Microsoft

Security Policy for FIPS 140-2 Validation

Code Integrity (ci.dll) in
Microsoft Windows 8.1 Enterprise
Windows Server 2012 R2
Windows Storage Server 2012 R2
Surface Pro 3
Surface Pro 2
Surface Pro
Surface 2
Surface
Windows RT 8.1
Windows Phone 8.1
Windows Embedded 8.1 Industry Enterprise
StorSimple 8000 Series
Azure StorSimple Virtual Array Windows Server 2012 R2

DOCUMENT INFORMATION

Version Number	2.1
Updated On	April 12, 2017

© 2017 Microsoft. All Rights Reserved

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

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

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

Complying with all applicable copyright laws is the responsibility of the user. This work is licensed under the Creative Commons Attribution-NoDerivs-NonCommercial License (which allows redistribution of the work). To view a copy of this license, visit http://creativecommons.org/licenses/by-nd-nc/1.0/ or send a letter to Creative Commons, 559 Nathan Abbott Way, Stanford, California 94305, USA.

Microsoft may have patents, patent applications, trademarks, copyrights, or other intellectual property rights covering subject matter in this document. Except as expressly provided in any written license agreement from Microsoft, the furnishing of this document does not give you any license to these patents, trademarks, copyrights, or other intellectual property.

© 2017 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.

CHANGE HISTORY

Date	Version	Updated By	Change
29 OCT 2013	0.1	Tim Myers	First Draft
7 MAY 2014	0.2	Tim Myers	Second Draft; FIPS 186-4 and FIPS 180-4 updates; processor name updates
1 JUL 2014	0.3	Tim Myers	Third Draft; added Security Levels Table and cryptographic algorithm certificate numbers
10 JUL 2014	1.0	Tim Myers	First Release to Validators; includes all algorithm and module certificate numbers
18 JUL 2014	1.1	Tim Myers	Update platforms; consolidate services
5 NOV 2014	1.2	Tim Myers	Update platform names
11 DEC 2014	1.3	Tim Myers	Update platform names
17 DEC 2014	1.4	Tim Myers	Update IG G.5 platforms; update binary versions; add StorSimple
29 JAN 2015	1.5	Tim Myers	Update Design Assurance
20 FEB 2015	1.6	Tim Myers	Add StorSimple to Validated Platforms
9 MAR 2015	1.7	Tim Myers	Prepare for publication
17 MAR 2015	1.8	Tim Myers	Reorder StorSimple 8100 OE description
20 APR 2015	2.0	Tim Myers	Add Microsoft Surface Pro 3 to Validated Platforms
12 APR 2017	2.1	FIPS Team	Add StorSimple Virtual Array to Validated Platforms

© 2017 Microsoft. All Rights Reserved
This Security Policy is non-proprietary and may be reproduced only in its original entirety (without revision).

Table of Contents

1 INTRODUCTION 7

1.1 LIST OF CRYPTOGRAPHIC MODULE BINARY EXECUTABLES 7
1.2 BRIEF MODULE DESCRIPTION 7
1.3 VALIDATED PLATFORMS 7
1.4 CRYPTOGRAPHIC BOUNDARY 9

2 SECURITY POLICY 9

2.1 FIPS 140-2 APPROVED ALGORITHMS 11
2.2 NON-APPROVED ALGORITHMS 12
2.3 CRYPTOGRAPHIC BYPASS 12
2.4 MACHINE CONFIGURATIONS 12

3 INTEGRITY CHAIN OF TRUST 12

3.1 CONVENTIONAL BIOS AND UEFI WITHOUT SECURE BOOT ENABLED 12
3.2 UEFI WITH SECURE BOOT ENABLED 12

4 PORTS AND INTERFACES 13

4.1 CODE INTEGRITY EXPORT FUNCTIONS 13
4.1.1 CIINITIALIZE() 13
4.1.1.1 CiValidateImageHeader() 14
4.1.1.2 CiValidateImageData() 14
4.1.1.3 CiQueryInformation() 15
4.1.1.4 CiQueryImageSignature() 15
4.1.1.5 CiImportRoots() 15
4.1.1.6 CiGetFileCache() 15
4.1.1.7 CiSetFileCache() 15
4.1.1.8 CiHashMemorySha256() 15
4.1.2 CIGETPEINFORMATION 15
4.1.3 CIVERIFYHASHINCATALOG 15
4.1.4 CICHECKSIGNEDFILE 15
4.1.5 CIFINDPAGEHASHESINCATALOG 15
4.1.6 CIFINDPAGEHASHESINSIGNEDFILE 16
4.1.7 CIFREEPOLICYINFO 16
4.2 CONTROL INPUT INTERFACE 16

© 2017 Microsoft. All Rights Reserved
This Security Policy is non-proprietary and may be reproduced only in its original entirety (without revision).

4.3 STATUS OUTPUT INTERFACE 16
4.4 DATA INPUT INTERFACE 16
4.5 DATA OUTPUT INTERFACE 16

5 SPECIFICATION OF ROLES 16

5.1 MAINTENANCE ROLES 16
5.2 MULTIPLE CONCURRENT INTERACTIVE OPERATORS 16

6 SERVICES 17

6.1 SHOW STATUS SERVICES 19
6.2 SELF-TEST SERVICES 19
6.3 SERVICE INPUTS / OUTPUTS 19

7 OPERATIONAL ENVIRONMENT 19

8 AUTHENTICATION 19

9 CRYPTOGRAPHIC KEY MANAGEMENT 19

9.1 CRITICAL SECURITY PARAMETERS 20
9.2 ACCESS CONTROL POLICY 20

10 SELF-TESTS 20

10.1 POWER-ON SELF-TESTS 20
10.2 CONDITIONAL SELF-TESTS 20

11 DESIGN ASSURANCE 20

12 MITIGATION OF OTHER ATTACKS 22

13 SECURITY LEVELS 22

14 ADDITIONAL DETAILS 22

15 APPENDIX A – HOW TO VERIFY WINDOWS VERSIONS AND DIGITAL SIGNATURES 24

© 2017 Microsoft. All Rights Reserved
This Security Policy is non-proprietary and may be reproduced only in its original entirety (without revision).

15.1 HOW TO VERIFY WINDOWS VERSIONS 24
15.2 HOW TO VERIFY WINDOWS DIGITAL SIGNATURES 24

© 2017 Microsoft. All Rights Reserved

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

1 Introduction

Code Integrity is a Microsoft Windows 8.1 Enterprise, Windows Server 2012 R2, Windows Storage Server 2012 R2, Surface Pro 3, Surface Pro 2, Surface Pro, Surface 2, Surface, Windows RT 8.1, Windows Phone 8.1, Windows Embedded 8.1 Industry Enterprise, StorSimple 8000 Series, and Azure StorSimple Virtual Array Windows Server 2012 R2 (herein referred to as Windows 8.1 OEs) feature that verifies the integrity of several key Windows binary image files as they are loaded into memory from the disk.

Code Integrity is not a general purpose cryptographic module. It is validated under FIPS 140-2 because it implements cryptographic algorithms and provides the integrity checks for the Windows general purpose cryptographic modules.

1.1 List of Cryptographic Module Binary Executables

CI.DLL – Versions 6.3.9600 and 6.3.9600.17031 for Windows 8.1 OEs

1.2 Brief Module Description

Code Integrity is a dynamically-linked library used to verify the integrity of other binary executable code files.

1.3 Validated Platforms

The Code Integrity component listed in Section 1.1 was validated using the following machine configurations:

1. Microsoft Windows 8.1 Enterprise (x86) running on a Dell PowerEdge SC440 without AES-NI;
2. Microsoft Windows Embedded 8.1 Industry Enterprise (x86) running on a Dell PowerEdge SC440 without AES-NI;
3. Microsoft Windows 8.1 Enterprise (x86) running on a Dell Dimension E521 without AES-NI;
4. Microsoft Windows Embedded 8.1 Industry Enterprise (x86) running on a Dell Dimension E521 without AES-NI;
5. Microsoft Windows 8.1 Enterprise (x86) running on an Intel Core i7 with AES-NI running on an Intel Maho Bay;
6. Microsoft Windows Embedded 8.1 Industry Enterprise (x86) running on an Intel Core i7 with AES-NI running on an Intel Maho Bay;
7. Microsoft Windows 8.1 Enterprise (x86) running on an HP Compaq Pro 6305 with AES-NI;
8. Microsoft Windows Embedded 8.1 Industry Enterprise (x86) running on an HP Compaq Pro 6305 with AES-NI;
9. Microsoft Windows 8.1 Enterprise (x64) running on a Dell PowerEdge SC440 without AES-NI;
10. Microsoft Windows Embedded 8.1 Industry Enterprise (x64) running on a Dell PowerEdge SC440 without AES-NI;
11. Microsoft Windows Server 2012 R2 (x64) running on a Dell PowerEdge SC440 without AES-NI;
12. Microsoft Windows Storage Server 2012 R2 (x64) running on a Dell PowerEdge SC440 without AES-NI;
13. Microsoft Windows 8.1 Enterprise (x64) running on a Dell Dimension E521 without AES-NI;
14. Microsoft Windows Embedded 8.1 Industry Enterprise (x64) running on a Dell Dimension E521 without AES-NI;
15. Microsoft Windows Server 2012 R2 (x64) running on a Dell Dimension E521 without AES-NI;

© 2017 Microsoft. All Rights Reserved

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

16. Microsoft Windows Storage Server 2012 R2 (x64) running on a Dell Dimension E521 without AES-NI;
17. Microsoft Windows 8.1 Enterprise (x64) running on an Intel Core i7 with AES-NI running on an Intel Maho Bay;
18. Microsoft Windows Embedded 8.1 Industry Enterprise (x64) running on an Intel Core i7 with AES-NI running on an Intel Maho Bay;
19. Microsoft Windows Server 2012 R2 (x64) running on an Intel Core i7 with AES-NI running on an Intel Maho Bay;
20. Microsoft Windows Storage Server 2012 R2 (x64) running on an Intel Core i7 with AES-NI running on an Intel Maho Bay;
21. Microsoft Windows 8.1 Pro (x64) running on an Intel x64 Processor with AES-NI running on a Microsoft Surface Pro;
22. Microsoft Windows 8.1 Pro (x64) running on an Intel i5 with AES-NI running on a Microsoft Surface Pro 2;
23. Microsoft Windows 8.1 Enterprise (x64) running on an HP Compaq Pro 6305 with AES-NI;
24. Microsoft Windows Embedded 8.1 Industry Enterprise (x64) running on an HP Compaq Pro 6305 with AES-NI;
25. Microsoft Windows Server 2012 R2 (x64) running on an HP Compaq Pro 6305 with AES-NI;
26. Microsoft Windows Storage Server 2012 R2 (x64) running on an HP Compaq Pro 6305 with AES-NI;
27. Microsoft Windows RT 8.1 (ARMv7 Thumb-2) running on an NVIDIA Tegra 3 Tablet;
28. Microsoft Windows RT 8.1 (ARMv7 Thumb-2) running on a Microsoft Surface RT;
29. Microsoft Windows RT 8.1 (ARMv7 Thumb-2) running on a Microsoft Surface 2;
30. Microsoft Windows RT 8.1 (ARMv7 Thumb-2) running on a Qualcomm Tablet;
31. Microsoft Windows Phone 8.1 (ARMv7 Thumb-2) running on a Qualcomm Snapdragon S4 running on a Windows Phone 8.1;
32. Microsoft Windows Phone 8.1 (ARMv7 Thumb-2) running on a Qualcomm Snapdragon 400 running on a Windows Phone 8.1;
33. Microsoft Windows Phone 8.1 (ARMv7 Thumb-2) running on a Qualcomm Snapdragon 800 running on a Windows Phone 8.1;
34. Microsoft Windows 8.1 Enterprise (x64) running on a Dell Inspiron 660s without AES-NI and with PCLMULQDQ and SSSE 3;
35. Microsoft Windows Embedded 8.1 Industry Enterprise (x64) running on a Dell Inspiron 660s without AES-NI and with PCLMULQDQ and SSSE 3;
36. Microsoft Windows Server 2012 R2 (x64) running on a Dell Inspiron 660s without AES-NI and with PCLMULQDQ and SSSE 3;
37. Microsoft Windows Storage Server 2012 R2 (x64) running on a Dell Inspiron 660s without AES-NI and with PCLMULQDQ and SSSE 3;
38. Microsoft Windows 8.1 Enterprise (x64) running on an Intel Core i5 with AES-NI and with PCLMULQDQ and SSSE 3 running on a Microsoft Surface Pro 2;
39. Microsoft Windows Embedded 8.1 Industry Enterprise (x64) running on an Intel Core i7 with AES-NI and with PCLMULQDQ and SSSE 3 running on an Intel Maho Bay;
40. Microsoft Windows Server 2012 R2 (x64) running on an Intel Core i7 with AES-NI and with PCLMULQDQ and SSSE 3 running on an Intel Maho Bay;
41. Microsoft Windows Storage Server 2012 R2 (x64) running on an Intel Core i7 with AES-NI and with PCLMULQDQ and SSSE 3 running on an Intel Maho Bay;

© 2017 Microsoft. All Rights Reserved

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

42. Microsoft Windows 8.1 Enterprise (x64) running on an HP Compaq Pro 6305 with AES-NI and with PCLMULQDQ and SSSE 3;
43. Microsoft Windows Embedded 8.1 Industry Enterprise (x64) running on an HP Compaq Pro 6305 with AES-NI and with PCLMULQDQ and SSSE 3;
44. Microsoft Windows Server 2012 R2 (x64) running on an HP Compaq Pro 6305 with AES-NI and with PCLMULQDQ and SSSE 3;
45. Microsoft Windows Storage Server 2012 R2 (x64) running on an HP Compaq Pro 6305 with AES-NI and with PCLMULQDQ and SSSE 3;
46. Windows Server 2012 R2 (x64) running on a Microsoft StorSimple 8100 with an Intel Xeon E5-2648L without AES-NI;
47. Windows Server 2012 R2 (x64) running on a Microsoft StorSimple 8100 with an Intel Xeon E5-2648L with AES-NI;
48. Microsoft Windows 8.1 Pro (x64) running on an Intel Core i7 with AES-NI and with PCLMULQDQ and SSSE 3 running on a Microsoft Surface Pro 3;
49. Azure StorSimple Virtual Array Windows Server 2012 R2 on Hyper-V 6.3 on Windows Server 2012 R2 (x64) running on a Dell Precision Tower 5810 with PAA;
50. Azure StorSimple Virtual Array Windows Server 2012 R2 on VMware Workstation 12.5 on Windows Server 2012 R2 (x64) running on a Dell XPS 8700 with PAA

Code Integrity maintains FIPS 140-2 validation compliance (according to FIPS 140-2 PUB Implementation Guidance G.5) on the following platforms:

x86 Microsoft Windows 8.1
x86 Microsoft Windows 8.1 Pro

x64 Microsoft Windows 8.1
x64 Microsoft Windows Server 2012 R2 Datacenter

x64-AES-NI Microsoft Windows 8.1
x64-AES-NI Microsoft Windows Server 2012 R2 Datacenter

1.4 Cryptographic Boundary

The cryptographic boundary for Code Integrity is defined as the enclosure of the computer system, on which Code Integrity is to be executed. The physical configuration of Code Integrity, as defined in FIPS-140-2, is multi-chip standalone.

2 Security Policy

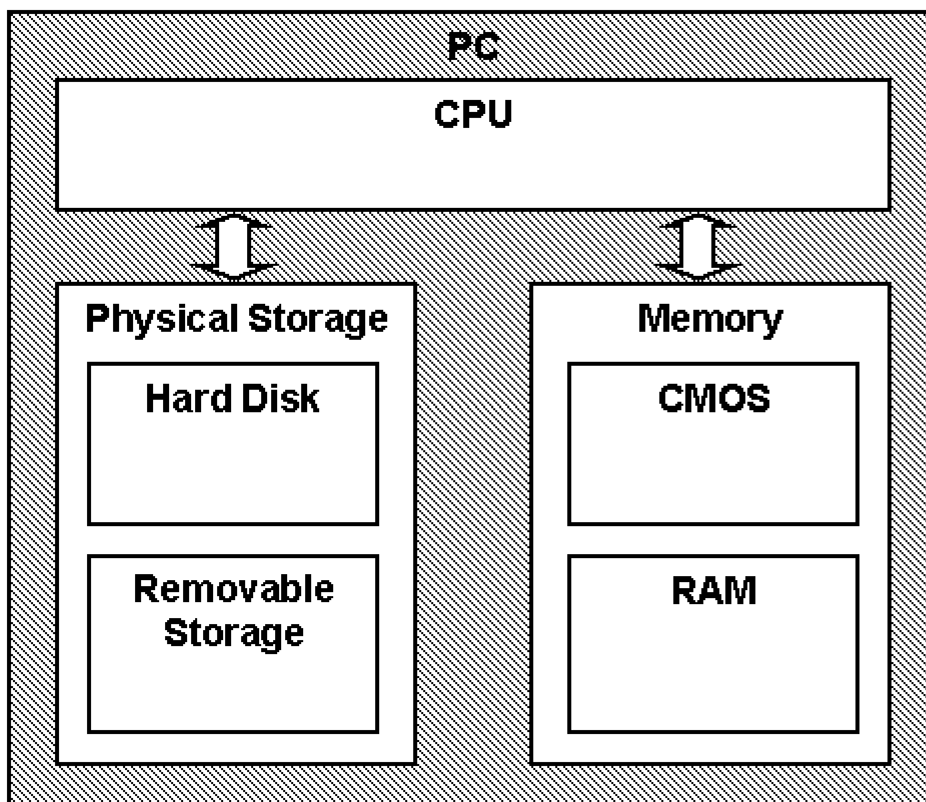
Code Integrity is considered to be in a FIPS mode of operation when the following rules are followed:

- Code Integrity is supported on Windows 8.1 OEs.
- Windows 8.1 OEs are operating systems supporting a “single user” mode where there is only one interactive user during a logon session.

© 2017 Microsoft. All Rights Reserved
This Security Policy is non-proprietary and may be reproduced only in its original entirety (without revision).

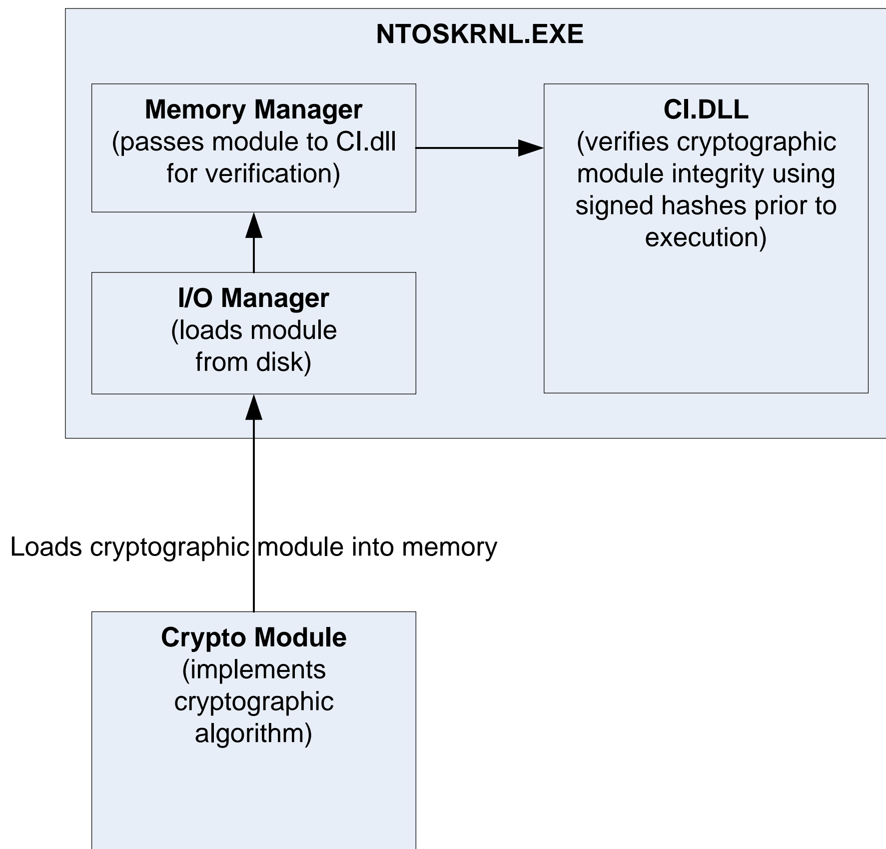
- Code Integrity is only in the "Approved mode of operation" when Windows is booted normally, meaning Debug mode has not been enabled and Driver Signing enforcement has not been disabled.
- The Debug mode status and Driver Signing enforcement status can be viewed by using the bcdedit tool.
- Code Integrity operates in FIPS mode of operation only when used with the FIPS approved version of Windows 8.1 OEs Winload OS Loader (winload.exe) validated to FIPS 140-2 under Cert. #2352 operating in FIPS mode.

The following diagram illustrates the master components of the Code Integrity module:



© 2017 Microsoft. All Rights Reserved
This Security Policy is non-proprietary and may be reproduced only in its original entirety (without revision).

The following diagram illustrates the interaction of Code Integrity with other cryptographic modules:



- Code Integrity's main service is to verify the integrity of digitally signed drivers and components within the computer (such as bcryptprimitives.dll, rsaenh.dll, dssenh.dll). In addition to this service, Code Integrity also provides status services. These status services indicate whether the aforementioned integrity checks passed.
- All services implemented within Code Integrity are available to the User and Crypto officer roles. The User and Crypto officer roles are assumed by the operating system or application processes that will invoke binary image verification in CI.dll. The Window Memory Manager is an example.
- Code Integrity verifies the integrity of the Windows 8.1 OEs general purpose cryptographic modules using the following FIPS-140-2 Approved algorithms.
    - RSA PKCS#1 (v1.5) verify with public key
    - SHA-1 hash
    - SHA-256 hash
    - SHA-384 hash
    - SHA-512 hash

2.1 FIPS 140-2 Approved Algorithms

Code Integrity implements the following FIPS-140-2 Approved algorithms:

© 2017 Microsoft. All Rights Reserved

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

- FIPS 186-4 RSA PKCS#1 (v1.5) digital signature verification with 1024, 2048, and 3072 moduli; supporting SHA-1, SHA-256, SHA-384, and SHA-512 (Cert. #1494)
- FIPS 180-4 SHS (SHA-1, SHA-256, SHA-384, SHA-512) (Cert. #2373)

Note that not all the algorithms and modes verified through the CAVP certificates listed are implemented by this module.

2.2 Non-Approved Algorithms

Code Integrity also includes a legacy implementation of MD5. MD5 is only used for backwards compatibility to verify the RSA signature over the file digest and certificate chains. MD5 is not allowed for use in file digests, which require a SHA-1 hash as the minimum.

2.3 Cryptographic Bypass

Cryptographic bypass is not supported by Code Integrity.

2.4 Machine Configurations

Code Integrity was tested using the machine configurations listed in Section 1.3 - Validated Platforms.

3 Integrity Chain of Trust

3.1 Conventional BIOS and UEFI without Secure Boot Enabled

Boot Manager is the start of the chain of trust. It cryptographically checks its own integrity during its startup. It then cryptographically checks the integrity of the Windows OS Loader (Winload.exe) before starting it. The Windows OS Loader checks the integrity of Code Integrity, which is protected by an RSA signature with a 2048-bit key and SHA-256 message digest, before loading it into memory. Code Integrity is used to verify the origin and integrity of Windows system binaries before they are loaded into memory and executed. Code Integrity also ensures kernel mode drivers are appropriately signed. When User Mode Code Integrity (UMCI) is enabled, Code Integrity ensured that all binaries are appropriately signed.

3.2 UEFI with Secure Boot Enabled

On UEFI systems with Secure Boot enabled, Boot Manager is still the OS binary from which the integrity of all other OS binaries is rooted, and it does cryptographically check its own integrity. However, Boot Manager's integrity is also checked and verified by the UEFI firmware, which is the root of trust on Secure Boot enabled systems.

© 2017 Microsoft. All Rights Reserved

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

4 Ports and Interfaces

4.1 Code Integrity export functions

The following list contains all the functions exported by Code Integrity to its callers inside the kernel. Code Integrity is not callable outside the kernel. The exported functions are explained further in the subsequent subsections.

- CiInitialize()
- CiGetPEInformation()
- CiVerifyHashInCatalog()
- CiCheckSignedFile()
- CiFindPageHashesInCatalog()
- CiFindPageHashesInSignedFile()
- CiFreePolicyInfo()

The following functions are not exported, but are accessed via a callback structure provided by the CiInitialize() function. These functions are also explained in subsequent subsections.

- CiValidateImageHeader()
- CiValidateImageData()
- CiQueryInformation()
- CiQueryImageSignature()
- CiImportRoots()
- CiGetFileCache()
- CiSetFileCache()
- CiHashMemorySha256()

4.1.1 CiInitialize()

CiInitialize() is the function exported by Code Integrity for initializing the image file integrity validation capability of Code Integrity.

As the power-on (startup) function of Code Integrity, CiInitialize() conducts the following power-on (startup) self-tests.

- SHS (SHA-1) Known Answer Test
- SHS (SHA-256) Known Answer Test
- SHS (SHA-512) Known Answer Test
- RSA verify using a verify test with a Known Signatures of the PKCS#1 v1.5 format:
  - RSA signature with 1024-bit key and SHA-1 message digest
  - RSA signature with 2048-bit key and SHA-256 message digest

If a self-test fails, CiInitialize() returns STATUS_INVALID_IMAGE_HASH. On the other hand, after the successful initialization, CiInitialize() returns a callback structure consisting of the following functions. A caller subsequently can use these functions to obtain the image file integrity validation service from Code Integrity.

- CiValidateImageHeader()

© 2017 Microsoft. All Rights Reserved

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

- CiValidateImageData()
- CiQueryInformation()
- CiQueryImageSignature()
- CiImportRoots()
- CiGetFileCache()
- CiSetFileCache()
- CiHashMemorySha256()

4.1.1.1 CiValidateImageHeader()

When a caller (such as the Memory Manager) wants to obtain the set of trusted per-page hashes of an image file, it calls CiValidateImageHeader(). Trusted per-page hashes can use the following algorithms:

- SHS (SHA-1)
- SHS (SHA-256)
- SHS (SHA-384)
- SHS (SHA-512)

In the case of a Windows 8.1 OEs general purpose cryptographic module (namely, bcryptprimitives.dll, rsaenh.dll, or dssenh.dll), if CiValidateImageHeader() does not find the set of trusted per-page hashes for the cryptographic module, then CiValidateImageHeader() verifies the full cryptographic module image by verifying a trusted file hash. The trusted file hash may be:

- SHS (SHA-1)
- SHS (SHA-256)
- SHS (SHA-384)
- SHS (SHA-512)

If this validation process fails, the cryptographic module is not valid. Subsequently, the Windows 8.1 OEs Memory Manager does not load any page of the cryptographic module.

Both the trusted file image hash and trusted page hashes are signed using the RSA signature algorithm with PKCS#1 v1.5 padding.

Code Integrity has a different verification procedure for kernel mode crypto modules that are loaded into memory all at once (not in a per-page fashion as the other user mode general purpose crypto modules). As a result, when CiValidateImageHeader() is called by the memory manager, the CI_VALIDATE_DRIVER_IMAGE flag is set, and the entire image is validated by verifying a trusted image hash. This is similar to the user mode module verification when page hashes are not present.

4.1.1.2 CiValidateImageData()

After calling CiValidateImageHeader() to obtain the set of trusted per-page hashes of an image file, a caller (such as the Memory Manager) would want to know the integrity of a page of the image file. The caller uses CiValidateImageData() to check the page integrity.

© 2017 Microsoft. All Rights Reserved

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

If the computed hash matches the identified trusted hash, then CiValidateImageData confirms the integrity of the page. Otherwise, CiValidateImageData() returns STATUS_INVALID_IMAGE_HASH. The Windows 8.1 OEs Enterprise Memory Manager does not load invalid pages.

4.1.1.3 CiQueryInformation()

Returns state data about the enforcement of Code Integrity. Whether CI is being enforced and whether test signing is enabled.

4.1.1.4 CiQueryImageSignature()

Returns whether a previously validated file is Windows signed (signing certificate chains to Microsoft Root and the Windows EKU). This check was done during a previous validation, and this function is just returning a cached result.

4.1.1.5 CiImportRoots()

Imports public keys that are used as trusted CAs for validation of user mode components.

4.1.1.6 CiGetFileCache()

For an input file, returns the previously validated signature level (MSFT, Windows, Authenticode) and the thumbprint of the signing certificate. This check was done during a previous validation, and this function is just returning a cached result.

4.1.1.7 CiSetFileCache()

For a verified file, saves the signature level and thumbprint of the signing certificate. If the file was not previously verified, it will verify the file against either its embedded signature or a system catalog.

4.1.1.8 CiHashMemorySha256()

Passes supplied data to CI's SHA256 implementation and returns the SHA256 hash of that data.

4.1.2 CiGetPEInformation

Creates an encrypted channel between the caller and CI. It is used as part of protected media path DRM, and allows information about kernel drivers and user mode binaries loaded into protected processes to be returned to the caller.

4.1.3 CiVerifyHashInCatalog

For an input Authenticode file digest, validates that the digest is contained within a verified system catalog. It optionally returns information about the catalog.

4.1.4 CiCheckSignedFile

For an input Authenticode file digest and an Authenticode signature, verifies that the digest is in the signature and that the signature validates. It optionally returns information about the signature.

4.1.5 CiFindPageHashesInCatalog

For an input Authenticode digest of the first page of a PE image, validates that the digest is contained within a verified system catalog. It optionally returns information about the catalog.

© 2017 Microsoft. All Rights Reserved

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

4.1.6 CiFindPageHashesInSignedFile

For an input Authenticode digest of the first page of a PE image and an Authenticode signature, verifies that the digest is in the signature and that the signature validates. It optionally returns information about the signature.

4.1.7 CiFreePolicyInfo

Frees memory allocated by the CiVerifyHashInCatalog(), CiCheckSignedFile(), CiFindPageHashesInCatalog(), and CiFindPageHashesInSignedFile() functions.

4.2 Control Input Interface

The Control Input Interface for Code Integrity consists of the three CI export functions. Options for control operations are passed as input parameters to the CI export functions. The SecureRequired parameter in CiValidateImageHeader() is the only control option provided by Code Integrity in the Control Input Interface.

4.3 Status Output Interface

The Status Output Interface for Code Integrity also consists of the three CI export functions. For each function, the status information is returned to the caller as the return value (e.g. STATUS_SUCCESS, STATUS_UNSUCCESSFUL, STATUS_INVALID_IMAGE_HASH) from the function.

4.4 Data Input Interface

The Data Input Interface for Code Integrity also consists of the three CI export functions. Data and options are passed to the interface as input parameters to the CI export functions. Data Input is kept separate from Control Input by passing Data Input in separate parameters from Control Input.

4.5 Data Output Interface

The Data Output Interface for Code Integrity also consists of the three CI export functions. For CiInitialize(), data is returned to its caller as the Callbacks output parameter. For CiValidateImageHeader(), data is returned to its caller as the SePool output parameter.

5 Specification of Roles

Code Integrity supports both User and Cryptographic Officer roles (as defined in FIPS 140-2). Both roles have access to all services implemented in Code Integrity through a caller component running in the kernel mode. The module does not provide authentication, as such both roles are implicitly assumed when the services exported by the module are invoked.

5.1 Maintenance Roles

Maintenance roles are not supported.

5.2 Multiple Concurrent Interactive Operators

There is only one interactive operator during a logon session. Multiple concurrent interactive operators sharing a logon session are not supported.

© 2017 Microsoft. All Rights Reserved

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

6 Services

Code Integrity's services are:

1. Verify the integrity of binary executable code.
2. Provide Show Status services that indicate whether the integrity checks passed.
3. Provide Self-Test services.
4. Legacy certificate chain authentication (non-FIPS Approved service)

Code Integrity does not offer any other services, operations, or functions that can be externally invoked. Code Integrity export functions are only available inside the kernel. The User and Cryptographic Officer roles are not able to invoke them directly.

The following table maps the services to their corresponding algorithms and critical security parameters (CSPs). All the Code Integrity export functions in section 4.1 map to the service to verify the integrity of binary executable code.

Service / Function	Algorithms	CSPs	Invocation
Verify the integrity of binary executable code	FIPS 186-4 RSA PKCS#1 (v1.5) verify with public key FIPS 180-4 SHS: SHA-1 hash SHA-256 hash SHA-384 hash SHA-512 hash	Asymmetric Public keys	This service is fully automatic. The User / Cryptographic Officer does not take any actions to explicitly start this service. This service is executed whenever a binary executable is loaded.
Provide Show Status services that indicate whether the integrity checks passed	None	None	This service is fully automatic. The User / Cryptographic Officer does not take any actions to explicitly start this service. This service is executed upon completion of an integrity check function.
Provide Self-Test services	FIPS 186-4 RSA PKCS#1 (v1.5) verify with public key and known signature FIPS 180-4 SHS: SHA-1 KAT SHA-256 KAT SHA-512 KAT	None	This service is fully automatic. The User / Cryptographic Officer does not take any actions to explicitly start this service. This service is executed upon startup of this module.
Legacy certificate chain authentication (non-FIPS approved service)	MD5 (non-FIPS approved algorithm)	None	This service is fully automatic. The User / Cryptographic Officer does not take any actions to explicitly start this service. This service is executed whenever a binary executable with a legacy MD5 certificate is loaded.
CiGetPEInformation (non-FIPS approved function)	AES (non-FIPS approved algorithm)	None	This function is called as part of protected media path DRM operations. The User /

© 2017 Microsoft. All Rights Reserved

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

			Cryptographic Officer does not take any actions to explicitly invoke it.

The following table maps services and the export functions from Section 4 Ports and Interfaces.

Service	Export Functions
Verify the integrity of binary executable code	CiInitialize() CiValidateImageHeader() CiValidateImageData() CiQueryImageSignature() CiImportRoots() CiGetFileCache() CiSetFileCache() CiHashMemorySha256() CiVerifyHashInCatalog() CiCheckSignedFile() CiFindPageHashesInCatalog() CiFindPageHashesInSignedFile() CiFreePolicyInfo()
Provide Show Status services that indicate whether the integrity checks passed	CiInitialize() CiValidateImageHeader() CiValidateImageData() CiQueryImageSignature() CiImportRoots() CiGetFileCache() CiSetFileCache() CiHashMemorySha256() CiVerifyHashInCatalog() CiCheckSignedFile() CiFindPageHashesInCatalog() CiFindPageHashesInSignedFile() CiFreePolicyInfo()
Provide Self-Test services	CiInitialize()
Legacy certificate chain authentication (non-FIPS approved service)	CiInitialize() CiValidateImageHeader() CiValidateImageData() CiQueryImageSignature() CiImportRoots() CiGetFileCache() CiSetFileCache() CiVerifyHashInCatalog() CiCheckSignedFile() CiFindPageHashesInCatalog() CiFindPageHashesInSignedFile() CiFreePolicyInfo()

© 2017 Microsoft. All Rights Reserved

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

There is also an export function that does not map to any service: CiQueryInformation().

6.1 Show Status Services

The User and Cryptographic Officer roles have the same Show Status functionality, which is, for each function, the status information is returned to the caller as the return value from the function.

6.2 Self-Test Services

The User and Cryptographic Officer roles have the same Self-Test functionality, which is described in Section 10 Self-Tests.

6.3 Service Inputs / Outputs

The User and Cryptographic Officer roles have service inputs and outputs as specified in Section 0 Ports and Interfaces.

7 Operational Environment

The operational environment for Code Integrity is the Windows 8.1 OEs running on the software and hardware configurations listed in Section 1.3 - Validated Platforms.

8 Authentication

Code Integrity does not implement any authentication services. The User and Cryptographic Officer roles are assumed implicitly by booting the Windows operating system. There are Code Integrity libraries that run before boot in Winload.exe and Winresume.exe. The CI.DLL is loaded in the kernel as part of the memory management path.

9 Cryptographic Key Management

Code Integrity does not handle security-relevant information such as secret and private cryptographic key, authentication data, nor any other protected information. Hence, there is no operation related to any of the below.

- Key generation
- Key output
- Key storage
- Key Zeroization

The only cryptographic keys the module supports are the RSA PKCS#1 public keys used to verify integrity. These public keys are accessible by both approved roles. Due to such simplicity, an access control policy table is not included in this document. The public keys are stored on the hard-drive. Zeroization is performed by deleting the Code Integrity module, ntph.cat, and ntpe.cat files.

© 2017 Microsoft. All Rights Reserved

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

9.1 Critical Security Parameters

The Code Integrity crypto module uses the following cryptographic keys:

Cryptographic Key	Key Description
Asymmetric Public keys	Keys used for RSA PKCS#1 (v1.5) verification

9.2 Access Control Policy

The Code Integrity crypto module does not contain CSPs that would require access controls.

10 Self-Tests

10.1 Power-On Self-Tests

Code Integrity performs the following power-on (startup) self-tests when a caller calls its CiInitialize():

- SHS (SHA-1) Known Answer Test
- SHS (SHA-256) Known Answer Test
- SHS (SHA-512) Known Answer Test
- RSA verify using a verify test with a Known Signature of the PKCS#1 v1.5 format with both 1024-bit keys with SHA1 digest and 2048-bit keys with SHA-256 digest.

The integrity of Code Integrity itself is protected by an RSA signature with a 2048-bit key and SHA-256 message digest, which is verified by Winload.exe before Code Integrity is loaded into memory. If the self-test fails, the module will not load and status will be returned. If the status is not STATUS_SUCCESS, then that is the indicator a self-test failed.

10.2 Conditional Self-Tests

Code Integrity does not perform conditional self-tests.

11 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 8.1 OEs. The various methods of delivery and installation for each product are listed in the following table.

Product	Delivery and Installation Method
Windows 8.1	• DVD • Pre-installed on the computer by OEM • Download that updates Windows 8
Windows Server 2012 R2	• DVD • Pre-installed on the computer by OEM

© 2017 Microsoft. All Rights Reserved

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

	• Download that updates Windows Server 2012
Windows Storage Server 2012 R2	• Pre-installed by the OEM (Third party)
Surface Pro 3, Surface Pro 2, Surface Pro, Surface 2, Surface	• Pre-installed by the OEM (Microsoft)
Windows RT 8.1	• Pre-installed on the device by OEM • Download that updates Windows RT
Windows Phone 8.1	• Pre-installed on the device by OEM or mobile network operator • Download that updates Windows 8
Windows Embedded 8.1 Industry Enterprise	• Pre-installed by the OEM (Third party)
StorSimple 8000 Series	• Pre-installed by the OEM (Third party)
Azure StorSimple Virtual Array Windows Server 2012 R2	• Pre-installed by Azure

After the operating system has been installed, it must be configured by enabling the "System cryptography: Use FIPS compliant algorithms for encryption, hashing, and signing" policy setting followed by restarting the system. This procedure is all the crypto officer and user behavior necessary for the secure operation of this cryptographic module.

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

The installed version of Windows 8.1 OEs must be verified to match the version that was validated. See Appendix A 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 for details on how to do this.

© 2017 Microsoft. All Rights Reserved

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

12 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

13 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

14 Additional Details

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

http://windows.microsoft.com

© 2017 Microsoft. All Rights Reserved

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

Code Integrity

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

http://technet.microsoft.com/en-us/library/cc750357.aspx

© 2017 Microsoft. All Rights Reserved

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

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

15.1 How to Verify Windows Versions

The installed version of Windows 8.1 OEs must be verified to match the version that was validated using one of the following methods:

1. The ver command
    a. From Start, open the Search charm.
    b. In the search field type "cmd" and press the Enter key.
    c. The command window will open with a "C:\>" prompt.
    d. At the prompt, type "ver" and press the Enter key.
    e. You should see the answer "`Microsoft Windows [Version 6.3.9600]`".
2. The systeminfo command
    a. From Start, open the Search charm.
    b. In the search field type "cmd" and press the Enter key.
    c. The command window will open with a "C:\>" prompt.
    d. At the prompt, type "systeminfo" and press the Enter key.
    e. Wait for the information to be loaded by the tool.
    f. Near the top of the output, you should see:

```
OS Name:                   Microsoft Windows 8.1 Enterprise
OS Version:                6.3.9600 N/A Build 9600
OS Manufacturer:           Microsoft Corporation
```

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

15.2 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: x.x.xxxx.xxxxx.
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.

© 2017 Microsoft. All Rights Reserved

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