSA digital signatures. These are 1024-bit DSA Public Keys.
ECDSA Public Keys	Keys used for the verification of ECDSA digital signatures. Curve sizes are P-256, P-384, and P-521.
ECDSA Private Keys	Keys used for the calculation of ECDSA digital signatures. Curve sizes are P-256, P-384, and P-521.
RSA Public Keys	Keys used for the verification of RSA digital signatures. Key sizes are between 1024 and 4096 bits.
RSA Private Keys	Keys used for the calculation of RSA digital signatures. Key sizes are between 2048 and 4096 bits.
DH Public and Private values	Public and private values used for Diffie-Hellman key

	establishment. Key sizes are 2048 bits.
ECDH Public and Private values	Public and private values used for EC Diffie-Hellman key establishment. Curve sizes are P-256, P-384, and P-521
DH/ECDH Shared Secret	Shared Secret values resulting from Diffie-Hellman and EC Diffie-Hellman key establishment
DRBG Parameters	DRBG Seed (384 bits), Entropy (512 bits), Key (256 bits) and State value ‘V’ (128 bits)

7.2 Access Control Policy

The BCRYPT.DLL 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.

BCRYPT.DLL crypto module SRDI/Service Access Policy	Security RelevanData Item	Symmetric 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	DH/ECDH Shared Secret	DRBG Parameters	DSA Public Keys	Hard coded CSP Cert
Service Categories													
Cryptographic Module Power Up and Power Down						r/x							
Key Formatting		w											
Random Number Generation (DRBG)		x									r,w,x		
Data Encryption and Decryption		x											
Hashing			x/w										
Acquiring a Table of Pointers to BCryptXXX 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					x	x
Secret Agreement and Key Derivation	▨							x	x	x			

7.3 Key Material

Each time an application links with BCRYPT.DLL, the DLL is instantiated and no keys exist within it. The user application is responsible for importing keys into BCRYPT.DLL or using BCRYPT.DLL's functions to generate keys.

7.4 Key Generation

BCRYPT.DLL can create and use keys for the following algorithms: RSA, DH, ECDH, ECDSA, RC2, RC4, DES, Triple-DES, AES, and HMAC (RC2, RC4 and DES may not be used in FIPS mode).

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 and AES key-pairs are generated using SP800-90A DRBG. ECDSA, DSA, DH, and ECDH keys and key-pairs are generated following the techniques given in FIPS PUB 186-2, Appendix 3, Random Number Generation. RSA key-pairs are generated per ANSIX9.31.

<u>Note</u>: Restrictions on key Generation

- ECDSA Key Generation as per 186-2 cannot be tested for 3SUB or 5SUB submissions and as such will not allow for usage in FIPS mode
- RSA Key Generation as per 186-2 cannot be tested for 3SUB or 5SUB submissions and as such will not be allowed for usage in FIPS mode
- Keys generated while not operating in the Approved mode of operation (as described in section 2) cannot be used in the Approved mode, and vice versa.

7.5 Key Establishment

BCRYPT.DLL can use FIPS approved Diffie-Hellman key agreement (DH), Elliptic Curve Diffie-Hellman key agreement (ECDH), and manual methods to establish keys.

BCRYPT.DLL 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_HASH. This KDF supports FIPS SP800-56A (Section 5.8), X9.63, and X9.42 key derivation.
- BCRYPT_KDF_HMAC. This KDF supports FIPS IPsec IKE v1 key derivation as specified in FIPS 140-2 Implementation Guidance.
- BCRYPT_KDF_TLS_PRF. This KDF supports FIPS SSLv3.1 and TLS v1.0/v1.1/v1.2 key derivation as specified in FIPS 140-2 Implementation Guidance.

7.6 Key Entry and Output

Keys in plaintext can be both exported and imported out of and into BCRYPT.DLL 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.

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

BCRYPT.DLL does not provide persistent storage of keys.

7.8 Key Archival

BCRYPT.DLL does not directly archive cryptographic keys. The Authenticated 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.

7.9 Key Zeroization

All keys are destroyed and their memory location zeroized when the User calls BCryptDestroyKey() or BCryptDestroySecret() on that key handle.

7.10 Mapping of Services, Algorithms, and Critical Security Parameters

The following table maps the services to their corresponding algorithms and critical security parameters (CSPs).

Service	Algorithms	CSPs
Power Up and Power Down	None	None
Algorithm Providers and Properties	None	None
Random Number Generation	AES-256 CTR DRBG NDRNG (allowed, used to provide entropy to DRBG)	AES-CTR DRBG Seed AES-CTR DRBG Entropy Input AES-CTR DRBG V AES-CTR DRBG Key
Key and Key-Pair Generation	RSA, DH, ECDH, ECDSA, RC2, RC4, DES, Triple-DES, AES, and HMAC (RC2, RC4, and DES cannot be used in FIPS mode.)	Symmetric Keys Asymmetric Public Keys Asymmetric Private Keys
Key Entry and Output	None	Symmetric Keys Asymmetric Public Keys
Encryption and Decryption	Triple-DES with 2 key (encryption disallowed) and 3 key in ECB and CBC modes; AES-128, AES-192, and AES-256 in ECB, CBC, CCM, CFB8 and CTR modes; AES-128 GCM mode; (AES-128 GCM mode encryption is not allowed in FIPS mode) (RC2, RC4, RSA, and DES, which cannot be used in FIPS mode)	Symmetric Keys Asymmetric Public Keys Asymmetric Private Keys

Hashing and Message Authentication	FIPS 180-4 SHA-1, SHA-256, SHA-384, and SHA-512;
	FIPS 180-4 SHA-1, SHA-256, SHA-384, SHA-512 HMAC;
	MD5 (allowed in TLS and EAP-TLS);
	MD2 and MD4 (disallowed in FIPS mode)
		Symmetric Keys (for HMAC)

Signing and Verification	FIPS 186-4 RSA (RSASSA-PKCS1v1_5) digital signature generation (with 1024 – 4096 modulus) and verification (with 2048 - 4096 modulus); supports SHA-1 FIPS 186-4 ECDSA with the following NIST curves: P-256, P384, P-521 for signature verification	RSA Public Keys RSA Private Keys ECDSA Public keys ECDSA Private keys DSA Public Keys
Secret Agreement and Key Derivation	KAS – SP 800-56A Diffie-Hellman Key Agreement; Finite Field Cryptography (FFC) KAS – SP 800-56A EC Diffie-Hellman Key Agreement SP 800-135 IKEv1 and TLS(1.0/1.1 & 1.2) KDF primitives	DH Private and Public Values ECDH Private and Public Values DH/ECDH Shared Secret
Show Status	None	None
Self-Tests	See Section 8 Self-Tests for the list of algorithms	Hard coded CSP cert
Zeroization	None	All Keys / CSPs can be zeroized

7.11 Mapping of Services, Export Functions, and Invocations

The following table maps the services to their corresponding export functions and invocations.

Service	Export Functions	Invocations
Power Up and Power Down	Driver Entry Driver Unload	This service is fully automatic. The User / Cryptographic Officer does not take any actions to explicitly start this service. This service is executed upon startup of this module.
Algorithm Providers and Properties	BCryptOpenAlgorithmProvider BCryptCloseAlgorithmProvider BCryptSetProperty BCryptGetProperty BCryptFreeBuffer	The User / Cryptographic Officer does not take any actions to explicitly start this service. This service is executed whenever one of these exported functions is called.
Random Number Generation	BcryptGenRandom	The User / Cryptographic Officer does not take any actions to explicitly start this service. This service is executed whenever one of these exported functions is called.
Key and Key-Pair Generation	BCryptGenerateSymmetricKey BCryptGenerateKeyPair BCryptFinalizeKeyPair BCryptDuplicateKey BCryptDestroyKey	The User / Cryptographic Officer does not take any actions to explicitly start this service. This service is executed whenever one of these exported functions is called.
Key Entry and Output	BCryptImportKey BCryptImportKeyPair BCryptExportKey	The User / Cryptographic Officer does not take any actions to explicitly start this service. This service is executed whenever one of these exported functions is called.

Encryption and Decryption	BCryptEncrypt BCryptDecrypt	The User / Cryptographic Officer does not take any actions to explicitly start this service. This service is executed whenever one of these exported functions is called.
Hashing and Message Authentication	BCryptCreateHash BCryptHashData BCryptDuplicateHash BCryptFinishHash BCryptDestroyHash	The User / Cryptographic Officer does not take any actions to explicitly start this service. This service is executed whenever one of these exported functions is called.
Signing and Verification	BCryptSignHash BCryptVerifySignature	The User / Cryptographic Officer does not take any actions to explicitly start this service. This service is executed whenever one of these exported functions is called.
Secret Agreement and Key Derivation	BCryptSecretAgreement BCryptDeriveKey BCryptDestroySecret	The User / Cryptographic Officer does not take any actions to explicitly start this service. This service is executed whenever one of these exported functions is called.
Show Status	All Exported Functions	This service is fully automatic. The User / Cryptographic Officer does not take any actions to explicitly start this service. This service is executed upon completion of an exported function.
Self-Tests	Driver Entry	This service is fully automatic. The User / Cryptographic Officer does not take any actions to explicitly start this service. This service is executed upon startup of this module.
Zeroization	BCryptDestroyKey BCryptDestroySecret	The User / Cryptographic Officer does not take any actions to explicitly start this service. This service is executed whenever one of these exported functions is called.

8 Self-Tests

BCRYPT.DLL performs the following power-on (startup) self-tests when DllMain is called by the operating system.

- SHA-1, SHA-256 & SHA-512 hash Known Answer Test
- HMAC-SHA-1 Known Answer Test
- Triple-DES encrypt/decrypt ECB Known Answer Test with 112 and 168 bit key sizes.
- Triple-DES encrypt/decrypt CBC Known Answer Test with 112 and 168 bit key sizes.
- AES-128, AES-192, AES-256 encrypt/decrypt ECB Known Answer Test
- AES-128, AES-192, AES-256 encrypt/decrypt CBC Known Answer Test
- AES-128, AES-192, AES-256 encrypt/decrypt CFB Known Answer Test
- AES-128, AES-192, AES-256 encrypt/decrypt CCM Known Answer Test
- AES-128 encrypt/decrypt GCM Known Answer Test

- RSA sign and verify Known Answer Test with 2048 bit key size.
- DSA sign/verify test with 1024 bit key size
- DH secret agreement Known Answer Test with 2048 bit key size.
- ECDSA sign/verify test on P256 curve.
- ECDH secret agreement Known Answer Test on P256 curve
- SP 800-56A concatenation KDF Known Answer Tests (same as Diffie-Hellman KAT)
- SP 800-90A AES-256 based counter mode random generator Known Answer Tests (instantiate, generate and reseed)
- Power-up Integrity Test (RSA Signature Verification)

BCRYPT.DLL performs the following conditional self-tests:

- CRNGT for SP 800-90A AES-CTR DRBG
- Assurances for SP 800-56A (According to sections 5.5.2, 5.6.2, and 5.6.3 of the standard)
- DRBG health test for SP 800-90A AES-CTR
- CRNGT for the entropy source of the DRBGs.
- Pairwise consistency tests for ECDSA and RSA key generations
- Pairwise consistency tests for Diffie-Hellman and EC Diffie-Hellman prime value generation

In all cases for any failure of a power-on (startup) self-test, BCRYPT.DLL DllMain fails to return the STATUS_SUCCESS status to the operating system. The only way to recover from the failure of a power- on (startup) self-test is to attempt to reload the BCRYPT.DLL, which will rerun the self-tests, and will only succeed if the self-tests passes.

9 Design Assurance

The BCRYPT.DLL crypto module is part of the overall Windows Embedded Compact 2013 operating system, which is a product family that has gone through and is continuously going through the Common Criteria Certification or equivalent under US NIAP CCEVS since Windows NT 3.5. The certification provides the necessary design assurance.

The BCRYPT.DLL is installed and started as part of the Windows Embedded Compact 2013 operating system.

10 Mitigation of Other Attacks

The BCRYPT.DLL crypto module does not provide any mechanisms to mitigate other attacks.

11 Additional details

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

CHANGE HISTORY			
AUTHOR	DATE	VERSION	COMMENT
Tolga Acar	6/7/2007	1.0	Windows Vista FIPS Approval Submission Version
Kevin Michelizzi	2/17/2012	1.1	Windows Embedded Compact 7 Version
Kevin Michelizzi	3/22/2012	1.2	Windows Embedded Compact 7 FIPS Approval Review

Kevin Michelizzi	7/18/2012	1.3	Windows Embedded Compact 7 HMAC and algorithm corrections
Kevin Michelizzi	10/30/2012	1.4	Update with final comments from review
Kevin Michelizzi	07/03/2013	1.5	Update with comments from CMVP
Kevin Michelizzi	07/18/2013	1.6	Add entropy caveat on key generation
Hua Liu	03/15/2017	1.7	Added support for Window Embedded Compact 2013 and updated information relevant to the new DRBG
Subramanyam Kannaboina	06/05/2017	1.8	Update with comments from CMVP
Subramanyam Kannaboina	06/04/2019	1.9	Updated to only Windows Embedded Compact 2013 for FIPS review