



# Microsoft Windows

# FIPS 140 Validation

Microsoft Windows Server 2019

Microsoft Azure Stack Edge

Microsoft Azure Stack Hub

Microsoft Azure Stack Edge Rugged

*Non-Proprietary*

## Security Policy Document

Version Number	1.5
Updated On	January 23, 2024

*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.*

*© 2024 Microsoft Corporation. All rights reserved.*

*Microsoft, Windows, the Windows logo, Windows Server, and BitLocker 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.*

**Version History**

Version	Date	Summary of changes
<b>1.0</b>	November 4, 2020	Draft sent to NIST CMVP
<b>1.1</b>	November 3, 2022	Updates in response to NIST feedback
<b>1.2</b>	January 13, 2023	Updates in response to NIST feedback
<b>1.3</b>	November 6, 2023	Updates in response to NIST feedback
<b>1.4</b>	December 12, 2023	Updates to bounded module certificates
<b>1.5</b>	January 23, 2024	Updates in response to NIST feedback

## TABLE OF CONTENTS

<b><u>SECURITY POLICY DOCUMENT</u></b> .....	<b><u>1</u></b>
<b><u>VERSION HISTORY</u></b> .....	<b><u>3</u></b>
<b><u>1 INTRODUCTION</u></b> .....	<b><u>8</u></b>
1.1 LIST OF CRYPTOGRAPHIC MODULE BINARY EXECUTABLES .....	9
1.2 VALIDATED PLATFORMS .....	9
1.3 CONFIGURE WINDOWS TO USE FIPS-APPROVED CRYPTOGRAPHIC ALGORITHMS .....	10
<b><u>2 CRYPTOGRAPHIC MODULE SPECIFICATION</u></b> .....	<b><u>10</u></b>
2.1 CRYPTOGRAPHIC BOUNDARY .....	11
2.2 FIPS 140-2 APPROVED ALGORITHMS .....	11
2.3 NON-APPROVED ALGORITHMS .....	13
2.4 FIPS 140-2 APPROVED ALGORITHMS FROM BOUNDED MODULES .....	15
2.5 CRYPTOGRAPHIC BYPASS .....	15
2.6 HARDWARE COMPONENTS OF THE CRYPTOGRAPHIC MODULE .....	15
<b><u>3 CRYPTOGRAPHIC MODULE PORTS AND INTERFACES</u></b> .....	<b><u>16</u></b>
3.1 EXPORT FUNCTIONS .....	16
3.2 CNG PRIMITIVE FUNCTIONS .....	17
3.2.1 ALGORITHM PROVIDERS AND PROPERTIES .....	18
3.2.1.1 BCryptOpenAlgorithmProvider .....	18
3.2.1.2 BCryptCloseAlgorithmProvider .....	18
3.2.1.3 BCryptSetProperty .....	18
3.2.1.4 BCryptGetProperty .....	18
3.2.1.5 BCryptFreeBuffer .....	19
3.2.2 KEY AND KEY-PAIR GENERATION .....	19
3.2.2.1 BCryptGenerateSymmetricKey .....	19
3.2.2.2 BCryptGenerateKeyPair .....	19
3.2.2.3 BCryptFinalizeKeyPair .....	20
3.2.2.4 BCryptDuplicateKey .....	20
3.2.2.5 BCryptDestroyKey .....	20
3.2.3 RANDOM NUMBER GENERATION .....	20
3.2.3.1 BCryptGenRandom .....	20
3.2.4 KEY ENTRY AND OUTPUT .....	21

3.2.4.1	BCryptImportKey.....	21
3.2.4.2	BCryptImportKeyPair .....	21
3.2.4.3	BCryptExportKey .....	21
3.2.5	ENCRYPTION AND DECRYPTION .....	21
3.2.5.1	BCryptEncrypt .....	21
3.2.5.2	BCryptDecrypt.....	22
3.2.6	HASHING AND MESSAGE AUTHENTICATION .....	22
3.2.6.1	BCryptCreateHash.....	22
3.2.6.2	BCryptHashData.....	22
3.2.6.3	BCryptDuplicateHash .....	23
3.2.6.4	BCryptFinishHash .....	23
3.2.6.5	BCryptDestroyHash.....	23
3.2.6.6	BCryptHash.....	23
3.2.6.7	BCryptCreateMultiHash .....	23
3.2.6.8	BCryptProcessMultiOperations.....	24
3.2.7	SIGNING AND VERIFICATION .....	24
3.2.7.1	BCryptSignHash.....	24
3.2.7.2	BCryptVerifySignature.....	24
3.2.8	SECRET AGREEMENT AND KEY DERIVATION.....	25
3.2.8.1	BCryptSecretAgreement .....	25
3.2.8.2	BCryptDeriveKey .....	25
3.2.8.3	BCryptDestroySecret.....	25
3.2.8.4	BCryptKeyDerivation.....	25
3.2.8.5	BCryptDeriveKeyPBKDF2.....	26
3.2.9	CRYPTOGRAPHIC TRANSITIONS.....	26
3.2.9.1	Bit Strengths of DH and ECDH.....	26
3.2.9.2	SHA-1.....	26
<b>3.3</b>	<b>CONTROL INPUT INTERFACE .....</b>	<b>26</b>
<b>3.4</b>	<b>STATUS OUTPUT INTERFACE .....</b>	<b>27</b>
<b>3.5</b>	<b>DATA OUTPUT INTERFACE .....</b>	<b>27</b>
<b>3.6</b>	<b>DATA INPUT INTERFACE .....</b>	<b>27</b>
<b>3.7</b>	<b>NON-SECURITY RELEVANT CONFIGURATION INTERFACES.....</b>	<b>27</b>
<b>4</b>	<b><u>ROLES, SERVICES AND AUTHENTICATION .....</u></b>	<b><u>28</u></b>
<b>4.1</b>	<b>ROLES.....</b>	<b>28</b>
<b>4.2</b>	<b>SERVICES .....</b>	<b>28</b>
4.2.1	MAPPING OF SERVICES, ALGORITHMS, AND CRITICAL SECURITY PARAMETERS .....	29
4.2.2	MAPPING OF SERVICES, EXPORT FUNCTIONS, AND INVOCATIONS .....	31
4.2.3	NON-APPROVED SERVICES.....	32
<b>4.3</b>	<b>AUTHENTICATION .....</b>	<b>32</b>

<b><u>5</u></b>	<b><u>FINITE STATE MODEL.....</u></b>	<b><u>32</u></b>
<b>5.1</b>	<b>SPECIFICATION.....</b>	<b>32</b>
<b><u>6</u></b>	<b><u>OPERATIONAL ENVIRONMENT.....</u></b>	<b><u>33</u></b>
<b>6.1</b>	<b>SINGLE OPERATOR.....</b>	<b>33</b>
<b>6.2</b>	<b>CRYPTOGRAPHIC ISOLATION.....</b>	<b>33</b>
<b>6.3</b>	<b>INTEGRITY CHAIN OF TRUST .....</b>	<b>34</b>
<b><u>7</u></b>	<b><u>CRYPTOGRAPHIC KEY MANAGEMENT .....</u></b>	<b><u>36</u></b>
<b>7.1</b>	<b>ACCESS CONTROL POLICY.....</b>	<b>37</b>
<b>7.2</b>	<b>KEY MATERIAL.....</b>	<b>38</b>
<b>7.3</b>	<b>KEY GENERATION .....</b>	<b>38</b>
<b>7.4</b>	<b>KEY ESTABLISHMENT .....</b>	<b>38</b>
<b>7.4.1</b>	<b>NIST SP 800-132 PASSWORD BASED KEY DERIVATION FUNCTION (PBKDF) .....</b>	<b>39</b>
<b>7.4.2</b>	<b>NIST SP 800-38F AES KEY WRAPPING.....</b>	<b>39</b>
<b>7.5</b>	<b>KEY ENTRY AND OUTPUT.....</b>	<b>39</b>
<b>7.6</b>	<b>KEY STORAGE .....</b>	<b>40</b>
<b>7.7</b>	<b>KEY ARCHIVAL .....</b>	<b>40</b>
<b>7.8</b>	<b>KEY ZEROIZATION .....</b>	<b>40</b>
<b><u>8</u></b>	<b><u>SELF-TESTS .....</u></b>	<b><u>40</u></b>
<b>8.1</b>	<b>POWER-ON SELF-TESTS .....</b>	<b>40</b>
<b>8.2</b>	<b>CONDITIONAL SELF-TESTS.....</b>	<b>41</b>
<b><u>9</u></b>	<b><u>DESIGN ASSURANCE .....</u></b>	<b><u>41</u></b>
<b><u>10</u></b>	<b><u>MITIGATION OF OTHER ATTACKS.....</u></b>	<b><u>42</u></b>
<b><u>11</u></b>	<b><u>SECURITY LEVELS .....</u></b>	<b><u>42</u></b>
<b><u>12</u></b>	<b><u>ADDITIONAL DETAILS .....</u></b>	<b><u>43</u></b>
<b><u>13</u></b>	<b><u>APPENDIX A – HOW TO VERIFY WINDOWS VERSIONS AND DIGITAL SIGNATURES .....</u></b>	<b><u>44</u></b>
<b>13.1</b>	<b>HOW TO VERIFY WINDOWS VERSIONS .....</b>	<b>44</b>

**13.2**    **HOW TO VERIFY WINDOWS DIGITAL SIGNATURES .....44**

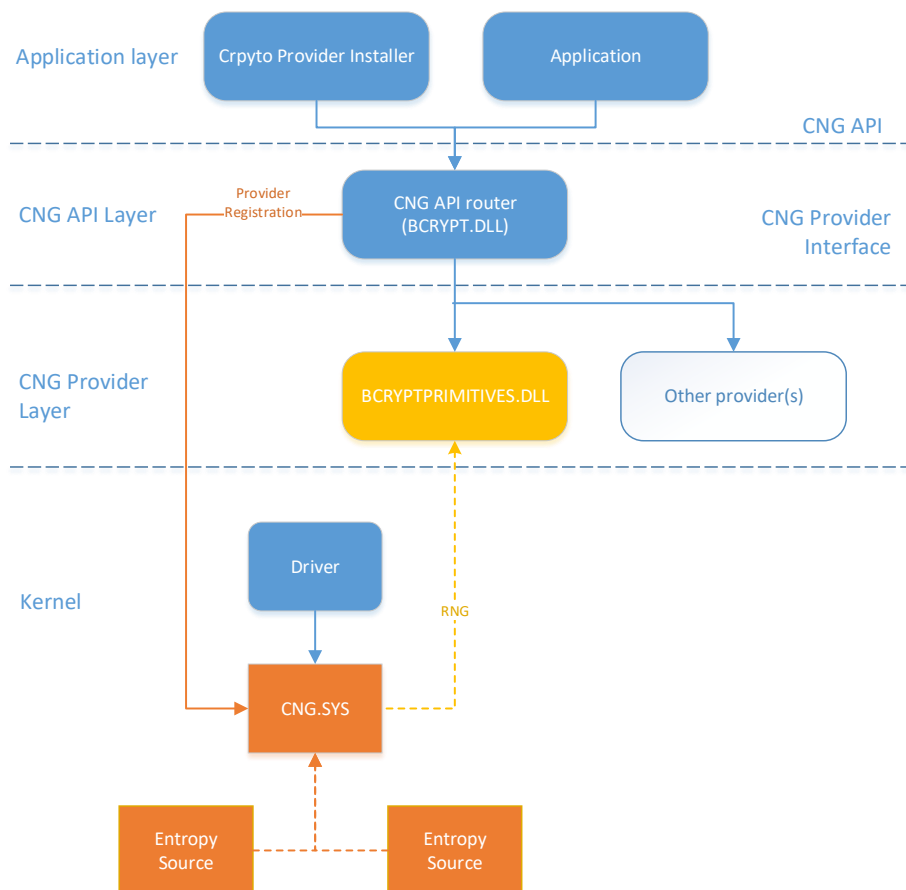
**14**      **APPENDIX B – REFERENCES.....45**

## 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. For the purpose of this validation, Cryptographic Primitives Library is classified as a Software-Hybrid cryptographic module because the validated platforms all implement the AES-NI instruction set.

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 Server 2019 build 10.0.17763.10021 and 10.0.17763.10127

## 1.2 Validated Platforms



The editions covered by this validation are:



- Windows Server 2019 Datacenter Core

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 Server listed in the table below are all 64-bit Intel architecture and implement the AES-NI instruction set but not the SHA Extensions.

*Table 1 Validated Platforms*

Computer	Windows Server 2019 Datacenter Core	Processor Image
<b>Microsoft Azure Stack Edge - Dell XR2 - Intel Xeon Silver 4114</b>	√	 wikichip.org
<b>Microsoft Azure Stack Hub - Dell PowerEdge R640 - Intel Xeon Gold 6230</b>	√	 wikichip.org

<b>Microsoft Azure Stack Hub - Dell PowerEdge R840 - Intel Xeon Platinum 8260</b>	√	 wikichip.org
<b>Microsoft Azure Stack Edge Rugged - Rugged Mobile Appliance – Intel Xeon D-1559</b>	√	 wikichip.org

### 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. As a general-purpose cryptographic library, it is the operator's

responsibility to configure services or applications which use cryptographic services to specify only FIPS-Approved algorithms, if that is required by their local security policy. 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

## 2.1 Cryptographic Boundary

The software-hybrid cryptographic boundary for the Cryptographic Primitives Library module consists of disjoint software and hardware components within the same physical boundary of the host platform. The software components are defined as the binary BCRYPTPRIMITIVES.DLL, and the hardware components are the CPUs running on each host platform.

## 2.2 FIPS 140-2 Approved Algorithms

Cryptographic Primitives Library implements the following FIPS-140-2 Approved algorithms:<sup>1</sup>

*Table 2 Approved Algorithms*

Algorithm	Windows Server 2019 build 10.0.17763.10021	Windows Server 2019 build 10.0.17763.10127
<b>FIPS 180-4 SHS SHA-1, SHA-256, SHA-384, and SHA-512</b>	#C1577	#C2044
<b>FIPS PUB 198-1 HMAC-SHA-1<sup>2</sup> and HMAC-SHA-256</b>	#C1577	#C2044
<b>FIPS 197 AES-128, AES-192, and AES-256 in ECB, CBC, CFB8, CFB128, and CTR modes (Generate Symmetric Key)</b>	#C1577	#C2044
<b>NIST SP 800-38B and SP 800-38C AES-128, AES-192, and AES-256 in CCM and CMAC modes (Generate Symmetric Key)</b>	#C1577	#C2044
<b>NIST SP 800-38D AES-128, AES-192, and AES-256 GCM decryption and GMAC (Generate Symmetric Key)</b>	#C1577	#C2044
<b>NIST SP 800-38E XTS-AES XTS-128 and XTS-256<sup>3</sup> (Generate Symmetric Key)</b>	#C1577	#C2044

<sup>1</sup> This module may not use some of the capabilities described in each CAVP certificate.

<sup>2</sup> For HMAC, only key sizes that are  $\geq$  112 bits in length are used by the module in FIPS mode.

<sup>3</sup> 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.

<b>FIPS 186-4 RSA PKCS#1 (v1.5) and RSASSA-PSS digital signature generation and verification with 1024<sup>4</sup>, 2048, and 3072 moduli; supporting SHA-1<sup>5</sup>, SHA-256, SHA-384, and SHA-512 (Generate Asymmetric Key / Signature)</b>	#C1577	#C2044
<b>FIPS 186-4 RSA key-pair generation with 2048 and 3072 moduli (Generate Asymmetric Key / Signature)</b>	#C1577	#C2044
<b>FIPS 186-4 ECDSA key pair generation and verification, signature generation and verification with the following NIST curves: P-256, P-384, P-521 (Generate Asymmetric Key / Signature)</b>	#C1577	#C2044
<b>FIPS 186-4 DSA PQG generation and verification, signature generation and verification (Generate Asymmetric Key / Signature), and key generation (2048- and 3072-bit moduli)</b>	#C1577	#C2044
<b>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</b>	#C1577	#C2044
<b>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</b>	#C1577	#C2044
<b>NIST SP 800-56B RSADP mod 2048 (CVL)</b>	#C1577	#C2044
<b>NIST SP 800-90A AES-256 counter mode DRBG</b>	#C1577	#C2044
<b>NIST SP 800-67r1 Triple-DES (3 key encryption / decryption) in ECB, CBC, CFB8 and CFB64 modes</b>	#C1577	#C2044
<b>NIST SP 800-108 Key Derivation Function (KDF) CMAC-AES (128, 192, 256), HMAC (SHA1, SHA-256, SHA-384, SHA-512)</b>	#C1584	#C2050
<b>NIST SP 800-38F AES Key Wrapping (KW) (128, 192, and 256) (Distributing Keys)</b>	#C1584	#C2050

<sup>4</sup> RSA 1024 is only applicable for signature verification.

<sup>5</sup> SHA-1 is only acceptable for legacy signature verification.

<b>Key Transport Scheme (KTS) using NIST SP 800-38F AES Key Wrapping (key establishment methodology provides between 128 and 256 bits of encryption strength)</b>	#C1584	#C2050
<b>NIST SP 800-135 IKEv1, IKEv2 and TLS KDF primitives (CVL)<sup>6</sup></b>	#C1577	#C2044
<b>NIST SP 800-132 KDF (also known as PBKDF) with HMAC (SHA-1, SHA-256, SHA-384, SHA-512) as the pseudo-random function (Derivation of Keys)</b>	Vendor Affirmed	Vendor Affirmed
<b>NIST SP 800-133 (sections 5.1, 5.2, 6.1, and 6.2) Cryptographic Key Generation</b>	Vendor Affirmed	Vendor Affirmed

## 2.3 Non-Approved Algorithms

Cryptographic Primitives Library implements the following non-Approved algorithms.

### Non-Approved algorithms allowed in the Approved mode of operation:

- SHA-1 hash, only for the use of legacy digital signature verification. Any use outside of legacy digital signature verification is disallowed.
- MD5 and HMAC-MD5 (allowed in TLS and EAP-TLS, with no security claimed, per FIPS 140-2 IG 1.23, example scenario 2a).
- ECDH (non-compliant) with the following curves and strengths, which are allowed in FIPS mode per FIPS 140-2 IG A.2:
  - brainpoolP224r1 (112 bits)
  - brainpoolP224t1 (112 bits)
  - brainpoolP256r1 (128 bits)
  - brainpoolP256t1 (128 bits)
  - brainpoolP320r1 (160 bits)
  - brainpoolP320t1 (160 bits)
  - brainpoolP384r1 (192 bits)
  - brainpoolP384t1 (192 bits)
  - brainpoolP512r1 (256 bits)
  - brainpoolP512t1 (256 bits)
  - nistP224 (112 bits)

<sup>6</sup> 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. These have been issued Component Validation List (CVL) certificates.

- numsP256t1 (128 bits)
- numsP384t1 (192 bits)
- numsP512t1 (256 bits)
- secP224k1 (112 bits)
- secP224r1 (112 bits)
- secP256k1 (128 bits)
- secP256r1 (128 bits)
- secP384r1 (192 bits)
- secP521r1 (256 bits)
- wtls12 (112 bits)
- x962P239v1 (120 bits)
- x962P239v2 (120 bits)
- x962P239v3 (120 bits)
- x962P256v1 (128 bits)
- Curve25519 (128 bits)

**Non-Approved algorithms disallowed in the Approved mode of operation:**

- IEEE 1619-2007 XTS-AES, XTS-128 and XTS-256 (non-compliant).
- Non-compliant NIST SP 800-67r1 Triple-DES (2 key legacy-use decryption)
- Non-compliant ANSI X9.63 and X9.42 key derivation.
- Non-compliant 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 (non-compliant).
- NIST SP 800-38D AES-128, AES-192, and AES-256 GCM encryption (non-compliant).
- FIPS 186-2 DSA with key length of 1024 bits.
- SHA-1 hash for digital signature generation.
- If HMAC-SHA1 is used, key sizes less than 112 bits (14 bytes) are disallowed for usage in HMAC generation, as per SP 800-131A (non-compliant).
- RSA 1024-bits for digital signature generation (non-compliant).
- RC2, RC4, MD2, and MD4.
- 2-Key DES encryption.
- DES in ECB, CBC, CFB8 and CFB64 modes.
- Legacy CAPI KDF (proprietary).
- RSA encrypt/decrypt (non-compliant).
- ECDH (non-compliant) with the following curves and strengths:
  - brainpoolP192r1 (96 bits)
  - brainpoolP192t1 (96 bits)
  - ec192wapi (96 bits)
  - nistP192 (96 bits)
  - secP160k1 (80 bits)
  - secP160r1 (80 bits)
  - secP160r2 (80 bits)
  - secP192k1 (96 bits)
  - secP192r1 (96 bits)

- wtls7 (80 bits)
- wtls9 (80 bits)
- x962P192v1 (96 bits)
- x962P192v2 (96 bits)
- x962P192v3 (96 bits)
- brainpoolP160r1 (80 bits)

## 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 Memory Integrity, called HVCI in previous Windows Server versions, is not enabled, Code Integrity (module certificate [#4602](#)) provides:

- CAVP certificates #C1577 and #C2044 (Windows Server) for FIPS 186-4 RSA PKCS#1 (v1.5) digital signature verification with 2048 moduli; supporting SHA-256
- CAVP certificates #C1577 and #C2044 (Windows Server) for FIPS 180-4 SHS SHA-256

When Memory Integrity, called HVCI in previous Windows Server versions, is enabled, Secure Kernel Code Integrity (module certificate [#4640](#)) provides:

- CAVP certificates #C1577 and #C2044 (Windows Server) for FIPS 186-4 RSA PKCS#1 (v1.5) digital signature verification with 2048 moduli; supporting SHA-256
- CAVP certificates #C1577 and #C2044 (Windows Server) for FIPS 180-4 SHS SHA-256

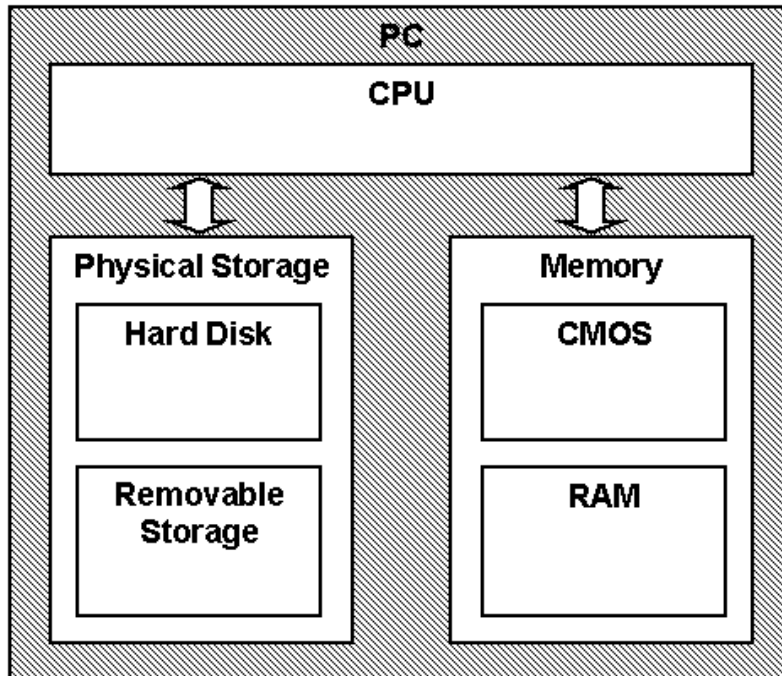
The Cryptographic Primitives Library depends on the Kernel Mode Cryptographic Primitives Library (module certificate [#4670](#)) non-deterministic random number generator (NDRNG) for AES-CTR DRBG Entropy Input. The NDRNG that provides the entropy is not a FIPS Approved algorithm, but is allowed by FIPS 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:



Note: The CPU provides Processor Algorithm Accelerator (PAA)

## 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 Server, 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,  
    PCHAR 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,  
    PCHAR pbKeyObject,  
    ULONG cbKeyObject,  
    PCHAR 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<sup>7</sup>. 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,  
    PCHAR 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.

## 3.2.3 Random Number Generation

### 3.2.3.1 *BCryptGenRandom*

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

The BCryptGenRandom() function fills a buffer with random bytes. BCRYPTPRIMITIVES.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.

---

<sup>7</sup> 1024 bits is not an approved key length for DSA.

### 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,  
    PCHAR pbKeyObject,  
    ULONG cbKeyObject,  
    PCHAR 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,  
    PCHAR 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,  
    PCHAR 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,  
    PCHAR 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,  
    PCHAR pbInput,  
    ULONG cbInput,  
    VOID *pPaddingInfo,  
    PCHAR pbIV,  
    ULONG cbIV,  
    PCHAR 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,  
    PCHAR pbHashObject,  
    ULONG cbHashObject,  
    PCHAR 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,  
    PCHAR 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,  
    PCHAR 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,  
    PCHAR 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,  
    PCHAR pbSecret,  
    ULONG cbSecret,  
    PCHAR pbInput,  
    ULONG cbInput,  
    PCHAR 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,  
    PCHAR pbInput,  
    ULONG cbInput,  
    PCHAR 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 *pAgreedSecret,
    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. This function is used by a non-Approved service that cannot be used in FIPS mode.

#### 3.2.8.2 BCryptDeriveKey

```

NTSTATUS WINAPI BCryptDeriveKey(
    BCRYPT_SECRET_HANDLE hSharedSecret,
    LPCWSTR pszKDF,
    BCRYPT_BUFFER_DESC *pParameterList,
    PCHAR 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) PCHAR 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,
    PCHAR pbPassword,
    ULONG cbPassword,
    PCHAR pbSalt,
    ULONG cbSalt,
    ULONGLONG cIterations,
    PCHAR 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.

*Table 3 Functions*

Function Name	Description
<b>BCryptEnumAlgorithms</b>	Enumerates the algorithms for a given set of operations.
<b>BCryptEnumProviders</b>	Returns a list of CNG providers for a given algorithm.
<b>BCryptRegisterConfigChangeNotify</b>	This is deprecated beginning with Windows Server.
<b>BCryptResolveProviders</b>	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.
<b>BCryptAddContextFunctionProvider</b>	Adds a cryptographic function provider to the list of providers that are supported by an existing CNG context.
<b>BCryptRegisterProvider</b>	Registers a CNG provider.
<b>BCryptUnregisterProvider</b>	Unregisters a CNG provider.
<b>BCryptUnregisterConfigChangeNotify</b>	Removes a CNG configuration change event handler.
<b>BCryptGetFipsAlgorithmMode</b>	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.

<b>BCryptQueryProviderRegistration</b>	Retrieves information about a CNG provider.
<b>BCryptEnumRegisteredProviders</b>	Retrieves information about the registered providers.
<b>BCryptCreateContext</b>	Creates a new CNG configuration context.
<b>BCryptDeleteContext</b>	Deletes an existing CNG configuration context.
<b>BCryptEnumContexts</b>	Obtains the identifiers of the contexts in the specified configuration table.
<b>BCryptConfigureContext</b>	Sets the configuration information for an existing CNG context.
<b>BCryptQueryContextConfiguration</b>	Retrieves the current configuration for the specified CNG context.
<b>BCryptAddContextFunction</b>	Adds a cryptographic function to the list of functions that are supported by an existing CNG context.
<b>BCryptRemoveContextFunction</b>	Removes a cryptographic function from the list of functions that are supported by an existing CNG context.
<b>BCryptEnumContextFunctions</b>	Obtains the cryptographic functions for a context in the specified configuration table.
<b>BCryptConfigureContextFunction</b>	Sets the configuration information for the cryptographic function of an existing CNG context.
<b>BCryptQueryContextFunctionConfiguration</b>	Obtains the cryptographic function configuration information for an existing CNG context.
<b>BCryptEnumContextFunctionProviders</b>	Obtains the providers for the cryptographic functions for a context in the specified configuration table.
<b>BCryptSetContextFunctionProperty</b>	Sets the value of a named property or a cryptographic function in an existing CNG context.
<b>BCryptQueryContextFunctionProperty</b>	Obtains the value of a named property for a cryptographic function in an existing CNG context.
<b>BCryptSetAuditingInterface</b>	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).

*Table 4 Services*

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, DSA, ECDH, ECDSA, RC2, RC4, DES, Triple-DES, AES, and HMAC (RC2, RC4, and DES are non-Approved and disallowed 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	<ul style="list-style-type: none"> <li>• Triple-DES with 3 key in ECB, CBC, CFB8, and CFB64 modes</li> <li>• AES-128, AES-192, and AES-256 in ECB, CBC, CFB8, CFB128, and CTR modes;</li> <li>• AES-128, AES-192, and AES-256 in CCM, CMAC, and GMAC modes;</li> <li>• AES-128, AES-192, and AES-256 GCM decryption;</li> <li>• XTS-AES XTS-128 and XTS-256;</li> </ul>	Symmetric Keys Asymmetric Public Keys Asymmetric Private Keys

	<ul style="list-style-type: none"> <li>• SP 800-56B RSADP mod 2048</li> <li>• (AES GCM encryption, RC2, RC4, RSA, and DES, which are all disallowed in FIPS mode)</li> </ul>	
Hashing and Message Authentication	<ul style="list-style-type: none"> <li>• FIPS 180-4 SHS SHA-1, SHA-256, SHA-384, and SHA-512;</li> <li>• FIPS 180-4 SHA-1, SHA-256, SHA-384, SHA-512 HMAC;</li> <li>• AES-128, AES-192, and AES-256 in CCM, CMAC, and GMAC;</li> <li>• MD5 and HMAC-MD5 (allowed in TLS and EAP-TLS);</li> <li>• MD2 and MD4 (disallowed in FIPS mode)</li> </ul>	Symmetric Keys (for HMAC, AES CCM, AES CMAC, and AES GMAC)
Signing and Verification	<ul style="list-style-type: none"> <li>• FIPS 186-4 RSA (RSASSA-PKCS1-v1_5 and RSASSA-PSS) digital signature generation and verification with 2048 and 3072 modulus; supporting SHA-1<sup>8</sup>, SHA-256, SHA-384, and SHA-512</li> <li>• FIPS 186-4 ECDSA with the following NIST curves: P-256, P-384, P-521</li> <li>• FIPS 186-4 DSA signature generation and verification</li> </ul>	Asymmetric Public Keys Asymmetric RSA Private Keys Asymmetric ECDSA Public Keys Asymmetric ECDSA Private keys Asymmetric DSA Public Keys Asymmetric DSA Private Keys
Secret Agreement and Key Derivation	<ul style="list-style-type: none"> <li>• KAS – SP 800-56A Diffie-Hellman Key Agreement; Finite Field Cryptography (FFC)</li> <li>• 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 <a href="#">Non-Approved Algorithms</a></li> <li>• SP 800-108 Key Derivation Function (KDF) CMAC-AES (128, 192, 256), HMAC (SHA1, SHA-256, SHA-384, SHA-512)</li> <li>• SP 800-132 PBKDF</li> </ul>	DH Private and Public Values ECDH Private and Public Values IKEv1 and IKEv2 DH shared secrets TLS Pre-Master Secret

<sup>8</sup> SHA-1 is only acceptable for signature verification.

	<ul style="list-style-type: none"> <li>• SP 800-135 IKEv1, IKEv2, and TLS KDF primitives</li> <li>• Legacy CAPI KDF (cannot be used in FIPS mode)</li> </ul>	
Show Status	None	None
Self-Tests	See Section <a href="#">Self-Tests</a> for the list of algorithms	None
Zeroizing Cryptographic Material	All	All

#### 4.2.2 Mapping of Services, Export Functions, and Invocations

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

*Table 5 Services, 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.
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 non-Approved services and non-security relevant or non-approved APIs exported from the crypto module.

*Table 6 Non-Approved Services and Export Functions*

Function Name	Description
<b>BCryptDeriveKeyCapi</b>	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.

## 4.3 Authentication

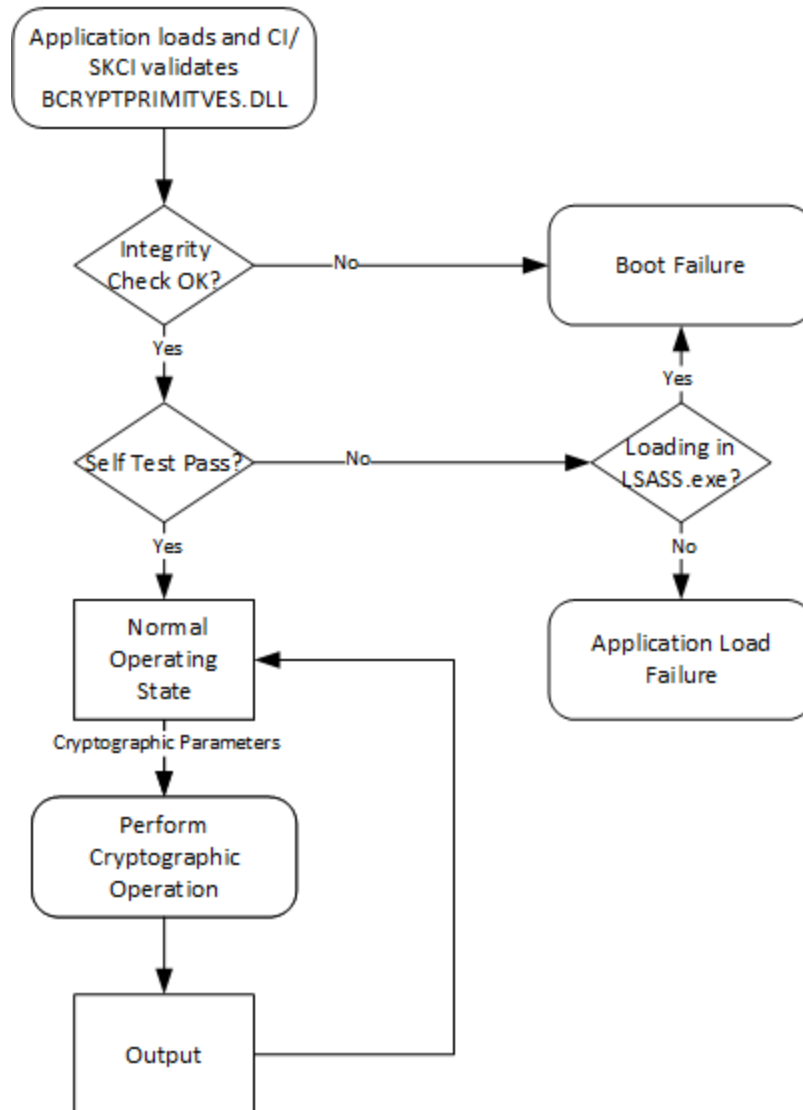
Cryptographic Primitives Library does not provide authentication of users. Roles are implicitly assumed based on the services that are executed.

# 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 Server operating system running on a supported hardware platform listed in section 1.2.

### 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.

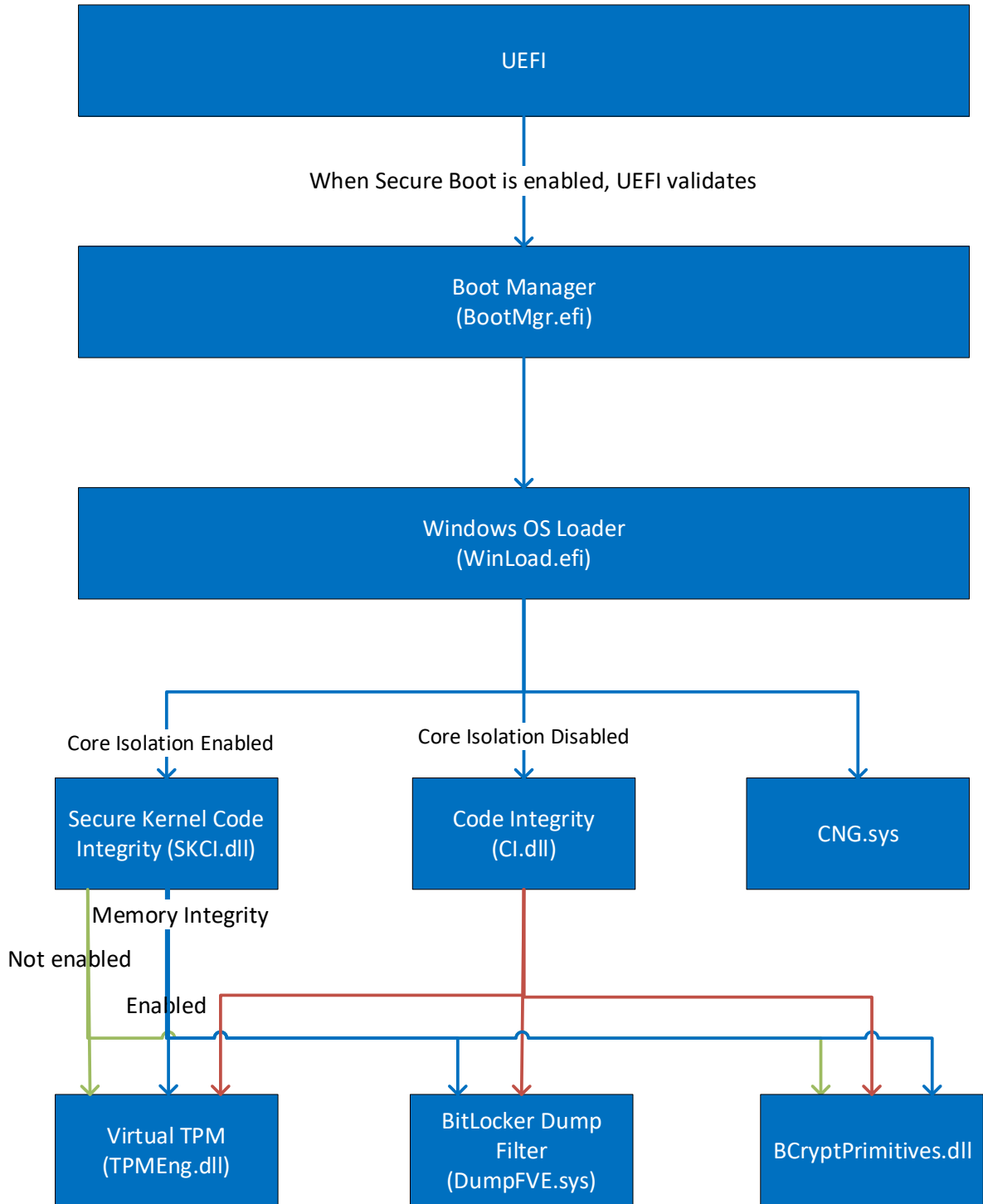
### 6.2 Cryptographic Isolation

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:

- Windows Server 2019 build 10.0.17763.10021 and 10.0.17763.10127



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:

*Table 7 CSPs*

Security Relevant Data Item	Description
<b>Symmetric encryption/decryption keys</b>	Keys used for AES encryption/decryption and Triple-DES encryption / decryption. Key sizes for AES are 128, 192, and 256 bits, and key size for Triple-DES is 192 bits.
<b>HMAC keys</b>	Keys used for HMAC-SHA1, HMAC-SHA256, HMAC-SHA384, and HMAC-SHA512
<b>Asymmetric DSA Public Keys</b>	Keys used for the verification of DSA digital signatures. Key sizes are 2048 and 3072 bits.
<b>Asymmetric DSA Private Keys</b>	Keys used for the calculation of DSA digital signatures. Key sizes are 2048 and 3072 bits. May be generated using DSA key generation.
<b>Asymmetric ECDSA Public Keys</b>	Keys used for the verification of ECDSA digital signatures. Curve sizes are P-256, P-384, and P-521. May be generated using ECDSA Key Generation.
<b>Asymmetric ECDSA Private Keys</b>	Keys used for the calculation of ECDSA digital signatures. Curve sizes are P-256, P-384, and P-521.
<b>Asymmetric RSA Public Keys</b>	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.
<b>Asymmetric RSA Private Keys</b>	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.
<b>AES-CTR DRBG Entropy Input</b>	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 <sup>9</sup>
<b>AES-CTR DRBG Seed</b>	A 384 bit secret value maintained internal to the module that provides the seed material for AES-CTR DRBG output <sup>10</sup>

<sup>9</sup> [Microsoft Common Criteria Windows Security Target](#), Page 29.

<sup>10</sup> Recommendation for Random Number Generation Using Deterministic Random Bit Generators, NIST SP 800-90A Revision 1, page 49.

<b>AES-CTR DRBG V</b>	A 128 bit secret value maintained internal to the module that provides the entropy material for AES-CTR DRBG output <sup>11</sup>
<b>AES-CTR DRBG key</b>	A 256 bit secret value maintained internal to the module that provides the entropy material for AES-CTR DRBG output <sup>12</sup>
<b>DH Private and Public values</b>	Private and public values used for Diffie-Hellman key establishment. Key sizes are 2048 to 4096 bits.
<b>ECDH Private and Public values</b>	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 2.3.
<b>IKEv1 and IKEv2 DH shared secrets</b>	Diffie-Hellman shared secret lengths are 2048 (SHA 256), 256 (SHA 256 ), and 384 (SHA 384 ).
<b>TLS Pre-Master Secret</b>	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.

*Table 8 Access Control Policy*

<b>Cryptographic Primitives Library crypto module</b> <b>Service Access Policy</b>	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 Secrets	TLS Pre-Master Secret
<b>Algorithm Providers and Properties</b>													
<b>Random Number Generation</b>											x		
<b>Key and Key-Pair Generation</b>	w d	w d	w d	w d	w d	w d	w d	w d	wd	wd	x		
<b>Key Entry and Output</b>	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw			
<b>Encryption and Decryption</b>	x												
<b>Hashing and Message Authentication</b>		xw											
<b>Signing and Verification</b>			x	x	x	x	x	x			x		

<sup>11</sup> Ibid.

<sup>12</sup> Ibid.

<b>Secret Agreement and Key Derivation</b>										x	x	x	x	x
<b>Show Status</b>														
<b>Self-Tests</b>														
<b>Zeroizing Cryptographic Material</b>	w d	w d	w d	w d	w d	w d	w d	w d	w d	wd	wd	wd	wd	wd

## 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 800-133r2 (Section 4).

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

In its Approved mode of operation, the Cryptographic Primitives Library supports approved Diffie-Hellman key agreement (DH), Elliptic Curve Diffie-Hellman key agreement (ECDH), manual methods to establish keys, and the following key derivation functions (KDFs) to derive key material from a specified secret value or password.

- `BCRYPT_KDF_SP80056A_CONCAT`. This KDF supports the Concatenation KDF as specified in SP 800-56A (Section 5.8.1).
- `BCRYPT_SP80056A_CONCAT_ALGORITHM`. This KDF supports the Concatenation KDF as specified in SP 800-56Ar2 (Section 5.8.1).
- `BCRYPT_KDF_HASH`. This KDF supports FIPS approved SP800-56A (Section 5.8) 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_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\_PBKDF2\_ALGORITHM. This KDF supports the Password Based Key Derivation Function specified in SP 800-132 (Section 5.3).
- BCRYPT\_KDF\_TLS\_PR. 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.

The following non-Approved KDFs are supported by the module, but disallowed in FIPS mode.

- BCRYPT\_KDF\_HASH. This KDF implements X9.63 and X9.42 key derivation.
- The proprietary KDF, BCRYPT\_CAPI\_KDF\_ALGORITHM is described at <https://msdn.microsoft.com/library/windows/desktop/aa379916.aspx>. This KDF 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.<sup>13</sup>

#### 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

---

<sup>13</sup> 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, such 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

In 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  $\neq$  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 Server operating system.

The Windows Server 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 Server 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 Server 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:

*Table 9 Mitigation of Other Attacks*

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.

*Table 10 Security Levels*

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	1

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

The Cryptographic Primitives Library is a multi-chip standalone software-hybrid module whose host platforms meet the level 1 physical security requirements. The module consists of production-grade components that include standard passivation techniques and is entirely contained within a metal or hard plastic production-grade enclosure that may include doors or removable covers.

## 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 Server 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](#)

Table 11 References

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

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