Microsoft Windows Server 2008 R2 Cryptographic Primitives Library (bcryptprimitives.dll) Security Policy Document

Microsoft Windows Server 2008 R2 Operating System

FIPS 140-2 Security Policy Document

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

July 7, 2010

Document Version: 2.1

This Security Policy is non-proprietary and may be reproduced only in its original entirety (without revision)

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.
Complying with all applicable copyright laws is the responsibility of the user. This work is licensed under the Creative Commons Attribution-NoDerivs-NonCommercial License (which allows redistribution of the work). To view a copy of this license, visit http://creativecommons.org/licenses/by-nd-nc/1.0/ or send a letter to Creative Commons, 559 Nathan Abbott Way, Stanford, California 94305, USA.Microsoft may have patents, patent applications, trademarks, copyrights, or other intellectual property rights covering subject matter in this document. Except as expressly provided in any written license agreement from Microsoft, the furnishing of this document does not give you any license to these patents, trademarks, copyrights, or other intellectual property.
The example companies, organizations, products, people and events depicted herein are fictitious. No association with any real company, organization, product, person or event is intended or should be inferred.
© 2007 Microsoft Corporation. All rights reserved.
Microsoft, Active Directory, Visual Basic, Visual Studio, Windows, the Windows logo, Windows NT, Windows Server, Windows Vista and Windows 7 are either registered trademarks or trademarks of Microsoft Corporation in the United States and/or other countries.
The names of actual companies and products mentioned herein may be the trademarks of their respective owners.

This Security Policy is non-proprietary and may be reproduced only in its original entirety (without revision)

1 Cryptographic Module Specification

The Microsoft Windows Cryptographic Primitives Library is a general purpose, software-based, cryptographic module. The primitive provider functionality is offered through one cryptographic module, BCRYPTPRIMITIVES.DLL (version 6.1.7600.16385), subject to FIPS-140-2 validation.
BCRYPTPRIMITIVES.DLL provides cryptographic services, through its documented interfaces, to Windows Server 2008 R2 components and applications running on Windows Server 2008 R2.
The cryptographic module, BCRYPTPRIMITIVES.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.

1.1 Cryptographic Boundary

The Windows Server 2008 R2 BCRYPTPRIMITIVES.DLL consists of a dynamically-linked library (DLL). The cryptographic boundary for BCRYPTPRIMITIVES.DLL is defined as the enclosure of the computer system, on which BCRYPTPRIMITIVES.DLL is to be executed. The physical configuration of BCRYPTPRIMITIVES.DLL, as defined in FIPS-140-2, is multi-chip standalone.

2 Security Policy

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

- BCRYPTPRIMITIVES.DLL is supported on Windows Server 2008 R2.
- BCRYPTPRIMITIVES.DLL operates in FIPS mode of operation only when used with the FIPS approved version of Windows Server 2008 R2 Code Integrity (ci.dll) validated to FIPS 140-2 under Cert. #1334 operating in FIPS mode
- Windows Server 2008 R2 is an operating system supporting a “single user” mode where there is only one interactive user during a logon session.
- BCRYPTPRIMITIVES.DLL is only in its Approved mode of operation when Windows is booted normally, meaning Debug mode is disabled and Driver Signing enforcement is enabled.
- BCRYPTPRIMITIVES.DLL 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\SelfTest Algorithms
- All users assume either the User or Cryptographic Officer roles.
- BCRYPTPRIMITIVES.DLL provides no authentication of users. Roles are assumed implicitly. The authentication provided by the Windows Server 2008 R2 operating system is not in the scope of the validation.
- All cryptographic services implemented within BCRYPTPRIMITIVES.DLL are available to the User and Cryptographic Officer roles.
- BCRYPTPRIMITIVES.DLL 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. #686)
    - 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 CCM (Cert. #1187)
    - AES-128, AES-192 and AES-256 GCM (Cert. #1168, vendor-affirmed)
    - AES-128, AES-192, and AES-256 GMAC (Cert#1168, vendor-affirmed)
    - RSA (RSASSA-PKCS1-v1_5 and RSASSA-PSS) digital signatures (Cert. #567) and X9.31 RSA key-pair generation (Cert. #559)
    - DSA (Cert. #391)

    - KAS – SP800-56A (vendor-affirmed) Diffie-Hellman Key Agreement; key establishment methodology provides at least 80-bits of encryption strength.
    - KAS – SP800-56A (vendor-affirmed) EC Diffie-Hellman Key Agreement; key establishment methodology provides between 128 and 256-bits of encryption strength
    - ECDSA with the following NIST curves: P-256, P-384, P-521 (Cert. #142)
    - SP800-90 AES-256 counter mode DRBG (Cert. #23)
    - SP800-90 Dual EC DRBG (Cert. #27)
    - FIPS 186-2 x-Change Notice Regular RNG (Cert. #649).
- BCRYPTPRIMITIVES.DLL supports the following non-Approved algorithms allowed for use in FIPS mode.
    - AES Key Wrap (AES Cert #1168, key wrapping; key establishment methodology provides between 128 and 256 bits of encryption strength)
    - TLS and EAP-TLS
    - IKEv1 Key Derivation Functions
- BCRYPTPRIMITIVES.DLL also supports the following non FIPS 140-2 approved algorithms, though these algorithms may not be used when operating the module 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 BCRYPTPRIMITIVES.DLL 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.

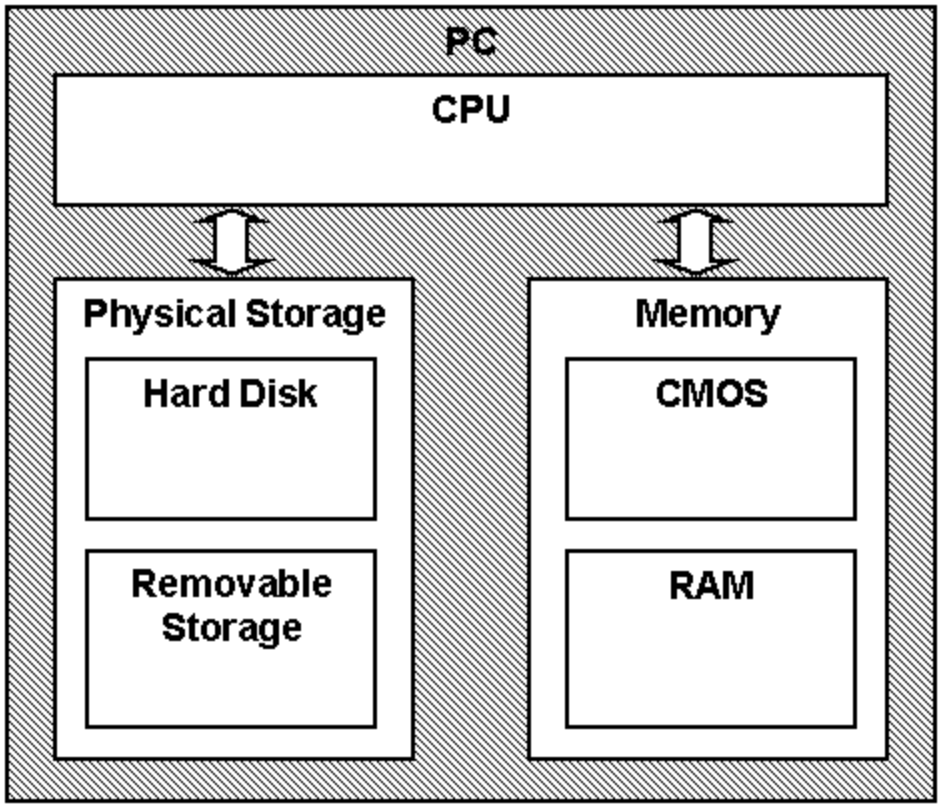

Figure 1 Master components of bcryptprimitives.dll module

BCRYPTPRIMITIVES.DLL was tested using the following machine configurations:

x64	Microsoft Windows Server 2008 R2 Ultimate Edition (x64 version) – HP Compaq dc7600
IA64	Microsoft Windows Server 2008 R2 Ultimate Edition (IA64 version) – HP zx2000

3 Cryptographic Module Ports and Interfaces

3.1 Ports and Interfaces

3.1.1 Export Functions

The BCRYPTPRIMITIVES.DLL module implements a set of algorithm providers for the Cryptography Next Generation (CNG) framework in Windows. Each provider in this module represents a single cryptographic algorithm or a set of closely related cryptographic algorithms. These algorithm providers are invoked through the CNG algorithm primitive functions, which are sometimes collectively referred to as the BCrypt API. For a full list of these algorithm providers, see http://msdn.microsoft.com/en-us/library/aa375534.aspx.

The BCRYPTPRIMITIVES.DLL module exposes its cryptographic services to the operating system through a small set of exported functions. These functions are used by the CNG framework to retrieve references to the different algorithm providers, in order to route BCrypt API calls appropriately to BCRYPTPRIMITIVES.DLL. For details, please see the CNG SDK, available at http://www.microsoft.com/downloads/details.aspx?familyid=1EF399E9-B018-49DB-A98B-0CED7CB8FF6F&displaylang=en.

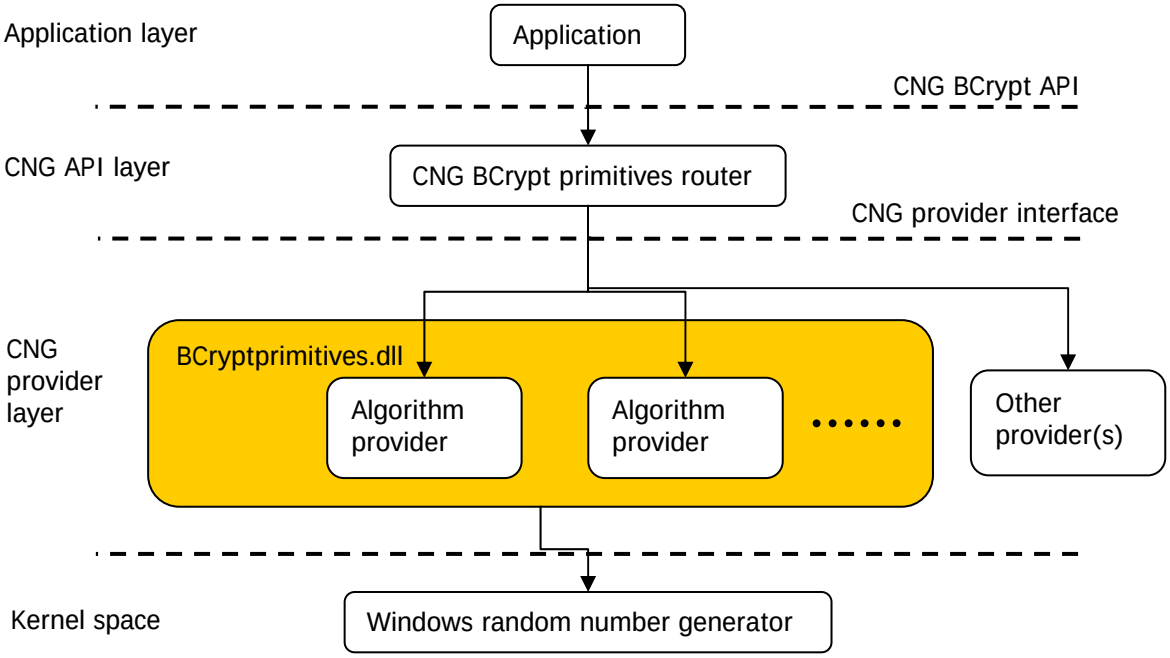

Figure 2 Relationships between bcryptprimitives.dll and other components – cryptographic boundary highlighted in gold.

The following functions are exported by BCRYPTPRIMITIVES.DLL:

- GetAsymmetricEncryptionInterface
- GetCipherInterface
- GetHashInterface
- GetRngInterface
- GetSecretAgreementInterface
- GetSignatureInterface

3.1.2 CNG Primitive Functions

The following list contains the CNG functions which can be used by callers to access the cryptographic services in BCRYPTPRIMITIVES.DLL.

- 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

3.1.3 Data Input and Output Interfaces

The Data Input Interface for BCRYPTPRIMITIVES.DLL consists of the CNG primitive functions listed in Section 3.1.2. Data and options are passed to the interface as input parameters to the CNG primitive functions. Data Input is kept separate from Control Input by passing Data Input in separate parameters from Control Input.

The Data Output Interface for BCRYPTPRIMITIVES.DLL also consists of the CNG primitive functions.

3.1.4 Control Input Interface

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

3.1.5 Status Output Interface

The Status Output Interface for BCRYPTPRIMITIVES.DLL also consists of the CNG primitive 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 BCRYPTPRIMITIVES.DLL.

4 Roles and Authentication

4.1 Roles

BCRYPTPRIMITIVES.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 BCRYPTPRIMITIVES.DLL.

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 BCRYPTPRIMITIVES.DLL executes (Microsoft Windows Server 2008 R2) does authenticate users.

Microsoft Windows Server 2008 R2 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 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.

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. BCRYPTPRIMITVES.DLL implements three random number generation algorithms:

- BCRYPT_RNG_ALGORITHM. This is the AES-256 counter mode based random generator as defined in SP800-90.
- BCRYPT_RNG_DUAL_EC_ALGORITHM. This is the dual elliptic curve based random generator as defined in SP800-90.
- BCRYPT_RNG_FIPS186_DSA_ALGORITHM. This is the random number generator required by the DSA algorithm as defined in FIPS 186-2.

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 generation algorithm.

During the function initialization, a seed is obtained from the output of an in-kernel random number generator. This RNG, which exists beyond the cryptographic boundary, provides the necessary entropy for the user-level RNGs available through this function.

5.3 Key and Key-Pair Generation

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_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.

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_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_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_AES_WRAP_KEY_BLOB, 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 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.

5.5.2 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 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.

5.6 Hashing and Message Authentication

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 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.

5.6.2 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.

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.

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 must be zero.

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), 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.

5.8.3 BCryptDestroySecret

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

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

6 Operational Environment

BCRYPTPRIMITIVES.DLL is intended to run on Windows Server 2008 R2 in Single User mode as defined in Section 2. When run in this configuration, multiple concurrent operators are not supported. Because BCRYPTPRIMITIVES.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

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

7.1 Cryptographic Keys, CSPs, and SRDIs

The BCRYPTPRIMITIVES.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 TDEA encryption/decryption.
HMAC keys	Keys used for HMAC-SHA1, HMAC-SHA256, HMAC-SHA384, and HMAC-SHA512
DSA Public Keys	Keys used for the verification of DSA digital signatures
DSA Private Keys	Keys used for the calculation of DSA digital signatures
ECDSA Public Keys	Keys used for the verification of ECDSA digital signatures
ECDSA Private Keys	Keys used for the calculation of ECDSA digital signatures
RSA Public Keys	Keys used for the verification of RSA digital signatures
RSA Private Keys	Keys used for the calculation of RSA digital signatures
DH Public and Private values	Public and private values used for Diffie-Hellman key establishment.
ECDH Public and Private values	Public and private values used for EC Diffie-Hellman key establishment.

7.2 Access Control Policy

The BCRYPTPRIMITIVES.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.

BCRYPTPRIMITIVES.DLL crypto module SRDI/Service Access Policy	Symmetric encryption and decryption keys	HMAC keys	DSA Public Keys	DSA Private Keys	ECDSA public keys	ECDSA Private keys	RSA Public Keys	RSA Private Keys	DH Public and Private values	ECDH Public and Private values
Cryptographic Module Power Up and Power Down										
Key Formatting	w									
Random Number Generation										
Data Encryption and Decryption	x									
Hashing		xw								
Acquiring a Table of Pointers to BCryptXXX Functions										
Algorithm Providers and Properties										
Key and Key-Pair Generation	wd	wd	wd	wd	wd	wd	wd	wd	wd	wd
Key Entry and Output	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw
Signing and Verification			x	x	x	x	x	x		
Secret Agreement and Key Derivation									x	x

7.3 Key Material

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

7.4 Key Generation

BCRYPTPRIMITIVES.DLL can create and use keys for the following algorithms: RSA, DSA, 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, DSA, DH, and ECDH keys and key-pairs are generated following the techniques given in section 5.2.

7.5 Key Establishment

BCRYPTPRIMITIVES.DLL 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.
BCRYPTPRIMITIVES.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_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 BCRYPTPRIMITIVES.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

BCRYPTPRIMITIVES.DLL does not provide persistent storage of keys.

7.8 Key Archival

BCRYPTPRIMITIVES.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 operator calls BCryptDestroyKey() or BCryptDestroySecret() on that key handle.

8 Self-Tests

BCRYPTPRIMITIVES.DLL performs the following power-on (start up) self-tests when DllMain is called by the operating system.

- HMAC-SHA-1Known Answer Test
- SHA-256 and SHA-512 Known Answer Tests
- Triple-DES encrypt/decrypt ECB Known Answer Test
- AES-128 encrypt/decrypt ECB 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
- DSA sign/verify test with 1024-bit key
- RSA sign and verify test with 2048-bit key
- DH secret agreement Known Answer Test with 1024-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 DSA random generator Known Answer Tests
- SP800-90 AES-256 based counter mode random generator Known Answer Tests (instantiate, generate and reseed)
- SP800-90 dual elliptic curve random generator Known Answer Tests (instantiate, generate and reseed)

BCRYPTPRIMITIVES.DLL performs pair-wise consistency checks upon each invocation of RSA, ECDH, DSA, and ECDSA key-pair generation and import as defined in FIPS 140-2. BCRYPTPRIMITIVES.DLL also performs a continuous RNG test on each of the implemented RNGs as defined in FIPS 140-2.

In all cases for any failure of a power-on (start up) self-test, BCRYPTPRIMITIVES.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 (start up) self-test is to attempt to reload the BCRYPTPRIMITIVES.DLL, which will rerun the self-tests, and will only succeed if the self-tests passes.

9 Design Assurance

The BCRYPTPRIMITIVES.DLL crypto module is part of the overall Windows Server 2008 R2 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 BCRYPTPRIMITIVES.DLL is installed and started as part of the Windows Server 2008 R2 operating system.

10 Additional details

For the latest information on Windows Server 2008 R2, check out the Microsoft web site at http://www.microsoft.com.

CHANGE HISTORY			
AUTHOR	**DATE**	**VERSION**	**COMMENT**
Tolga Acar	6/7/2007	1.0	FIPS Approval Submission
Stefan Santesson	10/30/2007	1.1	Added technical updates related to SP1 and WS2K8
Stefan Santesson	2/15/2008	1.2	Merged changes resulting from Gold CMVP review
Vijay Bharadwaj	2/28/2008	1.3	Revisions for SP1 and WS08
Vijay Bharadwaj	5/5/2009	2.0	Windows Server 2008 R2 changes – moving from BCRYPT to BCRYPTPRIMITIVES
Vijay Bharadwaj	7/7/2010	2.1	Updates to address CMVP comments