Microsoft Windows

FIPS 140 Validation

Microsoft Windows 10 (Fall Creators Update, April 2018 Update, October 2018 Update)

Microsoft Windows 10 Mobile (Fall Creators Update)

Microsoft Windows Server (versions 1709, 1803) and Windows Server 2019

Microsoft Azure Data Box Edge

Non-Proprietary

Security Policy Document

Version Number	1.4
Updated On	May 7, 2020

Version History

Version	Date	Summary of changes
1.0	October 3, 2017	Draft sent to NIST CMVP
1.1	November 18, 2017	Updates for Windows 10 version 1709
1.2	May 1, 2018	Editing updates
1.3	October 22, 2018	Updates for Windows 10 version 1803
1.4	May 7, 2020	Updates for Windows 10 version 1809

TABLE OF CONTENTS

SECURITY POLICY DOCUMENT 1

VERSION HISTORY 3

1 INTRODUCTION 8

1.1 LIST OF CRYPTOGRAPHIC MODULE BINARY EXECUTABLES 8
1.2 VALIDATED PLATFORMS 9
1.3 CONFIGURE WINDOWS TO USE FIPS-APPROVED CRYPTOGRAPHIC ALGORITHMS 14

2 CRYPTOGRAPHIC MODULE SPECIFICATION 14

2.1 CRYPTOGRAPHIC BOUNDARY 14
2.2 FIPS 140-2 APPROVED ALGORITHMS 15
2.3 NON-APPROVED ALGORITHMS 17
2.4 FIPS 140-2 APPROVED ALGORITHMS FROM BOUNDED MODULES 18
2.5 CRYPTOGRAPHIC BYPASS 20
2.6 HARDWARE COMPONENTS OF THE CRYPTOGRAPHIC MODULE 20

3 CRYPTOGRAPHIC MODULE PORTS AND INTERFACES 20

3.1 EXPORT FUNCTIONS 20
3.2 CNG PRIMITIVE FUNCTIONS 21
3.2.1 ALGORITHM PROVIDERS AND PROPERTIES 22
3.2.1.1 BCryptOpenAlgorithmProvider 22
3.2.1.2 BCryptCloseAlgorithmProvider 22
3.2.1.3 BCryptSetProperty 22
3.2.1.4 BCryptGetProperty 23
3.2.1.5 BCryptFreeBuffer 23
3.2.2 KEY AND KEY-PAIR GENERATION 23
3.2.2.1 BCryptGenerateSymmetricKey 23
3.2.2.2 BCryptGenerateKeyPair 24
3.2.2.3 BCryptFinalizeKeyPair 24
3.2.2.4 BCryptDuplicateKey 24
3.2.2.5 BCryptDestroyKey 24
3.2.3 RANDOM NUMBER GENERATION 25
3.2.3.1 BCryptGenRandom 25
3.2.4 KEY ENTRY AND OUTPUT 25

3.2.4.1 BCryptImportKey ... 25
3.2.4.2 BCryptImportKeyPair ... 25
3.2.4.3 BCryptExportKey ... 26
3.2.5 ENCRYPTION AND DECRYPTION ... 26
3.2.5.1 BCryptEncrypt ... 26
3.2.5.2 BCryptDecrypt ... 26
3.2.6 HASHING AND MESSAGE AUTHENTICATION ... 27
3.2.6.1 BCryptCreateHash ... 27
3.2.6.2 BCryptHashData ... 27
3.2.6.3 BCryptDuplicateHash ... 27
3.2.6.4 BCryptFinishHash ... 27
3.2.6.5 BCryptDestroyHash ... 28
3.2.6.6 BCryptHash ... 28
3.2.6.7 BCryptCreateMultiHash ... 28
3.2.6.8 BCryptProcessMultiOperations ... 28
3.2.7 SIGNING AND VERIFICATION ... 29
3.2.7.1 BCryptSignHash ... 29
3.2.7.2 BCryptVerifySignature ... 29
3.2.8 SECRET AGREEMENT AND KEY DERIVATION ... 29
3.2.8.1 BCryptSecretAgreement ... 29
3.2.8.2 BCryptDeriveKey ... 29
3.2.8.3 BCryptDestroySecret ... 30
3.2.8.4 BCryptKeyDerivation ... 30
3.2.8.5 BCryptDeriveKeyPBKDF2 ... 30
3.2.9 CRYPTOGRAPHIC TRANSITIONS ... 31
3.2.9.1 Bit Strengths of DH and ECDH ... 31
3.2.9.2 SHA-1 ... 31
3.3 CONTROL INPUT INTERFACE ... 31
3.4 STATUS OUTPUT INTERFACE ... 31
3.5 DATA OUTPUT INTERFACE ... 31
3.6 DATA INPUT INTERFACE ... 31
3.7 NON-SECURITY RELEVANT CONFIGURATION INTERFACES ... 31

4 ROLES, SERVICES AND AUTHENTICATION ... 33

4.1 ROLES ... 33
4.2 SERVICES ... 33
4.2.1 MAPPING OF SERVICES, ALGORITHMS, AND CRITICAL SECURITY PARAMETERS ... 33
4.2.2 MAPPING OF SERVICES, EXPORT FUNCTIONS, AND INVOCATIONS ... 35
4.2.3 NON-APPROVED SERVICES ... 36
4.3 AUTHENTICATION ... 37

5 FINITE STATE MODEL....37

5.1 SPECIFICATION....37

6 OPERATIONAL ENVIRONMENT....37

6.1 SINGLE OPERATOR....38
6.2 CRYPTOGRAPHIC ISOLATION....38
6.3 INTEGRITY CHAIN OF TRUST....38

7 CRYPTOGRAPHIC KEY MANAGEMENT....40

7.1 ACCESS CONTROL POLICY....41
7.2 KEY MATERIAL....42
7.3 KEY GENERATION....42
7.4 KEY ESTABLISHMENT....42
7.4.1 NIST SP 800-132 PASSWORD BASED KEY DERIVATION FUNCTION (PBKDF)....43
7.4.2 NIST SP 800-38F AES KEY WRAPPING....43
7.5 KEY ENTRY AND OUTPUT....43
7.6 KEY STORAGE....44
7.7 KEY ARCHIVAL....44
7.8 KEY ZEROIZATION....44

8 SELF-TESTS....44

8.1 POWER-ON SELF-TESTS....44
8.2 CONDITIONAL SELF-TESTS....45

9 DESIGN ASSURANCE....45

10 MITIGATION OF OTHER ATTACKS....46

11 SECURITY LEVELS....46

12 ADDITIONAL DETAILS....47

13 APPENDIX A – HOW TO VERIFY WINDOWS VERSIONS AND DIGITAL SIGNATURES....48

13.1 HOW TO VERIFY WINDOWS VERSIONS....48

13.2 HOW TO VERIFY WINDOWS DIGITAL SIGNATURES 48

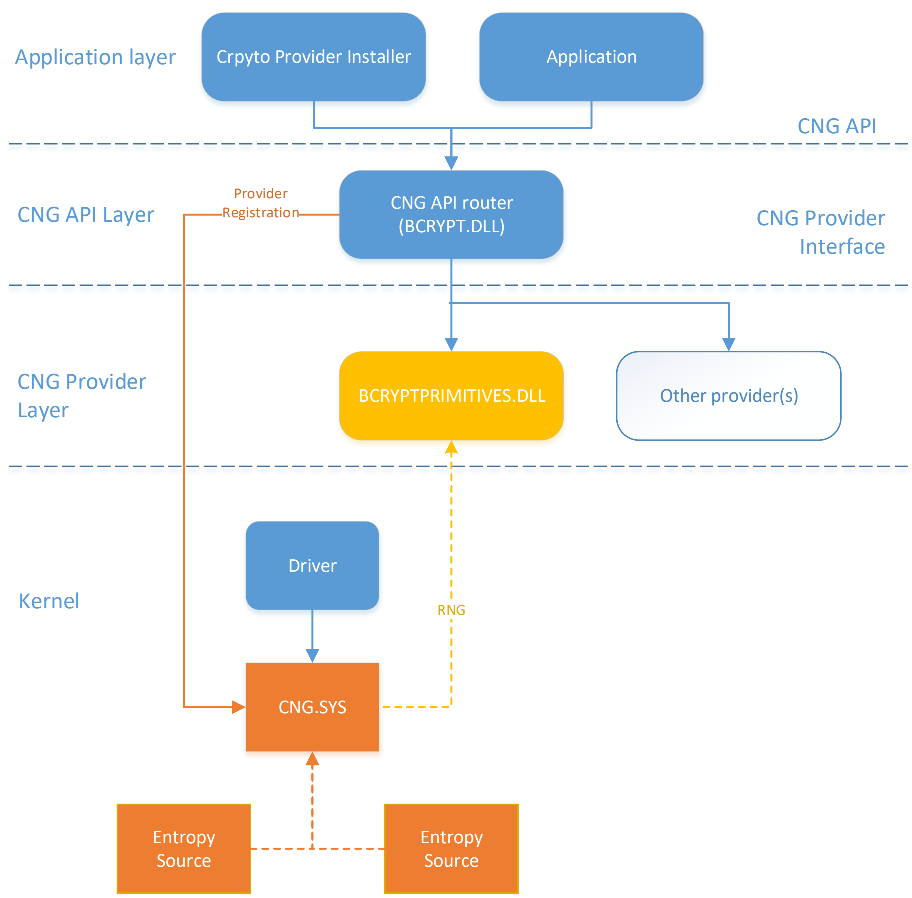
14 APPENDIX B – REFERENCES 49

1 Introduction

The Microsoft Windows Cryptographic Primitives Library is a general purpose, software-based cryptographic module. Cryptographic Primitives Library provides cryptographic services to user-mode applications running on the Windows operating system.

Cryptographic Primitives Library encapsulates several different cryptographic algorithms accessible via the Microsoft CNG (Cryptography, Next Generation) API which are exported by BCRYPT.DLL. BCRYPT.DLL is an API wrapper for BCRYPTPRIMITIVES.DLL and can be linked into applications by software developers to permit the use of general-purpose FIPS 140-2 Level 1 compliant cryptography.

The relationship between Cryptographic Primitives Library and other components is shown in the following diagram:



1.1 List of Cryptographic Module Binary Executables

The Microsoft Windows Cryptographic Primitives Library cryptographic module contains the following binaries:

- BCRYPTPRIMITIVES.DLL

The Windows builds covered by this validation are:

- Windows 10 version 1709 and Windows Server build version 1709 10.0.16299
- Windows 10 Mobile version 1709 build 10.0.15254
- Microsoft Surface Hub build 10.0.15063.674
- Windows 10 version 1803 and Windows Server version 1803 build 10.0.17134
- Windows 10 version 1809 and Windows Server 2019 build 10.0.17763
- Microsoft Azure Data Box Edge build 10.0.17763

1.2 Validated Platforms

The Windows editions covered by this validation are:

- Windows 10 Home Edition (32-bit version)
- Windows 10 Pro Edition (64-bit version)
- Windows 10 Enterprise Edition (64-bit version)
- Windows 10 Education Edition (64-bit version)
- Windows 10 S Edition (64-bit version)
- Windows 10 Mobile
- Microsoft Surface Hub
- Windows Server Standard Core
- Windows Server Datacenter Core
- Microsoft Azure Data Box Edge

The Cryptographic Primitives Library components listed in Section 1.1 were validated using the combination of computers and Windows operating system editions specified in the table below.

All the computers for Windows 10 and Windows Server listed in the table below are all 64-bit Intel architecture and implement the AES-NI instruction set but not the SHA Extensions. The exceptions are:

- Dell Inspiron 660s - Intel Core i3 without AES-NI and SHA Extensions
- HP Slimline Desktop - Intel Pentium with AES-NI and SHA Extensions

Windows 10 Mobile runs on the ARM architecture, which does not implement AES-Ni instructions or SHA extensions:

- Microsoft Lumia 950 - Qualcomm Snapdragon 808 (A57, A53)
- Microsoft Lumia 950 XL - Qualcomm Snapdragon 810 (A57, A53)
- Microsoft Lumia 650 - Qualcomm Snapdragon 212 (A7)
- HP Elite x3 - Qualcomm Snapdragon 820 (Kryo)

Table 1 Validated Platforms for Windows 10 Fall Creators Update and Windows Server version 1709

Computer	Windows 10 Home	Windows 10 Pro	Windows 10 Enterprise	Windows 10 Education	Surface Hub	Windows 10 S	Windows 10 Mobile	Windows Server Standard	Windows Server Datacenter
Microsoft Surface Book 2			√						
Microsoft Surface Laptop		√	√			√			
Microsoft Surface Pro		√	√	√					
Microsoft Surface Book			√						
Microsoft Surface Pro 4			√						
Microsoft Surface Pro 3		√							
Microsoft Surface 3 with LTE		√							
Microsoft Surface Studio			√						
Microsoft Surface Hub					√				
Windows Server Standard Core Hyper-V[1]								√	√
Windows Server 2016 Hyper-V[2]		√							
Microsoft Lumia 950							√		
Microsoft Lumia 950 XL							√		
Microsoft Lumia 650							√		
Dell Latitude 5285		√							
Dell Latitude 5290		√							
Dell Inspiron 660s	√								
Dell Precision Tower 5810MT		√						√	√

[1] Hardware platform: Dell 5810MT
[2] Hardware platform: Surface Pro 4

Dell PowerEdge R630		√						√	√
Dell PowerEdge R740								√	√
HP Elite X3							√		
HP Compaq Pro 6305		√							
HP Pro x2 612 G2 Detachable PC with LTE			√						
HP Slimline Desktop		√							
Panasonic Toughbook		√							

Table 2 Validated Platforms for Windows 10 and Windows Server version 1803

Computer	Windows 10 Home	Windows 10 Pro	Windows 10 Enterprise	Windows 10 Education	Windows Server Standard	Windows Server Datacenter
Microsoft Surface Go		√				
Microsoft Surface Book 2		√	√			
Microsoft Surface Pro LTE		√	√			
Microsoft Surface Laptop		√	√	√		
Microsoft Surface Studio			√			
Windows Server Standard Core Hyper-V[3]					√	√
Windows Server 2016 Hyper-V[4]					√	
Dell Latitude 5290		√				

[3] Hardware platform: Dell Precision Tower 5810MT
[4] Hardware platform: Dell PowerEdge R740

Computer	Windows 10 Home	Windows 10 Pro	Windows 10 Enterprise	Windows 10 Education	Windows Server Standard	Windows Server Datacenter
Dell Latitude 12 Rugged Tablet		√				
Dell Inspiron 660s	√					
Dell PowerEdge R740					√	√
HP Pro x2 612 G2 Detachable PC with LTE			√			
HP Slimline Desktop		√				
Microsoft Surface Pro 6		√				
Microsoft Surface Laptop 2		√				
Microsoft Surface Studio 2			√			

Table 3 Validated Platforms for Windows 10 and Windows Server version 1809 and Azure Data Box Edge

Computer	Windows 10 Home	Windows 10 Pro	Windows 10 Enterprise	Windows 10 Education	Windows Server 2019	Windows Server 2019 Datacenter	Azure Data Box Edge
Microsoft Surface Go – Intel Pentium		√					
Microsoft Surface Book 2 – Intel Core i7		√	√				
Microsoft Surface Pro LTE – Intel Core i5		√	√				
Microsoft Surface Laptop – Intel Core i5		√	√	√			

Microsoft Surface Studio – Intel Core i7			√				
Microsoft Windows Server 2019 Hyper-V[5]					√	√	
Microsoft Windows Server 2016 Hyper-V[6]					√		
Dell Latitude 12 Rugged Tablet – Intel Core i5		√					
Dell Latitude 5290 – Intel Core i7			√				
Dell PowerEdge R740 – Intel Xeon Gold					√	√	
Dell Inspiron 660s [with x86 Windows] – Intel Core i3	√						
HP Slimline Desktop – Intel Pentium		√					
HP Elite x2 1013 G3 Tablet – Intel Core i7		√					
HP EliteBook x360 1030 G2 – Intel Core i7			√				
Samsung Galaxy Book 10.6” – Intel Core m3		√					
Samsung Galaxy Book 12” – Intel Core i5			√				

[5] Hardware Platform: Dell Precision Tower 5810MT – Intel Xeon E5
[6] Hardware Platform: Dell PowerEdge R740 Server – Intel Xeon Gold

Microsoft Azure Data Box Edge – Intel Xeon Silver							√

1.3 Configure Windows to use FIPS-Approved Cryptographic Algorithms

Use the FIPS Local/Group Security Policy setting or a Mobile Device Management (MDM) to enable FIPS-Approved mode for Cryptographic Primitives Library.

The Windows operating system provides a group (or local) security policy setting, "**System cryptography: Use FIPS compliant algorithms for encryption, hashing, and signing**".

Consult the MDM documentation for information on how to enable FIPS-Approved mode. The Policy CSP - Cryptography includes the setting **AllowFipsAlgorithmPolicy** which is the setting the MDM will configure.

Changes to the Approved mode security policy setting do not take effect until the computer has been rebooted.

2 Cryptographic Module Specification

Cryptographic Primitives Library is a multi-chip standalone module that operates in FIPS-approved mode during normal operation of the computer and Windows operating system and when Windows is configured to use FIPS-approved cryptographic algorithms as described in Configure Windows to use FIPS-Approved Cryptographic Algorithms.

In addition to configuring Windows to use FIPS-Approved Cryptographic Algorithms, third-party applications and drivers installed on the Windows platform must not use any of the non-approved algorithms implemented by this module. Windows will not operate in an Approved mode when the operators chooses to use a non-Approved algorithm or service.

The following configurations and modes of operation will cause Cryptographic Primitives Library to operate in a non-approved mode of operation:

- Boot Windows in Debug mode
- Boot Windows with Driver Signing disabled
- Windows enters the ACPI S4 power state (for Windows 10 version 1803 and 1809)

2.1 Cryptographic Boundary

The software cryptographic boundary for Cryptographic Primitives Library is defined as the binary BCRYPTPRIMITIVES.DLL.

2.2 FIPS 140-2 Approved Algorithms

Cryptographic Primitives Library implements the following FIPS-140-2 Approved algorithms:[7]

Algorithm	Windows 10 and Windows Server version 1709	Windows 10 Mobile version 1709	Microsoft Surface Hub (10.0.15063.674)	Windows 10 and Windows Server version 1803	Windows 10 version 1809 and Windows Server 2019	Azure Data Box Edge
FIPS 180-4 SHS SHA-1, SHA-256, SHA-384, and SHA-512	#4009	#4010	#4011	#4633	#C211	#C211
FIPS PUB 198-1 HMAC-SHA-1[8] and HMAC-SHA-256	#3267	#3268	#3269	#3858	#C211	#C211
FIPS 197 AES-128, AES-192, and AES-256 in ECB, CBC, CFB8, CFB128, and CTR modes	#4897	#4901	#4902	# 5847	#C211	#C211
NIST SP 800-38B and SP 800-38C AES-128, AES-192, and AES-256 in CCM and CMAC modes	#4897	#4901	#4902	#5847	#C211	#C211
NIST SP 800-38D AES-128, AES-192, and AES-256 GCM decryption and GMAC	#4897	#4901	#4902	#5847	#C211	#C211
NIST SP 800-38E XTS-AES XTS-128 and XTS-256[9]	#4897	#4901	#4902	#5847	#C211	#C211
FIPS 186-4 RSA PKCS#1 (v1.5) digital signature generation and verification with 1024, 2048, and 3072 moduli; supporting SHA-1[10], SHA-256, SHA-384, and SHA-512	#2667	#2670	#2671	#3079	#C211	#C211
FIPS 186-4 RSA key-pair generation with 2048 and 3072 moduli	#2667	#2670	#2671	#3079	#C211	#C211
FIPS 186-4 ECDSA key pair generation and verification, signature generation and verification with the following NIST curves: P-256, P-384, P-521	#1246	#1249	#1250	#1563	#C211	#C211

[7] This module may not use some of the capabilities described in each CAVP certificate.
[8] For HMAC, only key sizes that are >= 112 bits in length are used by the module in FIPS mode.
[9] AES XTS must be used only to protect data at rest and the caller needs to ensure that the length of data encrypted does not exceed 2^{20} AES blocks.
[10] SHA-1 is only acceptable for legacy signature verification.

FIPS 186-4 DSA PQG generation and verification, signature generation and verification (L=2048, N=256; L=3072, N=256)	#1301	#1302	#1303	#1479	#C211	#C211
KAS – 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	#146	#147	#148	#200	#C211	#C211
KAS – 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	#146	#147	#148	#200	#C211	#C211
NIST SP 800-56B RSADP mod 2048	#1498	#1509	#1513	#2111	#C211	#C211
NIST SP 800-90A AES-256 counter mode DRBG	#1730	#1731	#1732	#2435	#C211	#C211
NIST SP 800-67r1 Triple-DES (2 key legacy-use decryption[11] and 3 key encryption/decryption) in ECB, CBC, CFB8 and CFB64 modes	#2556	#2557	#2558	#2862	#C211	#C211
NIST SP 800-108 Key Derivation Function (KDF) CMAC-AES (128, 192, 256), HMAC (SHA1, SHA-256, SHA-384, SHA-512)	#157	#158	#159	#242	#C347	#C347
NIST SP 800-38F AES Key Wrapping (128, 192, and 256)	#4898	#4899	#4900	#5860	#C347	#C347

[11] Two-key Triple-DES Decryption is only allowed for Legacy-usage (as per SP 800-131A). The use of two-key Triple-DES Encryption is disallowed. The caller is responsible for using the key for up to 2^20 encryptions for IETF protocols and 2^16 encryptions for any other use.

NIST SP 800-135 IKEv1, IKEv2 and TLS KDF primitives[12]	#1496	#1507	#1511	#2110	#C211	#C211
NIST SP 800-132 KDF (also known as PBKDF) with HMAC (SHA-1, SHA-256, SHA-384, SHA-512) as the pseudo-random function	Vendor affirmed					

2.3 Non-Approved Algorithms

Mode Cryptographic Primitives Library implements the following non-approved algorithms:

- SHA-1 hash, which is disallowed for use in digital signature generation. It can be used for legacy digital signature verification. Its use is Acceptable for non-digital signature generation applications.
- If HMAC-SHA1 is used, key sizes less than 112 bits (14 bytes) are not allowed for usage in HMAC generation, as per SP 800-131A.
- RSA 1024-bits for digital signature generation, which is disallowed.
- FIPS 186-2 DSA with key length of 1024 bits
- NIST SP 800-56A Key Agreement using Finite Field Cryptography (FFC) with parameter FA (p=1024, q=160). The key establishment methodology provides 80 bits of encryption strength instead of the Approved 112 bits of encryption strength listed above.
- MD5 and HMAC-MD5 (allowed in TLS and EAP-TLS)
- RC2, RC4, MD2, MD4 (disallowed in FIPS mode)
- 2-Key Triple-DES Encryption, which is disallowed for usage altogether as of the end of 2015.
- DES in ECB, CBC, CFB8 and CFB64 modes (disallowed in FIPS mode)
- Legacy CAPI KDF (proprietary; disallowed in FIPS mode)
- HKDF (disallowed in FIPS mode)
- RSA encrypt/decrypt (disallowed in FIPS mode)
- IEEE 1619-2007 XTS-AES, XTS-128 and XTS-256
- NIST SP 800-38D AES-128, AES-192, and AES-256 GCM encryption
- ECDH with the following curves that are allowed in FIPS mode as per FIPS 140-2 IG A.2

Curve	Security Strength (bits)	Allowed in FIPS mode
Curve25519	128	Yes
brainpoolP160r1	80	No
brainpoolP192r1	96	No
brainpoolP192t1	96	No
brainpoolP224r1	112	Yes

[12] This cryptographic module supports the TLS, IKEv1, and IKEv2 protocols with SP 800-135 rev 1 KDF primitives, however, the protocols have not been reviewed or tested by the NIST CAVP and CMVP.

Curve	Security Strength (bits)	Allowed in FIPS mode
brainpoolP224t1	112	Yes
brainpoolP256r1	128	Yes
brainpoolP256t1	128	Yes
brainpoolP320r1	160	Yes
brainpoolP320t1	160	Yes
brainpoolP384r1	192	Yes
brainpoolP384t1	192	Yes
brainpoolP512r1	256	Yes
brainpoolP512t1	256	Yes
ec192wapi	96	No
nistP192	96	No
nistP224	112	Yes
numsP256t1	128	Yes
numsP384t1	192	Yes
numsP512t1	256	Yes
secP160k1	80	No
secP160r1	80	No
secP160r2	80	No
secP192k1	96	No
secP192r1	96	No
secP224k1	112	Yes
secP224r1	112	Yes
secP256k1	128	Yes
secP256r1	128	Yes
secP384r1	192	Yes
secP521r1	256	Yes
wtls12	112	Yes
wtls7	80	No
wtls9	80	No
x962P192v1	96	No
x962P192v2	96	No
x962P192v3	96	No
x962P239v1	120	Yes
x962P239v2	120	Yes
x962P239v3	120	Yes
x962P256v1	128	Yes

2.4 FIPS 140-2 Approved Algorithms from Bounded Modules

A bounded module is a FIPS 140 module which provides cryptographic functionality that is relied on by a downstream module. As described in the Integrity Chain of Trust section, the Cryptographic Primitives Library depends on the following modules and algorithms:

When Hypvisor Code Integrity (HVCI) is not enabled, Code Integrity version 1709 (module certificate #3195) provides:

- CAVP certificates #2668 (Windows 10 and Windows Server), #2669 (Windows 10 Mobile), #2672 (Surface Hub) for FIPS 186-4 RSA PKCS#1 (v1.5) digital signature verification with 2048 moduli; supporting SHA-256
- CAVP certificates #4009 (Windows 10 and Windows Server), #4010 (Windows 10 Mobile), #4011 (Surface Hub) for FIPS 180-4 SHS SHA-256

When Memory Integrity, called HVCI in previous Windows 10 versions, is not enabled, Code Integrity version 1803 (module certificate #3195) provides:

- CAVP certificates #3080 (Windows 10 and Windows Server for FIPS 186-4 RSA PKCS#1 (v1.5) digital signature verification with 2048 moduli; supporting SHA-256
- CAVP certificates #4633 (Windows 10 and Windows Server) for FIPS 180-4 SHS SHA-256

When Memory Integrity, called HVCI in previous Windows 10 versions, is not enabled, Code Integrity version 1809 (module certificate #3644) provides:

- CAVP certificates #C211 (Windows 10 and Windows Server for FIPS 186-4 RSA PKCS#1 (v1.5) digital signature verification with 2048 moduli; supporting SHA-256
- CAVP certificates #C211(Windows 10 and Windows Server) for FIPS 180-4 SHS SHA-256

When HVCI is enabled, Secure Kernel Code Integrity version 1709 (module certificate #3096) provides:

- CAVP certificate #3080 (Windows 10 and Windows Server) for FIPS 186-4 RSA PKCS#1 (v1.5) digital signature verification with 2048 moduli; supporting SHA-256
- CAVP certificate #4633 (Windows 10 and Windows Server) for FIPS 180-4 SHS SHA-256

When Memory Integrity is enabled, Secure Kernel Code Integrity version 1803 (module certificate #3096) provides:

- CAVP certificate #3080 (Windows 10 and Windows Server) for FIPS 186-4 RSA PKCS#1 (v1.5) digital signature verification with 2048 moduli; supporting SHA-256
- CAVP certificate #4633 (Windows 10 and Windows Server) for FIPS 180-4 SHS SHA-256

When Memory Integrity is enabled, Secure Kernel Code Integrity version 1809 (module certificate #3651) provides:

- CAVP certificates #C211 (Windows 10 and Windows Server for FIPS 186-4 RSA PKCS#1 (v1.5) digital signature verification with 2048 moduli; supporting SHA-256
- CAVP certificates #C211 (Windows 10 and Windows Server) for FIPS 180-4 SHS SHA-256

The Cryptographic Primitives Library depends on Kernel Mode Cryptographic Primitives (module certificate #3196) non-deterministic random number generator (NDRNG) for AES-CTR DRBG Entropy

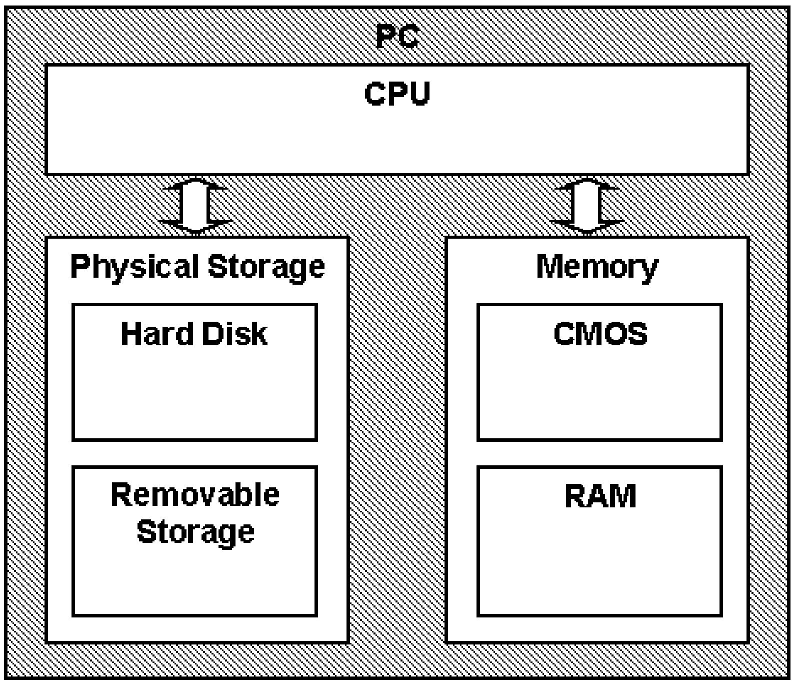
Input. The NDRNG that provides the entropy is not a FIPS Approved algorithm, but is allowed by FIPS 140.

2.5 Cryptographic Bypass

Cryptographic bypass is not supported by Cryptographic Primitives Library.

2.6 Hardware Components of the Cryptographic Module

The physical boundary of the module is the physical boundary of the computer that contains the module. The following diagram illustrates the hardware components used by the Cryptographic Primitives Library module:



3 Cryptographic Module Ports and Interfaces

3.1 Export Functions

The Cryptographic Primitives Library 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 https://msdn.microsoft.com/en-us/library/aa375534.aspx

The Cryptographic Primitives Library module exposes its cryptographic services to the operating system through a 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 Cryptographic Primitives Library. These functions return references to implementations of cryptographic functions that correspond directly to functions in the BCrypt API. For details, please see the CNG SDK for Windows 10, available at https://msdn.microsoft.com/en-us/library/windows/desktop/aa376210.aspx

The following functions are exported by Cryptographic Primitives Library:

- GetAsymmetricEncryptionInterface
- GetCipherInterface
- GetHashInterface
- GetKeyDerivationInterface
- GetRngInterface
- GetSecretAgreementInterface
- GetSignatureInterface
- ProcessPrng
- ProcessPrngGuid

3.2 CNG Primitive Functions

The following list contains the CNG functions which can be used by callers to access the cryptographic services in Cryptographic Primitives Library.

- BCryptCloseAlgorithmProvider
- BCryptCreateHash
- BCryptCreateMultiHash
- BCryptDecrypt
- BCryptDeriveKey
- BCryptDeriveKeyPBKDF2
- BCryptDestroyHash
- BCryptDestroyKey
- BCryptDestroySecret
- BCryptDuplicateHash
- BCryptDuplicateKey
- BCryptEncrypt
- BCryptExportKey
- BCryptFinalizeKeyPair
- BCryptFinishHash
- BCryptFreeBuffer
- BCryptGenerateKeyPair
- BCryptGenerateSymmetricKey
- BCryptGenRandom
- BCryptGetProperty
- BCryptHash

- BCryptHashData
- BCryptImportKey
- BCryptImportKeyPair
- BCryptKeyDerivation
- BCryptOpenAlgorithmProvider
- BCryptProcessMultiOperations
- BCryptSecretAgreement
- BCryptSetProperty
- BCryptSignHash
- BCryptVerifySignature

All of these functions are used in the approved mode. Furthermore, these are the only approved functions that this module can perform.

Cryptographic Primitives Library has additional export functions described in Non-Security Relevant Configuration Interfaces.

3.2.1 Algorithm Providers and Properties

3.2.1.1 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 https://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.

3.2.1.2 BCryptCloseAlgorithmProvider

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

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

3.2.1.3 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.

3.2.1.4 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.

3.2.1.5 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.

3.2.2 Key and Key-Pair Generation

3.2.2.1 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 directly from a DRBG for use with a symmetric encryption algorithm or key derivation 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 or key derivation.

3.2.2.2 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.

Note: for when generating a key pair with “BCRYPT_DSA_ALGORITHM” If the key length is 1024 bits, then a process conformant with FIPS 186-2 DSA will be used to generate the key pair and perform subsequent DSA operations[13]. If the key length is 2048 or 3072 bits, then a process conformant with FIPS 186-4 DSA is used to generate the key pair and perform subsequent DSA operations.

3.2.2.3 BCryptFinalizeKeyPair

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

The BCryptFinalizeKeyPair() function completes a public/private key pair import or generation directly from the output of a DRBG. 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.

3.2.2.4 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.

3.2.2.5 BCryptDestroyKey

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

The BCryptDestroyKey() function destroys a key.

[13] 1024 bits is not an approved key length for DSA.

3.2.3 Random Number Generation

3.2.3.1 BCryptGenRandom

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

The BCryptGenRandom() function fills a buffer with random bytes. BCRYPTPRIMITVES.DLL implements the following random number generation algorithm:

- BCRYPT_RNG_ALGORITHM. This is the AES-256 counter mode based random generator as defined in SP 800-90A.

3.2.4 Key Entry and Output

3.2.4.1 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.

3.2.4.2 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.

3.2.4.3 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.

3.2.5 Encryption and Decryption

3.2.5.1 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.

3.2.5.2 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.

3.2.6 Hashing and Message Authentication

3.2.6.1 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, AES GMAC and AES CMAC.

3.2.6.2 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.

3.2.6.3 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.

3.2.6.4 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.

3.2.6.5 BCryptDestroyHash

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

The BCryptDestroyHash() function destroys a hash object.

3.2.6.6 BCryptHash

```
NTSTATUS WINAPI BCryptHash(
        BCRYPT_ALG_HANDLE hAlgorithm,
        PUCHAR pbSecret,
        ULONG cbSecret,
        PUCHAR pbInput,
        ULONG cbInput,
        PUCHAR pbOutput,
        ULONG  cbOutput);
```

The function BCryptHash() performs a single hash computation. This is a convenience function that wraps calls to the BCryptCreateHash(), BCryptHashData(), BCryptFinishHash(), and BCryptDestroyHash() functions.

3.2.6.7 BCryptCreateMultiHash

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

BCryptCreateMultiHash() is a function that creates a new MultiHash object that is used in parallel hashing to improve performance. The MultiHash object is equivalent to an array of normal (reusable) hash objects.

3.2.6.8 BCryptProcessMultiOperations

```
NTSTATUS WINAPI BCryptProcessMultiOperations(
        BCRYPT_HANDLE  hObject,
        BCRYPT_MULTI_OPERATION_TYPE operationType,
        PVOID pOperations,
        ULONG cbOperations,
        ULONG dwFlags );
```

The BCryptProcessMultiOperations() function is used to perform multiple operations on a single multi-object handle such as a MultiHash object handle. If any of the operations fail, then the function will return an error.

3.2.7 Signing and Verification

3.2.7.1 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.

Note: this function accepts SHA-1 hashes, which according to NIST SP 800-131A is *disallowed* for digital signature generation. SHA-1 is currently *legacy-use* for digital signature verification.

3.2.7.2 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.

Note: this function accepts SHA-1 hashes, which according to NIST SP 800-131A is *disallowed* for digital signature generation. SHA-1 is currently *legacy-use* for digital signature verification.

3.2.8 Secret Agreement and Key Derivation

3.2.8.1 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.

3.2.8.2 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.

3.2.8.3 BCryptDestroySecret

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

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

3.2.8.4 BCryptKeyDerivation

```
    NTSTATUS WINAPI BCryptKeyDerivation(
        _In_       BCRYPT_KEY_HANDLE hKey,
        _In_opt_   BCryptBufferDesc     *pParameterList,
        _Out_writes_bytes_to_(cbDerivedKey, *pcbResult) PUCHAR pbDerivedKey,
        _In_       ULONG             cbDerivedKey,
        _Out_      ULONG             *pcbResult,
        _In_       ULONG             dwFlags);
```

The BCryptKeyDerivation() function executes a Key Derivation Function (KDF) on a key generated with BCryptGenerateSymmetricKey() function. It differs from the BCryptDeriveKey() function in that it does not require a secret agreement step to create a shared secret.

3.2.8.5 BCryptDeriveKeyPBKDF2

```
    NTSTATUS WINAPI BCryptDeriveKeyPBKDF2(
        BCRYPT_ALG_HANDLE hPrf,
        PUCHAR pbPassword,
        ULONG cbPassword,
        PUCHAR pbSalt,
        ULONG cbSalt,
        ULONGLONG cIterations,
        PUCHAR pbDerivedKey,
        ULONG cbDerivedKey,
        ULONG dwFlags);
```

The BCryptDeriveKeyPBKDF2() function derives a key from a hash value by using the password based key derivation function as defined by NIST SP 800-132 PBKDF and IETF RFC 2898 (specified as PBKDF2).

3.2.9 Cryptographic Transitions

3.2.9.1 Bit Strengths of DH and ECDH

Through the year 2010, implementations of DH and ECDH were allowed to have an acceptable bit strength of at least 80 bits of security (for DH at least 1024 bits and for ECDH at least 160 bits). From 2011 through 2013, 80 bits of security strength was considered deprecated, and was disallowed starting January 1, 2014. As of that date, only security strength of at least 112 bits is acceptable. ECDH uses curve sizes of at least 256 bits (that means it has at least 128 bits of security strength), so that is acceptable. However, DH has a range of 1024 to 4096 and that changed to 2048 to 4096 after 2013.

3.2.9.2 SHA-1

From 2011 through 2013, SHA-1 could be used in a deprecated mode for use in digital signature generation. As of Jan. 1, 2014, SHA-1 is no longer allowed for digital signature generation, and it is allowed for legacy use only for digital signature verification.

3.3 Control Input Interface

The Control Input Interface are the functions in Algorithm Providers and Properties. Options for control operations are passed as input parameters to these functions.

3.4 Status Output Interface

The Status Output Interface for Cryptographic Primitives Library consists of the CNG primitive functions listed in CNG Primitive Functions. For each function, the status information is returned to the caller as the return value from the function.

3.5 Data Output Interface

The Data Output Interface for Cryptographic Primitives Library consists of the Cryptographic Primitives Library export functions except for the Control Input Interfaces. Data is returned to the function's caller via output parameters.

3.6 Data Input Interface

The Data Input Interface for Cryptographic Primitives Library consists of the Cryptographic Primitives Library export functions except for the Control Input Interfaces. Data and options are passed to the interface as input parameters to the export functions. Data Input is kept separate from Control Input by passing Data Input in separate parameters from Control Input.

3.7 Non-Security Relevant Configuration Interfaces

These non-cryptographic functions are used to configure cryptographic providers on the system. Note that these functions are interfaces exported by the module, but are implemented in CNG.SYS. See the the Cryptographic Primitives Library Security Policy Document for details on the services provided by these functions.

Function Name	Description
BCryptEnumAlgorithms	Enumerates the algorithms for a given set of operations.
BCryptEnumProviders	Returns a list of CNG providers for a given algorithm.
BCryptRegisterConfigChangeNotify	This is deprecated beginning with Windows 10.
BCryptResolveProviders	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.
BCryptAddContextFunctionProvider	Adds a cryptographic function provider to the list of providers that are supported by an existing CNG context.
BCryptRegisterProvider	Registers a CNG provider.
BCryptUnregisterProvider	Unregisters a CNG provider.
BCryptUnregisterConfigChangeNotify	Removes a CNG configuration change event handler.
BCryptGetFipsAlgorithmMode	Determines whether Cryptographic Primitives Library is operating in FIPS mode. Some applications use the value returned by this API to alter their own behavior, such as blocking the use of some SSL versions.
BCryptQueryProviderRegistration	Retrieves information about a CNG provider.
BCryptEnumRegisteredProviders	Retrieves information about the registered providers.
BCryptCreateContext	Creates a new CNG configuration context.
BCryptDeleteContext	Deletes an existing CNG configuration context.
BCryptEnumContexts	Obtains the identifiers of the contexts in the specified configuration table.
BCryptConfigureContext	Sets the configuration information for an existing CNG context.
BCryptQueryContextConfiguration	Retrieves the current configuration for the specified CNG context.
BCryptAddContextFunction	Adds a cryptographic function to the list of functions that are supported by an existing CNG context.
BCryptRemoveContextFunction	Removes a cryptographic function from the list of functions that are supported by an existing CNG context.
BCryptEnumContextFunctions	Obtains the cryptographic functions for a context in the specified configuration table.
BCryptConfigureContextFunction	Sets the configuration information for the cryptographic function of an existing CNG context.
BCryptQueryContextFunctionConfiguration	Obtains the cryptographic function configuration information for an existing CNG context.
BCryptEnumContextFunctionProviders	Obtains the providers for the cryptographic functions for a context in the specified configuration table.
BCryptSetContextFunctionProperty	Sets the value of a named property or a cryptographic function in an existing CNG context.

BCryptQueryContextFunctionProperty	Obtains the value of a named property for a cryptographic function in an existing CNG context.
BCryptSetAuditingInterface	Sets the auditing interface.

4 Roles, Services and Authentication

4.1 Roles

When an application requests the cryptographic 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. FIPS 140 validations define formal "User" and "Cryptographic Officer" roles. Both roles can use any of this module's services.

4.2 Services

Cryptographic Primitives Library services are described below.

1. **Algorithm Providers and Properties** – This module provides interfaces to register algorithm providers
2. **Random Number Generation**
3. **Key and Key-Pair Generation**
4. **Key Entry and Output**
5. **Encryption and Decryption**
6. **Hashing and Message Authentication**
7. **Signing and Verification**
8. **Secret Agreement and Key Derivation**
9. **Show Status** – The module provides a show status service that is automatically executed by the module to provide the status response of the module either via output to the computer monitor or to log files.
10. **Self-Tests** - The module provides a power-up self-tests service that is automatically executed when the module is loaded into memory.
11. **Zeroizing Cryptographic Material -** This service is executed as part of the module shutdown. See Cryptographic Key Management

4.2.1 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
Algorithm Providers and Properties	None	None
Random Number Generation	AES-256 CTR 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	SP 800-38F AES Key Wrapping (128, 192, and 256)	Symmetric Keys Asymmetric Public Keys Asymmetric Private Keys
Encryption and Decryption	• Triple-DES with 2 key (encryption disallowed) and 3 key in ECB, CBC, CFB8 and CFB64 modes; • AES-128, AES-192, and AES-256 in ECB, CBC, CFB8, CFB128, and CTR modes; • AES-128, AES-192, and AES-256 in CCM, CMAC, and GMAC modes; • AES-128, AES-192, and AES-256 GCM decryption; • XTS-AES XTS-128 and XTS-256; • SP 800-56B RSADP mod 2048 (IEEE 1619-2007 XTS-AES, AES GCM encryption, 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 SHS SHA-1, SHA-256, SHA-384, and SHA-512; • FIPS 180-4 SHA-1, SHA-256, SHA-384, SHA-512 HMAC; • AES-128, AES-192, and AES-256 in CCM, CMAC, and GMAC; • MD5 and HMAC-MD5 (allowed in TLS and EAP-TLS); • MD2 and MD4 (disallowed in FIPS mode)	Symmetric Keys (for HMAC, AES CCM, AES CMAC, and AES GMAC)
Signing and Verification	• FIPS 186-4 RSA (RSASSA-PKCS1-v1_5 and RSASSA-PSS) digital signature generation and verification with 2048 and 3072 modulus;	Asymmetric Public Keys Asymmetric RSA Private Keys Asymmetric ECDSA Public Keys Asymmetric ECDSA Private keys

	supporting SHA-1[14], SHA-256, SHA-384, and SHA-512 • FIPS 186-4 ECDSA with the following NIST curves: P-256, P-384, P-521	
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 with the following NIST curves: P-256, P-384, P-521 and the FIPS non-Approved curves listed in Non-Approved Algorithms • SP 800-108 Key Derivation Function (KDF) CMAC-AES (128, 192, 256), HMAC (SHA1, SHA-256, SHA-384, SHA-512) • SP 800-132 PBKDF • Legacy CAPI KDF (cannot be used in FIPS mode) • HKDF (cannot be used in FIPS mode)	DH Private and Public Values ECDH Private and Public Values
Show Status	None	None
Self-Tests	See Section Self-Tests for the list of algorithms	None
Zeroizing Cryptographic Material	None	None

4.2.2 Mapping of Services, Export Functions, and Invocations

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

Service	Export Functions	Invocations
Algorithm Providers and Properties	BCryptOpenAlgorithmProvider BCryptCloseAlgorithmProvider BCryptSetProperty BCryptGetProperty BCryptFreeBuffer	This service is executed whenever one of these exported functions is called.
Random Number Generation	BcryptGenRandom	This service is executed whenever one of these exported functions is called.

[14] SHA-1 is only acceptable for signature verification.

Key and Key-Pair Generation	BCryptGenerateSymmetricKey BCryptGenerateKeyPair BCryptFinalizeKeyPair BCryptDuplicateKey BCryptDestroyKey	This service is executed whenever one of these exported functions is called.
Key Entry and Output	BCryptImportKey BCryptImportKeyPair BCryptExportKey	This service is executed whenever one of these exported functions is called.
Encryption and Decryption	BCryptEncrypt BCryptDecrypt	This service is executed whenever one of these exported functions is called.
Hashing and Message Authentication	BCryptCreateHash BCryptHashData BCryptDuplicateHash BCryptFinishHash BCryptDestroyHash BCryptHash BCryptCreateMultiHash BCryptProcessMultiOperations	This service is executed whenever one of these exported functions is called.
Signing and Verification	BCryptSignHash BCryptVerifySignature	This service is executed whenever one of these exported functions is called.
Secret Agreement and Key Derivation	BCryptSecretAgreement BCryptDeriveKey BCryptDestroySecret BCryptKeyDerivation BCryptDeriveKeyPBKDF2	This service is executed whenever one of these exported functions is called.
Show Status	All Exported Functions	This service is executed upon completion of an exported function.
Self-Tests	DllMain	This service is executed upon startup of this module.
Zeroizing Cryptographic Material	BCryptDestroyKey BCryptDestroySecret	This service is executed whenever one of these exported functions is called.

4.2.3 Non-Approved Services

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

Function Name	Description
BCryptDeriveKeyCapi	Derives a key from a hash value. This function is provided as a helper function to assist in migrating from legacy Cryptography API (CAPI) to CNG.
BCRYPT_KDF_HKDF	Derives a key from a hash value. This function is provided to support potential enhancements to Windows.

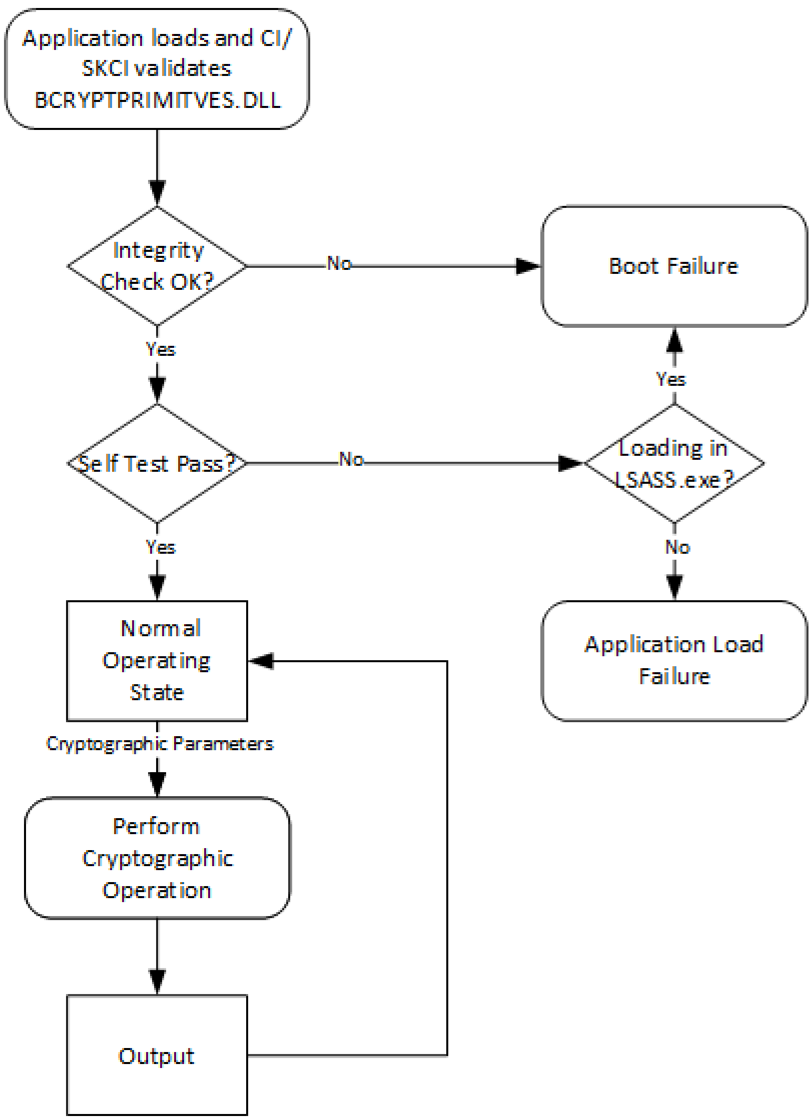
4.3 Authentication

Cryptographic Primitives Library does not provide authentication of users. Roles are implicitly assumed based on the services that are executed.

5 Finite State Model

5.1 Specification

The following diagram shows the finite state model for Cryptographic Primitives:



6 Operational Environment

The operational environment for Cryptographic Primitives Library is the Windows 10 operating system running on a supported hardware platform.

6.1 Single Operator

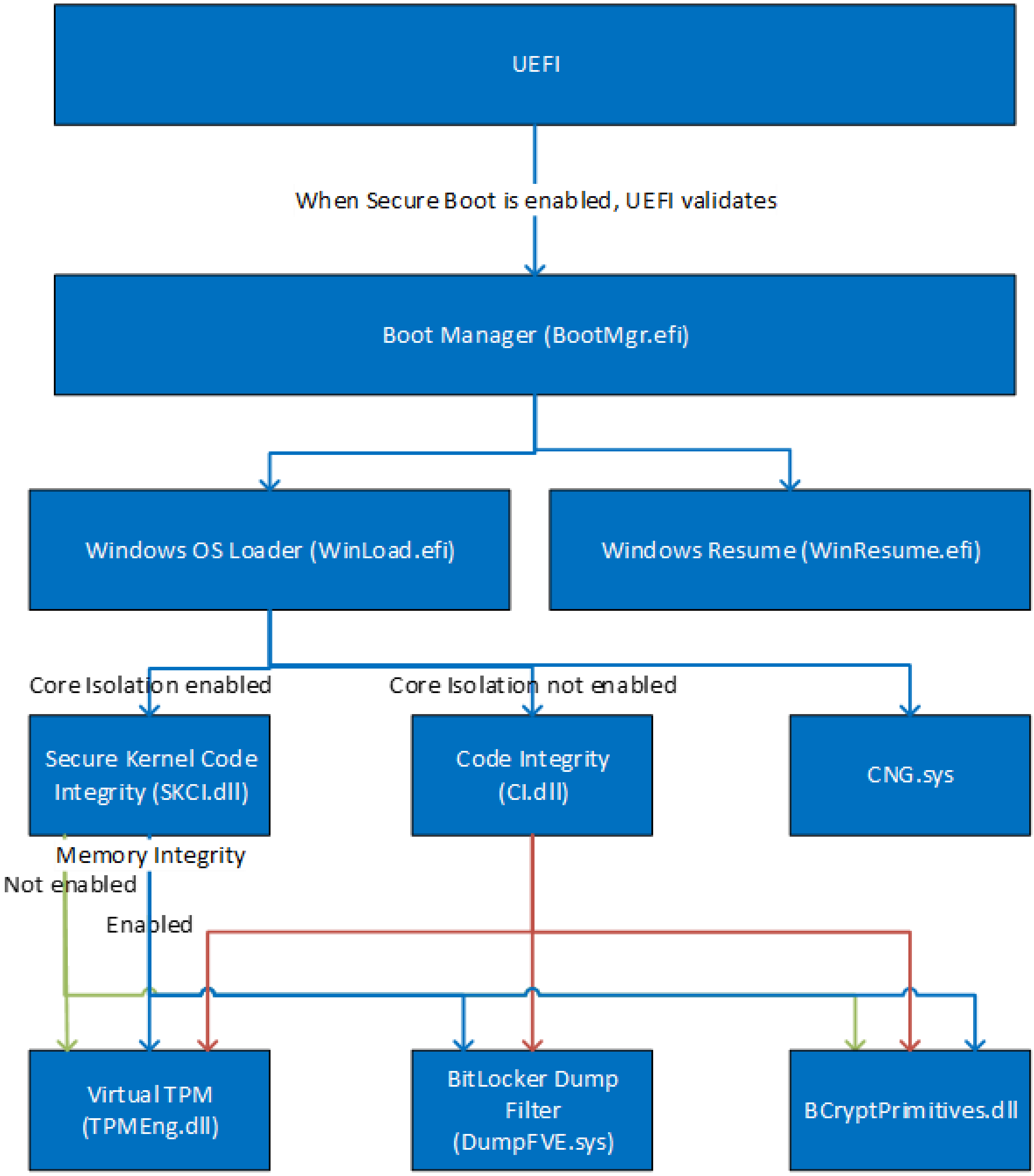
The for Cryptographic Primitives Library is loaded into process memory for a single application. The "single operator" for the module is the identity associated with the parent process.

6.2 Cryptographic Isolation

Windows dynamic link libraries, which includes BCRYPTPRIMITIVES.DLL, are loaded into a user-mode process to expose the services offered by that DLL. The operating system environment enforces process isolation including memory (where keys and intermediate key data are stored) and CPU scheduling.

6.3 Integrity Chain of Trust

Windows uses several mechanisms to provide integrity verification depending on the stage in the boot sequence and also on the hardware and configuration. The following diagram describes the integrity chain of trust for each supported configuration:



The integrity of Cryptographic Primitives Library is checked by Code Integrity or Secure Kernel Code Integrity before it is loaded into process memory.

Windows binaries include a SHA-256 hash of the binary signed with the 2048 bit Microsoft RSA code-signing key (i.e., the key associated with the Microsoft code-signing certificate). The integrity check uses the public key component of the Microsoft code signing certificate to verify the signed hash of the binary.

7 Cryptographic Key Management

The Cryptographic Primitives Library crypto module uses the following critical security parameters (CSPs) for FIPS Approved security functions:

Security Relevant Data Item	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 192 and 128 bits.
HMAC keys	Keys used for HMAC-SHA1, HMAC-SHA256, HMAC-SHA384, and HMAC-SHA512
Asymmetric DSA Public Keys	Keys used for the verification of DSA digital signatures. Key sizes are 2048 and 3072 bits.
Asymmetric DSA Private Keys	Keys used for the calculation of DSA digital signatures. Key sizes are 2048 and 3072 bits.
Asymmetric ECDSA Public Keys	Keys used for the verification of ECDSA digital signatures. Curve sizes are P-256, P-384, and P-521.
Asymmetric ECDSA Private Keys	Keys used for the calculation of ECDSA digital signatures. Curve sizes are P-256, P-384, and P-521.
Asymmetric RSA Public Keys	Keys used for the verification of RSA digital signatures. Key sizes are 2048 and 3072 bits. These keys can be produced using RSA Key Generation.
Asymmetric RSA Private Keys	Keys used for the calculation of RSA digital signatures. Key sizes are 2048 and 3072 bits. These keys can be produced using RSA Key Generation.
AES-CTR DRBG Entropy Input	A secret value that is at least 256 bits and maintained internal to the module that provides the entropy material for AES-CTR DRBG output[15]
AES-CTR DRBG Seed	A 384 bit secret value maintained internal to the module that provides the seed material for AES-CTR DRBG output[16]
AES-CTR DRBG V	A 128 bit secret value maintained internal to the module that provides the entropy material for AES-CTR DRBG output[17]

[15] Microsoft Common Criteria Windows Security Target, Page 29.
[16] Recommendation for Random Number Generation Using Deterministic Random Bit Generators, NIST SP 800-90A Revision 1, page 49.
[17] Ibid.

AES-CTR DRBG key	A 256 bit secret value maintained internal to the module that provides the entropy material for AES-CTR DRBG output[18]
DH Private and Public values	Private and public values used for Diffie-Hellman key establishment. Key sizes are 2048 to 4096 bits.
ECDH Private and Public values	Private and public values used for EC Diffie-Hellman key establishment. Curve sizes are P-256, P-384, and P-521 and the ones listed in section 0.
IKEv1 and IKEv2 DH shared secrets	Diffie-Hellman shared secret lengths are 2048 (SHA 256), 256 (SHA 256), and 384 (SHA 384).
TLS Derived Key	A key derived for the TLS protocol. Key size is 384 bits.

7.1 Access Control Policy

The Cryptographic Primitives Library cryptographic module allows controlled access to security relevant data items contained within it. The following table defines the access that a service has to each. The permissions are categorized as a set of four separate permissions: read (r), write (w), execute (x), delete (d). If no permission is listed, the service has no access to the item.

Cryptographic Primitives Library crypto module **Service Access Policy**	Symmetric encryption and decryption keys	HMAC keys	Asymmetric DSA Public Keys	Asymmetric DSA Private Keys	Asymmetric ECDSA Public keys	Asymmetric ECDSA Private keys	Asymmetric RSA Public Keys	Asymmetric RSA Private Keys	DH Public and Private values	ECDH Public and Private values	AES-CTR DRBG Seed, AES-CTR DRBG Entropy Input, AES-CTR DRBG V, & AES-CTR DRBG key	IKEv1 & IKEv2 DH Shared Secret	TLS Derived Key
Algorithm Providers and Properties													
Random Number Generation											x		
Key and Key-Pair Generation	w d	w d	w d	w d	w d	w d	w d	w d	w d	w d	x		
Key Entry and Output	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw			
Encryption and Decryption	x												
Hashing and Message Authentication		xw											
Signing and Verification			x	x	x	x	x	x			x		
Secret Agreement and Key Derivation									x	x	x	x	x
Show Status													
Self-Tests													

[18] Ibid.

Zeroizing Cryptographic Material	w d	w d	w d	w d	w d	w d	w d	w d	w d	w d	wd	w d	w d

7.2 Key Material

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

7.3 Key Generation

Cryptographic Primitives Library can create and use keys for the following algorithms: RSA, DSA, DH, ECDH, ECDSA, RC2, RC4, DES, Triple-DES, AES, and HMAC. However, RC2, RC4, and DES cannot 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, AES, RSA, ECDSA, DSA, DH, and ECDH keys and key-pairs are generated following the techniques given in SP 800-56Ar2 (Section 5.8.1).

Keys generated while not operating in the FIPS mode of operation (as described in section 2) cannot be used in FIPS mode, and vice versa.

7.4 Key Establishment

Cryptographic Primitives Library 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. Alternatively, the module can also use Approved KDFs to derive key material from a specified secret value or password.

Cryptographic Primitives Library 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 SP 800-56A (Section 5.8), X9.63, and X9.42 key derivation.
- BCRYPT_KDF_HMAC. This KDF supports the IPsec IKEv1 key derivation that is non-Approved but is an allowed legacy implementation in FIPS mode when used to establish keys for IKEv1 as per scenario 4 of IG D.8.
- BCRYPT_KDF_TLS_PRF. This KDF supports the SSLv3.1 and TLSv1.0 key derivation that is non-Approved but is an allowed legacy implementation in FIPS mode when used to establish keys for SSLv3.1 or TLSv1.0 as specified in as per scenario 4 of IG D.8.

Cryptographic Primitives Library can use the following FIPS approved key derivation functions (KDF) from a key handle created from a specified secret or password:

- BCRYPT_SP800108_CTR_HMAC_ALGORITHM. This KDF supports the counter-mode variant of the KDF specified in SP 800-108 (Section 5.1) with HMAC as the underlying PRF.
- BCRYPT_SP80056A_CONCAT_ALGORITHM. This KDF supports the Concatenation KDF as specified in SP 800-56Ar2 (Section 5.8.1).
- BCRYPT_PBKDF2_ALGORITHM. This KDF supports the Password Based Key Derivation Function specified in SP 800-132 (Section 5.3).

In addition, the industry standard KDF, HKDF (CNG flag BCRYPT_KDF_HKDF), and the legacy proprietary CryptDerive Key KDF, (BCRYPT_CAPI_KDF_ALGORITHM, described at https://msdn.microsoft.com/library/windows/desktop/aa379916.aspx). cannot be used in a FIPS approved mode.

7.4.1 NIST SP 800-132 Password Based Key Derivation Function (PBKDF)

There are two options presented in NIST SP 800-132, pages 8 – 10, that are used to derive the Data Protection Key (DPK) from the Master Key. With the Cryptographic Primitives Library, it is up to the caller to select the option to generate/protect the DPK. For example, DPAPI uses option 2a. Cryptographic Primitives Library provides all the building blocks for the caller to select the desired option.

The Cryptographic Primitives Library supports the following HMAC hash functions as parameters for PBKDF:

- SHA-1 HMAC
- SHA-256 HMAC
- SHA-384 HMAC
- SHA-512 HMAC

Keys derived from passwords, as described in SP 800-132, may only be used for storage applications. In order to run in a FIPS Approved manner, strong passwords must be used and they may only be used for storage applications. The password/passphrase length is enforced by the caller of the PBKDF interfaces when the password/passphrase is created and not by this cryptographic module.[19]

7.4.2 NIST SP 800-38F AES Key Wrapping

As outlined in FIPS 140-2 IG, D.2 and D.9, AES key wrapping serves as a form of key transport, which in turn is a form of key establishment. This implementation of AES key wrapping is in accordance with NIST SP 800-38F Recommendation for Block Cipher Modes of Operation: Methods for Key Wrapping.

7.5 Key Entry and Output

[19] The probability of guessing a password is determined by its length and complexity, an organization should define a policy for these based based their threat model, suh as the example guidance in NIST SP800-63b, Appendix A.

Keys can be both exported and imported out of and into Cryptographic Primitives Library 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.

7.6 Key Storage

Cryptographic Primitives Library does not provide persistent storage of keys.

7.7 Key Archival

Cryptographic Primitives Library 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.

7.8 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

8.1 Power-On Self-Tests

The Cryptographic Primitives Library module implements Known Answer Test (KAT) functions each time the module is loaded into a process and the default DLL entry point, DllMain is called.

Cryptographic Primitives Library performs the following power-on (startup) self-tests:

- HMAC (SHA-1, SHA-256, and SHA-512) Known Answer Tests
- Triple-DES encrypt/decrypt ECB Known Answer Tests
- AES-128 encrypt/decrypt ECB Known Answer Tests
- AES-128 encrypt/decrypt CCM Known Answer Tests
- AES-128 encrypt/decrypt CBC Known Answer Tests
- AES-128 CMAC Known Answer Test
- AES-128 encrypt/decrypt GCM Known Answer Tests
- XTS-AES encrypt/decrypt Known Answer Tests
- RSA sign/verify Known Answer Tests using RSA_SHA256_PKCS1 signature generation and verification
- DSA sign/verify tests with 2048-bit key
- ECDSA sign/verify Known Answer Tests on P256 curve
- DH secret agreement Known Answer Test with 2048-bit key
- 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)
- SP 800-108 KDF Known Answer Test
- SP 800-132 PBKDF Known Answer Test
- SHA-256 Known Answer Test
- SHA-512 Known Answer Test
- SP800-135 TLS 1.0/1.1 KDF Known Answer Test
- SP800-135 TLS 1.2 KDF Known Answer Test
- IKE SP800_135 KDF Known Answer Test

If any self-test fails, Cryptographic Primitives Library DllMain returns an error code. The caller may attempt to reload the Cryptographic Primitives Library.

8.2 Conditional Self-Tests

Cryptographic Primitives Library performs the following conditional self-tests on key generation and import:

- Pairwise consistency tests for DSA, ECDSA, and RSA keys
- DH and ECDH assurances (including pairwise consistency tests) according to NIST SP 800-56A

A Continuous Random Number Generator Test (CRNGT) and the DRBG health tests are performed for SP 800-90A AES-256 CTR DRBG.

When BCRYPT_ENABLE_INCOMPATIBLE_FIPS_CHECKS flag (required by policy) is used with BCryptGenerateSymmetricKey, then the XTS-AES Key_1 ≠ Key_2 check is performed in compliance with FIPS 140-2 IG A.9.

If the conditional self-test fails, the module will not load and a status code other than STATUS_SUCCESS will be returned.

9 Design Assurance

The secure installation, generation, and startup procedures of this cryptographic module are part of the overall operating system secure installation, configuration, and startup procedures for the Windows 10 operating system.

The Windows 10 operating system must be pre-installed on a computer by an OEM, installed by the end-user, by an organization's IT administrator, or updated from a previous Windows 10 version downloaded from Windows Update.

An inspection of authenticity of the physical medium can be made by following the guidance at this Microsoft web site: https://www.microsoft.com/en-us/howtotell/default.aspx

The installed version of Windows 10 must be verified to match the version that was validated. See Appendix A – How to Verify Windows Versions and Digital Signatures for details on how to do this.

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

10 Mitigation of Other Attacks

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

Algorithm	Protected Against	Mitigation
SHA1	Timing Analysis Attack	Constant time implementation
	Cache Attack	Memory access pattern is independent of any confidential data
SHA2	Timing Analysis Attack	Constant time implementation
	Cache Attack	Memory access pattern is independent of any confidential data
Triple-DES	Timing Analysis Attack	Constant time implementation
AES	Timing Analysis Attack	Constant time implementation
	Cache Attack	Memory access pattern is independent of any confidential data Protected against cache attacks only when used with AES NI

11 Security Levels

The security level for each FIPS 140-2 security requirement is given in the following table.

Security Requirement	Security Level
Cryptographic Module Specification	1
Cryptographic Module Ports and Interfaces	1
Roles, Services, and Authentication	1
Finite State Model	1
Physical Security	NA
Operational Environment	1

Cryptographic Key Management	1
EMI/EMC	1
Self-Tests	1
Design Assurance	2
Mitigation of Other Attacks	1

12 Additional Details

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

https://www.microsoft.com/en-us/windows

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

https://docs.microsoft.com/en-us/windows/security/threat-protection/fips-140-validation

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

13.1 How to Verify Windows Versions

The installed version of Windows 10 must be verified to match the version that was validated using the following method:

1. In the Search box type "cmd" and open the Command Prompt desktop app.
2. The command window will open.
3. At the prompt, enter "ver”.
4. The version information will be displayed in a format like this:
   `Microsoft Windows [Version 10.0.xxxxx]`

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

13.2 How to Verify Windows Digital Signatures

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

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

14 Appendix B – References

This table lists the specifications for each elliptic curve in section Non-Approved Algorithms

Curve	Specification
Curve25519	https://cr.yp.to/ecdh/curve25519-20060209.pdf
brainpoolP160r1	http://www.ecc-brainpool.org/download/Domain-parameters.pdf
brainpoolP192r1	http://www.ecc-brainpool.org/download/Domain-parameters.pdf
brainpoolP192t1	http://www.ecc-brainpool.org/download/Domain-parameters.pdf
brainpoolP224r1	http://www.ecc-brainpool.org/download/Domain-parameters.pdf
brainpoolP224t1	http://www.ecc-brainpool.org/download/Domain-parameters.pdf
brainpoolP256r1	http://www.ecc-brainpool.org/download/Domain-parameters.pdf
brainpoolP256t1	http://www.ecc-brainpool.org/download/Domain-parameters.pdf
brainpoolP320r1	http://www.ecc-brainpool.org/download/Domain-parameters.pdf
brainpoolP320t1	http://www.ecc-brainpool.org/download/Domain-parameters.pdf
brainpoolP384r1	http://www.ecc-brainpool.org/download/Domain-parameters.pdf
brainpoolP384t1	http://www.ecc-brainpool.org/download/Domain-parameters.pdf
brainpoolP512r1	http://www.ecc-brainpool.org/download/Domain-parameters.pdf
brainpoolP512t1	http://www.ecc-brainpool.org/download/Domain-parameters.pdf
ec192wapi	http://www.gbstandards.org/GB_standards/GB_standard.asp?id=900 (The GB standard is available here for purchase)
nistP192	http://csrc.nist.gov/groups/ST/toolkit/documents/dss/NISTReCur.pdf
nistP224	http://csrc.nist.gov/groups/ST/toolkit/documents/dss/NISTReCur.pdf
numsP256t1	https://www.microsoft.com/en-us/research/wp-content/uploads/2016/02/curvegen.pdf
numsP384t1	https://www.microsoft.com/en-us/research/wp-content/uploads/2016/02/curvegen.pdf
numsP512t1	https://www.microsoft.com/en-us/research/wp-content/uploads/2016/02/curvegen.pdf
secP160k1	http://www.secg.org/sec2-v2.pdf
secP160r1	http://www.secg.org/sec2-v2.pdf
secP160r2	http://www.secg.org/sec2-v2.pdf
secP192k1	http://www.secg.org/sec2-v2.pdf
secP192r1	http://www.secg.org/sec2-v2.pdf
secP224k1	http://www.secg.org/sec2-v2.pdf
secP224r1	http://www.secg.org/sec2-v2.pdf
secP256k1	http://www.secg.org/sec2-v2.pdf
secP256r1	http://www.secg.org/sec2-v2.pdf
secP384r1	http://www.secg.org/sec2-v2.pdf
secP521r1	http://www.secg.org/sec2-v2.pdf
wtls12	http://www.openmobilealliance.org/tech/affiliates/wap/wap-261-wtls-20010406-a.pdf
wtls7	http://www.openmobilealliance.org/tech/affiliates/wap/wap-261-wtls-20010406-a.pdf

Curve	Specification
wtls9	http://www.openmobilealliance.org/tech/affiliates/wap/wap-261-wtls-20010406-a.pdf
x962P192v1	https://global.ihs.com/doc_detail.cfm?&item_s_key=00325725&item_key_date=941231&input_doc_number=ANSI%20X9%2E62&input_doc_title= (The ANSI X9.62 standard is available here for purchase)
x962P192v2	https://global.ihs.com/doc_detail.cfm?&item_s_key=00325725&item_key_date=941231&input_doc_number=ANSI%20X9%2E62&input_doc_title= (The ANSI X9.62 standard is available here for purchase)
x962P192v3	https://global.ihs.com/doc_detail.cfm?&item_s_key=00325725&item_key_date=941231&input_doc_number=ANSI%20X9%2E62&input_doc_title= (The ANSI X9.62 standard is available here for purchase)
x962P239v1	https://global.ihs.com/doc_detail.cfm?&item_s_key=00325725&item_key_date=941231&input_doc_number=ANSI%20X9%2E62&input_doc_title= (The ANSI X9.62 standard is available here for purchase)
x962P239v2	https://global.ihs.com/doc_detail.cfm?&item_s_key=00325725&item_key_date=941231&input_doc_number=ANSI%20X9%2E62&input_doc_title= (The ANSI X9.62 standard is available here for purchase)
x962P239v3	https://global.ihs.com/doc_detail.cfm?&item_s_key=00325725&item_key_date=941231&input_doc_number=ANSI%20X9%2E62&input_doc_title= (The ANSI X9.62 standard is available here for purchase)
x962P256v1	https://global.ihs.com/doc_detail.cfm?&item_s_key=00325725&item_key_date=941231&input_doc_number=ANSI%20X9%2E62&input_doc_title= (The ANSI X9.62 standard is available here for purchase)