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	1.2
Updated On	September 8, 2023

© 2023 Microsoft Corporation. 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.

© 2023 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
1.0	November 4, 2020	Draft sent to NIST CMVP
1.1	November 3, 2022	Updates in response to NIST feedback
1.2	September 8, 2023	Updates in response to NIST feedback, updated bounded module certificates

© 2023 Microsoft Corporation. All Rights Reserved

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

TABLE OF CONTENTS

SECURITY POLICY DOCUMENT 1

VERSION HISTORY 3

1 INTRODUCTION 6

1.1 LIST OF CRYPTOGRAPHIC MODULE BINARY EXECUTABLES 6
1.2 VALIDATED PLATFORMS 6

2 CRYPTOGRAPHIC MODULE SPECIFICATION 8

2.1 CRYPTOGRAPHIC BOUNDARY 8
2.2 FIPS 140-2 APPROVED ALGORITHMS 8
2.3 NON-APPROVED ALGORITHMS 9
2.4 FIPS 140-2 APPROVED ALGORITHMS FROM BOUNDED MODULES 9
2.5 CRYPTOGRAPHIC BYPASS 9
2.6 HARDWARE COMPONENTS OF THE CRYPTOGRAPHIC MODULE 9

3 CRYPTOGRAPHIC MODULE PORTS AND INTERFACES 10

3.1 SKCI EXPORT FUNCTIONS 10
3.1.1 SKCIINITIALIZE 10
3.1.2 SKCICREATECODECATALOG 11
3.1.3 SKCICREATESECUREIMAGE 11
3.1.4 SKCIVALIDATEIMAGEDATA 11
3.1.5 SKCIVALIDATEDYNAMICCODEPAGES 11
3.1.6 SKCIFINALIZESECUREIMAGEHASH 11
3.1.7 SKCIFINISHIMAGEVALIDATION 11
3.1.8 SKCIFREEIMAGECONTEXT 11
3.1.9 SKCITRANSFERVERSIONRESOURCE 11
3.1.10 SKCIMATCHHOTPATCH 12
3.2 CONTROL INPUT INTERFACE 12
3.3 STATUS OUTPUT INTERFACE 12
3.4 DATA INPUT INTERFACE 12
3.5 DATA OUTPUT INTERFACE 12

4 ROLES, SERVICES AND AUTHENTICATION 12

© 2023 Microsoft Corporation. All Rights Reserved

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

4.1 ROLES 12
4.2 SERVICES 12
4.3 AUTHENTICATION 13

5 FINITE STATE MODEL 14

5.1 SPECIFICATION 14

6 OPERATIONAL ENVIRONMENT 14

6.1 SINGLE OPERATOR 14
6.2 CRYPTOGRAPHIC ISOLATION 15
6.3 INTEGRITY CHAIN OF TRUST 15

7 CRYPTOGRAPHIC KEY MANAGEMENT 17

8 SELF-TESTS 17

9 DESIGN ASSURANCE 17

10 MITIGATION OF OTHER ATTACKS 18

11 SECURITY LEVELS 19

12 ADDITIONAL DETAILS 19

13 APPENDIX A – HOW TO VERIFY WINDOWS VERSIONS AND DIGITAL SIGNATURES 20

13.1 HOW TO VERIFY WINDOWS VERSIONS 20
13.2 HOW TO VERIFY WINDOWS DIGITAL SIGNATURES 20

© 2023 Microsoft Corporation. All Rights Reserved

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

1 Introduction

Secure Kernel Code Integrity (SKCI) is a code integrity mechanism that runs in the Virtual Secure Mode (VSM) of the Windows Hyper-V hypervisor. SKCI is implemented in a Dynamic Link Library (DLL) file, SKCI.DLL.

Code Integrity and Secure Kernel Code Integrity are closely related modules that are used, depending on configuration of Windows, to validate system and application binaries. For the purpose of this validation, Secure Kernel Code Integrity is classified as a Software cryptographic module.

Two Windows configuration options dictate whether Code Integrity or Secure Kernel Code Integrity are used to verify a binary image:

- Virtual Secure Mode (VSM), also known as Core Isolation: Windows can use the Hypervisor to start an execution environment, called the Secure Kernel, that can enforce additional security rules. When VSM is configured, Secure Kernel Code Integrity verifies the integrity of critical user-mode modules such as BCRYPTPRIMITIVES.DLL instead of the Code Integrity module.
- Hypervisor Code Integrity (HVCI) , also known as Memory Integrity: This feature depends on VSM. When enabled, all drivers loaded into the Windows kernel are integrity verified by Secure Kernel Code Integrity.

This Security Policy Document assumes that the following hardware prerequisites are available:

- UEFI Secure Boot is available and enabled
- Trusted Platform Module (TPM)
- Hardware virtualization support (VT-x or AMD-V)

Additionally, VSM must be configured for SKCI.DLL to be loaded and used.

1.1 List of Cryptographic Module Binary Executables

Secure Kernel Code Integrity cryptographic module contains the following binary:

- skci.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

SKCI was validated using the combination of computers and Windows operating system editions specified in the table below.

© 2023 Microsoft Corporation. All Rights Reserved

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

All the computers for Windows Server listed in the table below are all 64-bit Intel architecture.

Table 1 Validated Platforms

Computer	Windows Server 2019 Datacenter Core	Processor Image
Microsoft Azure Stack Edge - Dell XR2 - Intel Xeon Silver 4114	√	2nd Gen Intel® Xeon® Scalable Processor wikichip.org
Microsoft Azure Stack Hub - Dell PowerEdge R640 - Intel Xeon Gold 6230	√	2nd Gen Intel® Xeon® Scalable Processor wikichip.org
Microsoft Azure Stack Hub - Dell PowerEdge R840 - Intel Xeon Platinum 8260	√	2nd Gen Intel® Xeon® Scalable Processor wikichip.org
Microsoft Azure Stack Edge Rugged - Rugged Mobile Appliance – Intel Xeon D-1559	√	INTEL® XEON® Processor D Product Family wikichip.org

© 2023 Microsoft Corporation. All Rights Reserved

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

2 Cryptographic Module Specification

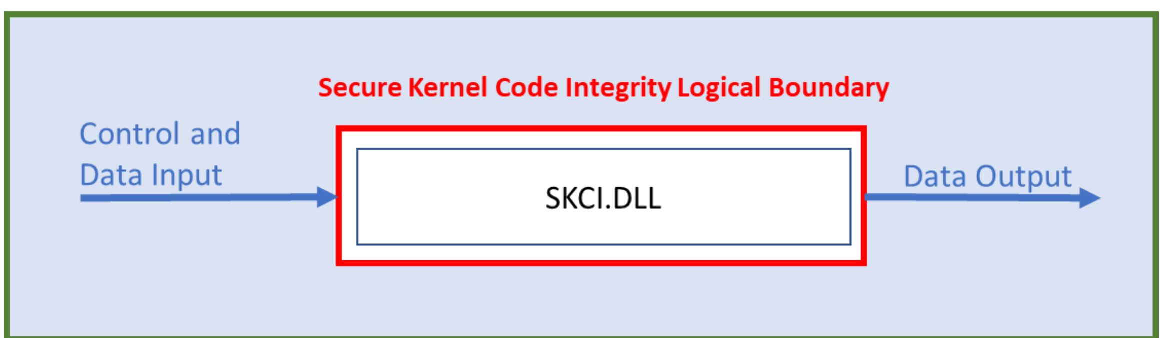
Secure Kernel Code Integrity is a multi-chip standalone module that operates in FIPS-approved mode during normal operation of the computer and Windows operating system.

The following configurations and modes of operation will cause Secure Kernel Code Integrity 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 binary that comprises the cryptographic boundary for Secure Kernel Code Integrity is SKCI.DLL.



2.2 FIPS 140-2 Approved Algorithms

SKCI implements the following FIPS 140-2 Approved algorithms:[1]

Table 2 Approved Algorithms

Algorithm	Windows Server 2019 build 10.0.17763.10021	Windows Server 2019 build 10.0.17763.10127
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	#C1577	#C2044
FIPS 180-4 SHS SHA-1, SHA-256, SHA-384, and SHA-512	#C1577	#C2044

[1] This module may not use some of the capabilities described in each CAVP certificate.

© 2023 Microsoft Corporation. All Rights Reserved

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

2.3 Non-Approved Algorithms

Secure Kernel Code Integrity only implements approved algorithms.

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, Secure Kernel Code Integrity depends on the following modules and algorithms:

The Windows OS Loader for Windows Server version 1809 build 10.0.17763.10021 (module certificate #4545) provides:

- CAVP certificate #C1586 (Windows Server) for FIPS 186-4 RSA PKCS#1 (v1.5) digital signature verification with 2048 moduli; supporting SHA-256
- CAVP certificate #C1577 (Windows Server) for FIPS 180-4 SHS SHA-256

The Windows OS Loader for Windows Server version 1809 build 10.0.17763.10127 (module certificate #4545 provides:

- CAVP certificate #C2052 (Windows Server) for FIPS 186-4 RSA PKCS#1 (v1.5) digital signature verification with 2048 moduli; supporting SHA-256
- CAVP certificate #C2044 (Windows Server) for FIPS 180-4 SHS SHA-256

Note that the validated platforms listed in section 1.2 include processors that support AES-NI. This module does not implement AES, but the bounded modules may implement AES and, therefore, use AES-NI.

2.5 Cryptographic Bypass

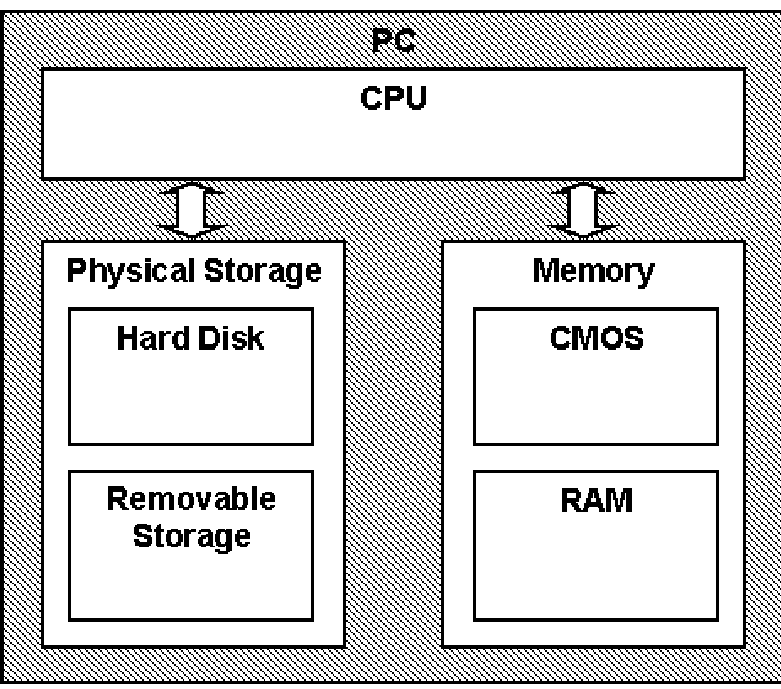
Cryptographic bypass is not supported by SKCI.

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 Secure Kernel Code Integrity module:

© 2023 Microsoft Corporation. All Rights Reserved

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



3 Cryptographic Module Ports and Interfaces

3.1 SKCI Export Functions

The following list contains all the functions exported by SKCI that are imported by the Secure Kernel. Note that SKCI is not callable outside the Secure Kernel. These functions are also explained in the subsequent subsections.

- SkciInitialize
- SkciCreateCodeCatalog
- SkciCreateSecureImage
- SkciValidateImageData
- SkciValidateDynamicCodePages
- SkciFinalizeSecureImageHash
- SkciFinishImageValidation
- SkciFreeImageContext
- SkciTransferVersionResource
- SkciMatchHotPatch

3.1.1 SkciInitialize

SkciInitialize is the function exported by SKCI for initializing the Secure Kernel version of Code Integrity. During this call, SKCI will get its configuration data from the Secure Kernel loader.

© 2023 Microsoft Corporation. All Rights Reserved
This Security Policy is non-proprietary and may be reproduced only in its original entirety (without revision).

See Self-Tests for information regarding cryptographic self-tests.

If a self-test fails, SkciInitialize returns STATUS_INVALID_IMAGE_HASH.

3.1.2 SkciCreateCodeCatalog

This function is called to create a code catalog object. The specified address range corresponds to a secure allocation object. It returns a catalog object. The secure allocation must be freed by SKCI when the catalog object is deleted.

3.1.3 SkciCreateSecureImage

This function is called when a new secure image section is created. It creates a context for validating an image. The caller specifies the type of hash algorithm that should be used to validate the image. It returns a pointer to the validation context, which is a state block.

3.1.4 SkciValidateImageData

This function is called to validate image data. When called for a file-hashed file that is still in the loading state, it is expected to generate the contents of page hashes. When in this mode, it will return STATUS_SUCCESS upon success. When page hashes are no-longer being generated and instead, page hashes have been used to verify the supplied pages, STATUS_VALID_IMAGE_HASH will be returned upon success.

3.1.5 SkciValidateDynamicCodePages

This function is called to validate dynamic code pages that were not part of a signed image.

3.1.6 SkciFinalizeSecureImageHash

This function is called to finalize (complete) the hash of a secure image. It returns the file or page hash of the image.

3.1.7 SkciFinishImageValidation

This function is called when initial validation of the image is complete. It completes the image validation process. The function is responsible to verify that the contents of the image header and/or file hash are correct, and, if successful, should update the image state to enable subsequent validation using page hashes. It is responsible for verifying that the data is verified by the page hashes for the resource section only. It returns information about the signing level; how the image is signed; the catalog ID used to validate the image; the algorithm with which a hash must be recalculated, if necessary; and the type of image the pages may be mapped into.

3.1.8 SkciFreeImageContext

This function is called when a secure image is unloaded and the context is to be freed.

3.1.9 SkciTransferVersionResource

This function is called to process the supplied version resource for an image, so that version data can be used during SkciFinishImageValidation.

© 2023 Microsoft Corporation. All Rights Reserved

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

3.1.10 SkciMatchHotPatch

This routine compares an image hash with the CI data embedded in a hot patch to determine whether the image matches the expected hash.

3.2 Control Input Interface

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

3.3 Status Output Interface

The Status Output Interface for SKCI consists of the exported functions listed in SKCI Export Functions. The status information is returned to the caller as the return value of each function (e.g. STATUS_SUCCESS, STATUS_UNSUCCESSFUL, STATUS_INVALID_IMAGE_HASH).

3.4 Data Input Interface

The Data Input Interface for SKCI consists of the exported functions listed in SKCI Export Functions. 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.

3.5 Data Output Interface

The Data Output Interface for SKCI also consists of the export functions listed in SKCI Export Functions with the exception of the initialization and status functions. Data is returned to the function's caller via output parameters.

4 Roles, Services and Authentication

4.1 Roles

Secure Kernel Code Integrity is a library used solely by the Windows Secure Kernel and does not interact with the user through any service. The module's functions are fully automatic and not configurable. FIPS 140 validations define formal "User" and "Cryptographic Officer" roles. Both roles can use any Secure Kernel Code Integrity service.

4.2 Services

Secure Kernel Code Integrity's services are:

1. **Verify the integrity of binary executable code –** This service is called by Secure Windows Kernel to verify the integrity of digitally signed drivers and other critical binary components of the operating system.
2. **Show Status** – The module does not provide an explicit status interface. Operational status is indicated by successfully initializing the module using SkciInitialize and success status messages using the binary integrity verification functions.
3. **Self-Tests** - The module provides a power-up self-tests service that is automatically executed when the module is loaded into memory.

© 2023 Microsoft Corporation. All Rights Reserved

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

The following table maps the services to their corresponding algorithms, critical security parameters (CSPs), and how they are invoked.

Table 3 Services

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	RSA public key	This service is fully automatic. This service is executed whenever a binary executable is loaded.
Show Status	None	None	This service is fully automatic. This service is executed upon completion of an integrity check function.
Self-Tests	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 following table maps SKCI services and export functions.

Service	Export Functions
Verify the integrity of binary executable code	SkciCreateCodeCatalog SkciCreateSecureImage SkciValidateImageData SkciValidateDynamicCodePages SkciFinalizeSecureImageHash SkciFinishImageValidation SkciFreeImageContext SkciTransferVersionResource SkciMatchHotPatch
Show Status	SkciInitialize All exported functions
Self-Tests	SkciInitialize

4.3 Authentication

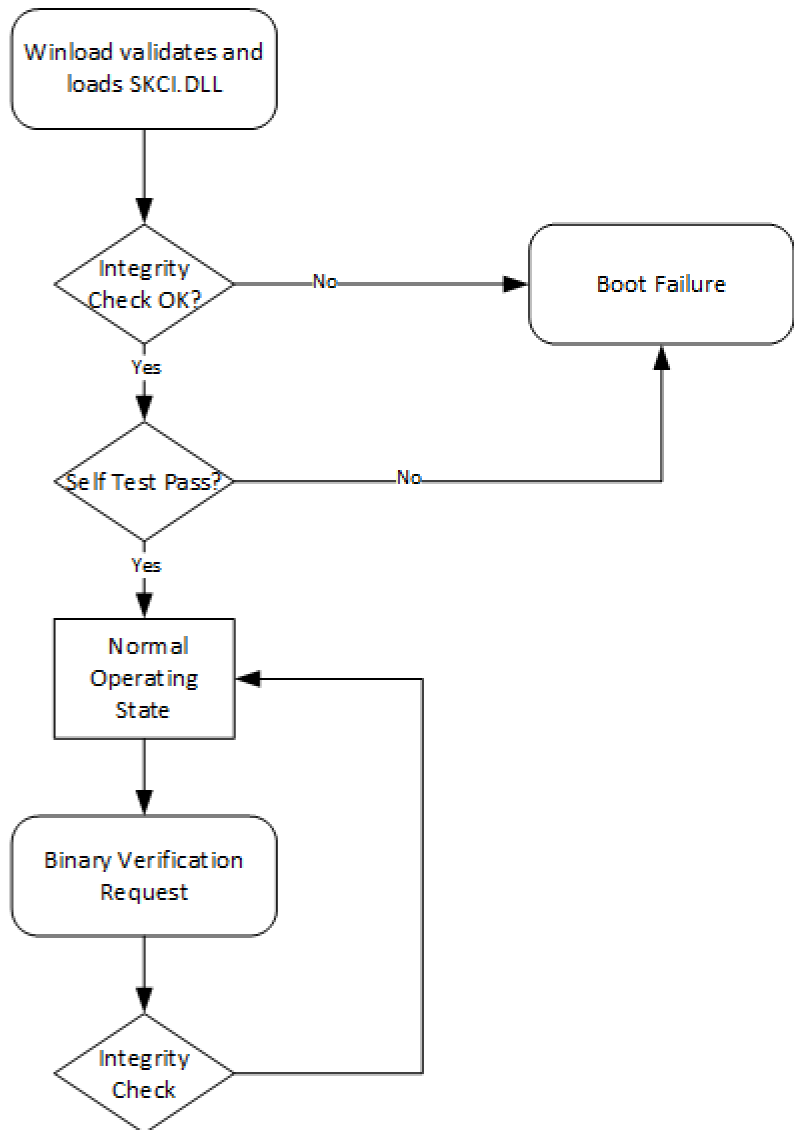
The module does not provide authentication. Roles are implicitly assumed based on the services that are executed.

© 2023 Microsoft Corporation. All Rights Reserved
This Security Policy is non-proprietary and may be reproduced only in its original entirety (without revision).

5 Finite State Model

5.1 Specification

The following diagram shows the finite state model for Secure Kernel Code Integrity:



6 Operational Environment

The operational environment for SKCI is the Windows Server operating system running on a supported hardware platform listed in section 1.2.

6.1 Single Operator

Secure Kernel Code Integrity is invoked by the Windows Secure Kernel as a fully automatic service with no user interaction.

© 2023 Microsoft Corporation. All Rights Reserved

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

6.2 Cryptographic Isolation

In the Windows operating system, all secure kernel-mode modules, including SKCI.DLL, are loaded into the Windows Secure Kernel which executes as a single process. The Windows operating system environment enforces process isolation from user-mode processes including memory and processor scheduling between the kernel and user-mode processes.

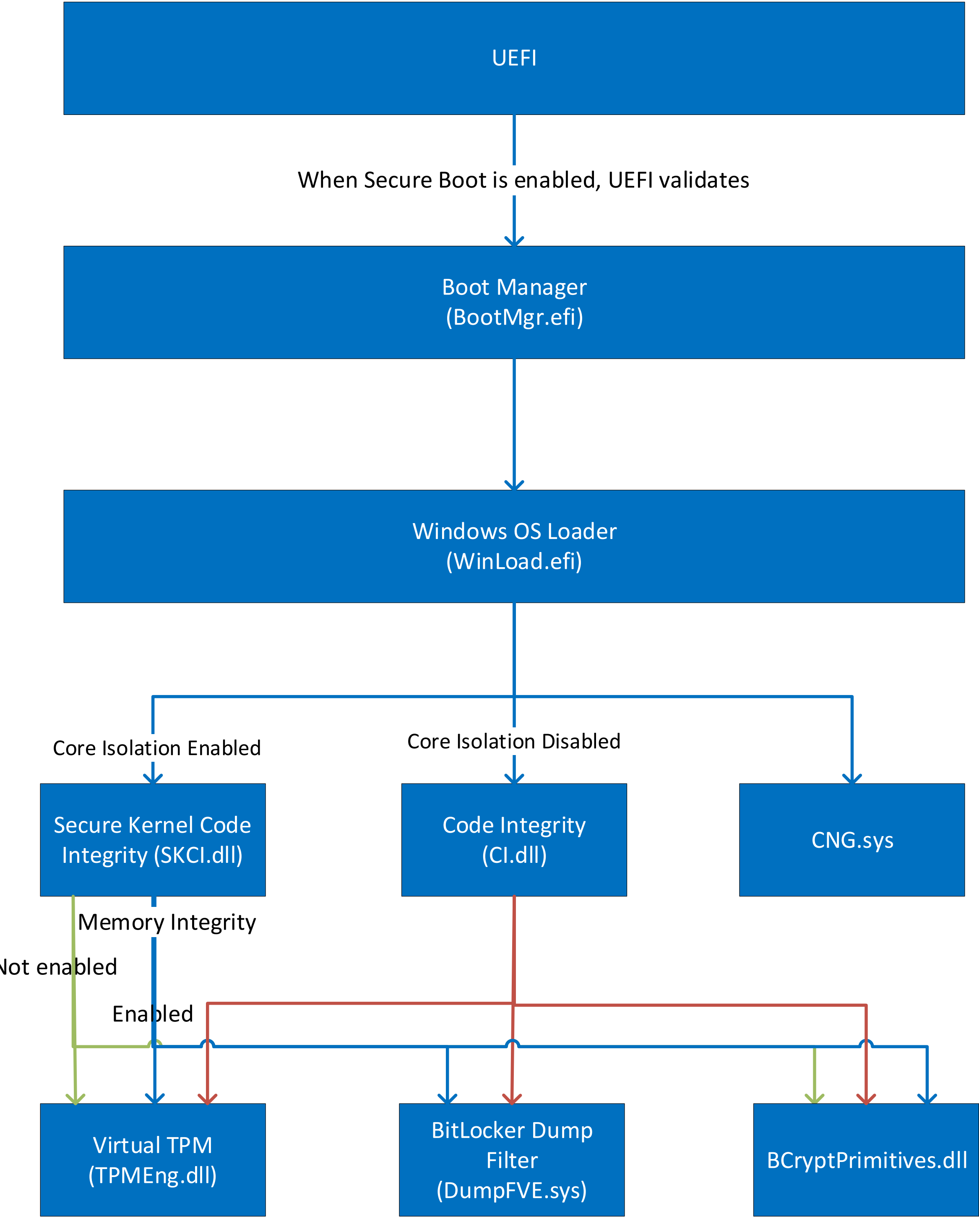
6.3 Integrity Chain of Trust

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

- Windows Server 2019 build 10.0.17763.10021 and 10.0.17763.10127

© 2023 Microsoft Corporation. All Rights Reserved

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



The integrity of the Secure Kernel Code Integrity module is checked by the Windows OS Loader. If VSM is enabled, then the Secure Kernel Code Integrity module is then invoked by the Code Integrity module to check the integrity of user mode binaries (including BCRYPTPRIMITIVES.DLL) as they are loaded. If HVCI

© 2023 Microsoft Corporation. All Rights Reserved

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

is enabled, the Secure Kernel Code Integrity module is then invoked by the Code Integrity module to check the integrity of kernel mode binaries (including DUMPFVE.SYS) as they are loaded.

Refer to the Introduction for information on the relationship between Code Integrity and Secure Kernel Code Integrity.

7 Cryptographic Key Management

Secure Kernel Code Integrity does not generate or store any persistent cryptographic keys; and uses the following RSA public key for validating file integrity.

- Microsoft Root Certificate Authority (CA) Public Key – 2048-bit RSA key with SHA-256.

8 Self-Tests

The Secure Kernel Code Integrity module implements Known Answer Test (KAT) functions each time the module is loaded by the Windows kernel.

Secure Kernel Code Integrity performs 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 Signature of the PKCS#1 v1.5 format with both 1024-bit keys with SHA1 digest and 2048-bit keys with SHA-256 digest.

If the self-test fails, the module will not load and status will be returned. If the status is STATUS_INVALID_IMAGE_HASH, then a self-test failed. Otherwise, STATUS_SUCCESS is 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 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 OEs must be verified to match the version that was validated. See Appendix A for details on how to do this.

© 2023 Microsoft Corporation. All Rights Reserved

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

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.

10 Mitigation of Other Attacks

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

Table 4 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

© 2023 Microsoft Corporation. All Rights Reserved

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

11 Security Levels

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

Table 5 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	NA
Operational Environment	1
Cryptographic Key Management	1
EMI/EMC	1
Self-Tests	1
Design Assurance	2
Mitigation of Other Attacks	1

12 Additional Details

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

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

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

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

© 2023 Microsoft Corporation. All Rights Reserved

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

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

13.1 How to Verify Windows Versions

The installed version of Windows Server OEs 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.

© 2023 Microsoft Corporation. All Rights Reserved

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