Microsoft Corporation

Code Integrity

FIPS 140-3 Non-Proprietary Security Policy Document

Microsoft Windows 11 version 22H2
(Pro, Enterprise, IoT Enterprise, Education, and Home Editions)

Microsoft Windows Server 2022
(Standard and Datacenter Editions)

Prepared By	Microsoft Corporation One Microsoft Way Redmond, WA 98052-6399
Document Version Number	1.0
Updated On	May 4, 2026

COPYRIGHT AND DISCLAIMER

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.

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

The example companies, organizations, products, people and events depicted herein are fictitious. No association with any real company, organization, product, person or event is intended or should be inferred.

Microsoft, Active Directory, Azure, Visual Basic, Visual Studio, Windows, the Windows logo, Windows NT, and Windows Server 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.

Table of Contents

List of Tables

List of Figures

1 General

1.1 Overview

The Code Integrity module (the “module”) is a multi-chip standalone software cryptographic module that verifies the integrity of Windows executable files as they are loaded into memory from storage. The module implements approved cryptographic algorithms. This FIPS 140-3 Security Policy contains a specification of the rules under which the module must operate and describes how the module meets the requirements specified in Federal Information Processing Standards Publication 140-3 (FIPS PUB 140-3) and International Standard ISO/IEC 19790:2012 (Information technology – Security techniques – Security requirements for cryptographic modules). This document is intended for the FIPS 140-3 testing lab, the Cryptographic Module Validation Program (CMVP), and administrators and users of the module.

1.2 Security Levels

The overall security rating for the module is level 1. The table below lists the security levels of individual clauses for this validation. As a software module executing in a modifiable environment, physical and non-invasive security requirements are optional and are out of scope for this validation.

Section	Title	Security Level
1	General	1
2	Cryptographic module specification	1
3	Cryptographic module interfaces	1
4	Roles, services, and authentication	1
5	Software/Firmware security	1
6	Operational environment	1
7	Physical security	N/A
8	Non-invasive security	N/A
9	Sensitive security parameter management	1
10	Self-tests	1
11	Life-cycle assurance	1
12	Mitigation of other attacks	1
	Overall Level	1

Table 1: Security Levels

2 Cryptographic Module Specification

2.1 Description

Purpose and Use:

The Code Integrity module is a multi-chip standalone software cryptographic module that verifies the integrity of Windows executable files as they are loaded into memory from storage. Code Integrity is implemented as a device driver named CI.DLL. Code Integrity is not a general-purpose cryptographic module. It is validated under FIPS 140 because it implements cryptographic algorithms and provides integrity checks for the Windows general-purpose cryptographic modules.

The Code Integrity module is closely related to the Secure Kernel Code Integrity module. Depending on the hardware and Windows configuration, Secure Kernel Code Integrity will also validate system and application binaries. The Secure Kernel Code Integrity module has its own Security Policy document.

Two Windows configuration options dictate whether Code Integrity or Secure Kernel Code Integrity is used to verify an executable:

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

Core Isolation and Memory Integrity are enabled by default when Windows runs on a Secured-core PC or a Secured-core server.

Module Type: Software

Module Embodiment: MultiChipStand

Cryptographic Boundary:

The software component of the module defines its cryptographic boundary and includes the single binary listed in the following table.

Software Component	Description
CI.DLL	Binary file that contains the module.

Table 2: Module Software Components

Tested Operational Environment's Physical Perimeter (TOEPP):

The Tested Operational Environment's Physical Perimeter (TOEPP) is the physical perimeter of the computer that contains the module.

The following block diagram illustrates the module's components, physical perimeter (TOEPP), and cryptographic boundary. The cryptographic boundary of the module is the module software component, CI.DLL. The CI.DLL binary is loaded from the OS volume in physical storage and executes in the computer memory. The control input, data input / output, and status output of the module exist within the computer memory as well.

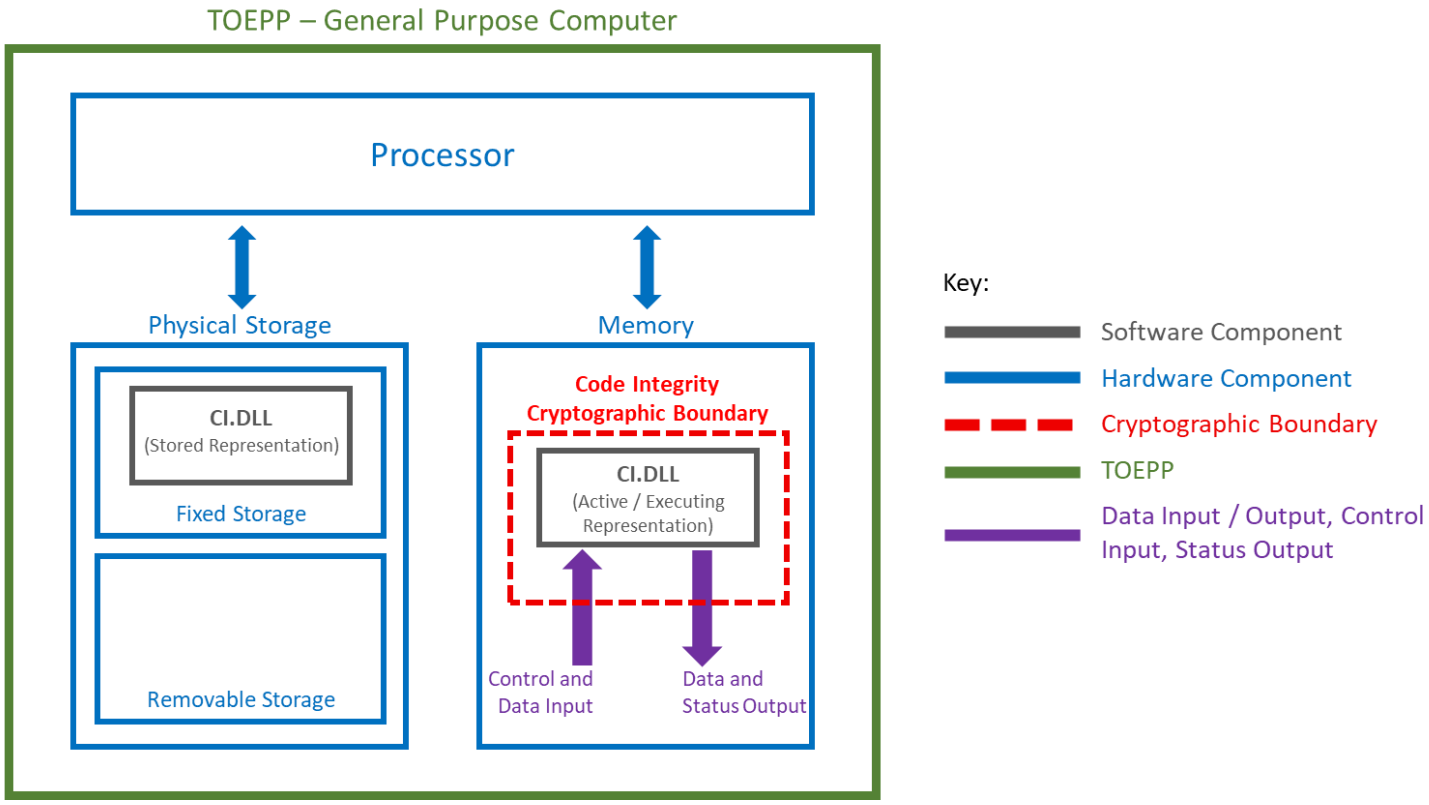

Figure 1: Hardware Components within the TOEPP

2.2 Tested and Vendor Affirmed Module Version and Identification

This validation includes the following Windows products and versions, each of which can be identified by its build number. The cryptographic module is a distinct implementation in each product build and for each processor architecture. Some Windows products may be installed as different editions; however, the cryptographic module is the same implementation in different editions of the same product.

Windows Product	Build	Edition(s) in Scope
Windows 11 version 22H2	10.0.22621.1	Enterprise Edition Home Edition IoT Enterprise Edition Pro Edition Education Edition
Windows Server 2022	10.0.20348.1668 (including the March 2023 updates)	Standard Edition Datacenter Edition

Table 3: Version Information

Tested Module Identification – Hardware:

N/A for this module.

Tested Module Identification – Software, Firmware, Hybrid (Executable Code Sets):

Package or File Name	Software/ Firmware Version	Features	Integrity Test
CI.DLL (Windows 11 version 22H2)	Windows 11 version 22H2, build 10.0.22621.1	N/A	Yes
CI.DLL (Windows Server 2022)	Windows Server 2022, build 10.0.20348.1668	N/A	Yes

Table 4: Tested Module Identification – Software, Firmware, Hybrid (Executable Code Sets)

Tested Module Identification – Hybrid Disjoint Hardware:

N/A for this module.

Tested Operational Environments - Software, Firmware, Hybrid:

The operational environment for the module is the Windows operating system running on a supported hardware platform, as listed in the table below. All hardware platforms in the table below are 64-bit Intel architecture. The tested operational environments provide Processor Algorithm Acceleration (PAA) in the form of the Advanced Encryption Standard New Instructions (AES-NI) but, as it does not apply to any of the algorithms used by the module, the PAA/PAI column below is marked as No.

Operating System	Hardware Platform	Processors	PAA/PAI	Hypervisor or Host OS	Version(s)
Windows 11 version 22H2, Education Edition	Dell Latitude 7420	11th Gen Intel Core i7-1185G7	No	N/A	Windows 11 version 22H2, build 10.0.22621.1
Windows 11 version 22H2, Enterprise Edition	Microsoft Surface Laptop 5	12th Gen Intel Core i7-1265U	No	N/A	Windows 11 version 22H2, build 10.0.22621.1
Windows 11 version 22H2, Home Edition	Microsoft Surface Laptop 5	12th Gen Intel Core i7-1265U	No	N/A	Windows 11 version 22H2, build 10.0.22621.1
Windows 11 version 22H2, IoT Enterprise Edition	Microsoft Surface Laptop 5	12th Gen Intel Core i7-1265U	No	N/A	Windows 11 version 22H2, build 10.0.22621.1
Windows 11 version 22H2, Pro Edition	HP ZBook Power G8	11th Gen Intel Core i5-11500H	No	N/A	Windows 11 version 22H2, build 10.0.22621.1
Windows Server 2022 Datacenter, including the March 2023 Updates	Dell PowerEdge R640	Intel Xeon Gold 6130	No	N/A	Windows Server 2022, build 10.0.20348.1668
Windows Server 2022 Standard, including the March 2023 Updates	Dell PowerEdge R640	Intel Xeon Gold 6130	No	N/A	Windows Server 2022, build 10.0.20348.1668

Table 5: Tested Operational Environments - Software, Firmware, Hybrid

Vendor-Affirmed Operational Environments - Software, Firmware, Hybrid:

Operating System	Hardware Platform
Any Microsoft operating system which is a relabeled version of the operating systems listed in Section 2.2.	Any UEFI-based x64 computer.

Table 6: Vendor-Affirmed Operational Environments - Software, Firmware, Hybrid

The CMVP makes no statement as to the correct operation of the module or the security strengths of the generated keys when so ported if the specific operational environment is not listed on the validation certificate.

2.3 Excluded Components

No components within the cryptographic boundary are claimed as excluded.

2.4 Modes of Operation

Modes List and Description:

Mode Name	Description	Type	Status Indicator
Approved Mode	Normal operation of the computer, Windows OS, and module. This is the only mode claimed by the module.	Approved	Successful completion of any of the module's services (see Section 3.4.1 Code Integrity Export Functions and Section 4.3 Approved Services).

Table 7: Modes List and Description

2.5 Algorithms

Approved Algorithms:

The tables below list the approved algorithms used in the module. The module may not use some of the capabilities described in each CAVP certificate. As the module has separate CAVP certificates for Windows 11 and Windows Server 2022, each approved algorithm is listed twice, with CAVP certificates #A3767 and #A4008 for Windows 11 and CAVP certificates #A3768 and #A4009 for Windows Server 2022. See Section 13 Standards References for links to the standards referenced in the tables below.

The Code Integrity cryptographic module relies on some functionality that is implemented in the Windows OS Loader cryptographic module. FIPS 140-3 deems the Code Integrity module as binding to the Windows OS Loader module (certificate #5405), which is referred to as an Existing Validated Module (EVM) in this document. All Windows cryptographic modules are installed together as described in section 11 Life-Cycle Assurance. See section 5.1 Integrity Techniques for more information on the dependencies between Windows modules.

Table 9 below lists the approved algorithms in the Code Integrity module. Table 8 below lists the approved cryptographic algorithms in the Windows OS Loader module that are used by the Code Integrity module.

Existing Validated Module [EVM]

Algorithm	CAVP Cert	Properties	Reference
RSA SigVer (FIPS186-4)	A3767	Signature Type - PKCS 1.5 Modulo - 2048	FIPS 186-4
RSA SigVer (FIPS186-4)	A3768	Signature Type - PKCS 1.5 Modulo - 2048	FIPS 186-4
SHA2-256	A4008	Message Length - Message Length: 0-65536 Increment 8	FIPS 180-4
SHA2-256	A4009	Message Length - Message Length: 0-65536 Increment 8	FIPS 180-4

Table 8: Approved Algorithms - Existing Validated Module [EVM]

Algorithm	CAVP Cert	Properties	Reference
RSA SigVer (FIPS186-4)	A4008	Signature Type - PKCS 1.5 Modulo - 1024, 2048, 3072, 4096	FIPS 186-4

Algorithm	CAVP Cert	Properties	Reference
RSA SigVer (FIPS186-4)	A4009	Signature Type - PKCS 1.5 Modulo - 1024, 2048, 3072, 4096	FIPS 186-4
SHA-1	A4008	Message Length - Message Length: 0-65536 Increment 8	FIPS 180-4
SHA-1	A4009	Message Length - Message Length: 0-65536 Increment 8	FIPS 180-4
SHA2-256	A4008	Message Length - Message Length: 0-65536 Increment 8	FIPS 180-4
SHA2-256	A4009	Message Length - Message Length: 0-65536 Increment 8	FIPS 180-4
SHA2-384	A4008	Message Length - Message Length: 0-65536 Increment 8	FIPS 180-4
SHA2-384	A4009	Message Length - Message Length: 0-65536 Increment 8	FIPS 180-4
SHA2-512	A4008	Message Length - Message Length: 0-65536 Increment 8	FIPS 180-4
SHA2-512	A4009	Message Length - Message Length: 0-65536 Increment 8	FIPS 180-4

Table 9: Approved Algorithms -

Vendor-Affirmed Algorithms:

N/A for this module.

Non-Approved, Allowed Algorithms:

N/A for this module.

Non-Approved, Allowed Algorithms with No Security Claimed:

N/A for this module.

Non-Approved, Not Allowed Algorithms:

N/A for this module.

2.6 Security Function Implementations

The table below lists the module's Security Function Implementations. The Algorithms column presents algorithm and key size information for each CAVP certificate listed in Section 2.5 Algorithms. Blank cells are either not applicable or optional according to the CMVP. As the module has separate CAVP certificates for Windows 11 and Windows Server 2022, each approved algorithm is listed twice in the Algorithms column.

Name	Type	Description	Properties	Algorithms
DigSig-Legacy	DigSig-SigVer	Legacy RSA signature verification used by the Verify the Integrity of Executable Code service to verify integrity of digitally signed drivers and other binary components of the OS. SHA secure hash functions support this by verifying the RSA signature.		RSA SigVer (FIPS186-4): (A4008, A4009) Modulus Size: 1024 bits SHA-1: (A4008, A4009) SHA Size: 160 bits
DigSig1	DigSig-SigVer	RSA signature verification used by the Verify the Integrity of Executable Code service to verify integrity of digitally signed drivers and other binary components of the OS. SHA secure hash functions support this by verifying the RSA signature.		RSA SigVer (FIPS186-4): (A4008, A4009) Modulus Size: 2048, 3072, and 4096 bits SHA2-256: (A4008, A4009) SHA Size: 256 bits SHA2-384: (A4008, A4009) SHA Size: 384 bits SHA2-512: (A4008, A4009) SHA Size: 512 bits
DigSig2	DigSig-SigVer	RSA signature verification used for the module pre-operational software integrity check only, executed by the [EVM] OS Loader module (IG 1.A Documentation Requirements 5). SHA secure hash functions support this by verifying the RSA signature.		RSA SigVer (FIPS186-4): (A3767, A3768) Modulus Size: 2048 bits SHA2-256: (A4008, A4009) SHA Size: 256 bits

Name	Type	Description	Properties	Algorithms
SHS1	SHA	Secure hash functions used by the Verify the Integrity of Executable Code service.		SHA2-256: (A4008, A4009) SHA Size: 256 bits SHA2-384: (A4008, A4009) SHA Size: 384 bits SHA2-512: (A4008, A4009) SHA Size: 512 bits
SHS2	SHA	Secure hash function used for the module pre-operational software integrity check only, executed by the [EVM] OS Loader module (IG 1.A Documentation Requirements 5).		SHA2-256: (A4008, A4009) SHA Size: 256 bits

Table 10: Security Function Implementations

2.7 Algorithm Specific Information

2.7.1 FIPS 186-4 and 186-5

The module claims compliance with FIPS 186-5 for RSA signature verification, although the CAVP testing for RSA was completed against FIPS 186-4. Because the FIPS 186-4 RSA CAVP tests are mathematically identical to the FIPS 186-5 RSA CAVP tests, the module can claim a FIPS 186-5 compliance for these tests. The scope of FIPS 186-4 testing complies with IG C.K. Additional Comment 3. Although FIPS 186-4 has been superseded by FIPS 186-5, algorithms implemented under FIPS 186-4 remain approved for use under NIST SP 800-131A Rev. 2.

2.7.2 Legacy Signature Verification using RSA and SHA-1

1. SHA-1 is used by the Verify the Integrity of Executable Code service for legacy signature verification only.

2. RSA with a 1024-bit key is used by the Verify the Integrity of Executable Code service for legacy signature verification only.

3. Algorithms designated as “Legacy” may only be used on data that was generated prior to the Legacy Date specified in FIPS 140-3 IG C.M.

2.8 RBG and Entropy

N/A for this module.

N/A for this module.

2.9 Key Generation

N/A because the module does not generate cryptographic keys.

2.10 Key Establishment

N/A because the module does not perform key establishment.

2.11 Industry Protocols

N/A as none are claimed.

3 Cryptographic Module Interfaces

3.1 Ports and Interfaces

As a software module, the module has no physical ports of its own. The physical ports of the module are interpreted as those on the underlying hardware platform and control of them is outside the scope of the module.

Physical Port	Logical Interface(s)	Data That Passes
N/A	Data Input	The Data Input Interface for the module is the set of exported and callable functions listed in section 3.4.1 Code Integrity Export Functions, with the exception of the initialization and status 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.
N/A	Data Output	The Data Output Interface for the module is the set of exported and callable functions listed in section 3.4.1 Code Integrity Export Functions, with the exception of the initialization and status functions. Data is returned to the function's caller via output parameters.
N/A	Control Input	The module Control Input Interface consists of the exported functions. Options for control operations are passed as input parameters to the export functions.
N/A	Status Output	The Status Output Interface for the module consists of the exported and callable functions listed in section 3.4.1 Code Integrity Export Functions. The status information is returned to the caller as the return value of each function (for example, STATUS_SUCCESS, STATUS_UNSUCCESSFUL, STATUS_INVALID_IMAGE_HASH).
N/A	Power	N/A

Table 11: Ports and Interfaces

3.4 Additional Information

3.4.1 Code Integrity Export Functions

The table below presents all the functions exported by the module to kernel-mode callers. The module is not callable outside the kernel.

Function	Description
CiInitialize	The function exported for initializing the image file integrity validation capability of the module. See section 10 Self-Tests for information regarding cryptographic self-tests. If the self-tests succeed, CiInitialize() returns a callback structure consisting of the binary executable file integrity validation functions, as listed in section 3.4.2 Code Integrity Callback Functions.
CiGetPEInformation	This function returns system configuration data which is related to the protected media subsystem.
CiVerifyHashInCatalog	For an input Authenticode file digest, this function validates that the digest is contained within a verified system catalog. It optionally returns information about the catalog.
CiCheckSignedFile	For an input Authenticode file digest and an Authenticode signature, this function verifies that the digest is in the signature and that the signature validates. It optionally returns information about the signature.
CiFindPageHashesInCatalog	For an input Authenticode digest of the first page of a PE image, this function validates that the digest is contained within a verified system catalog. It optionally returns information about the catalog.
CiFindPageHashesInSignedFile	For an input Authenticode digest of the first page of a PE image and an Authenticode signature, this function verifies that the digest is in the signature and that the signature validates. It optionally returns information about the signature.
CiFreePolicyInfo	This function frees memory allocated by the CiVerifyHashInCatalog, CiCheckSignedFile, CiFindPageHashesInCatalog, and CiFindPageHashesInSignedFile functions.

Function	Description
CiSetTrustedOriginClaimId	This function is invoked by Appid.sys when an App Control for Business policy is being processed.
CiValidateFileObject	This function verifies the signature of a file object and returns the policy info along with the timestamp and signing time.

Table 12: Functions Exported by the Module

3.4.2 Code Integrity Callback Functions

The functions listed and described in the following table are not exported but are accessed using a callback structure provided by the CiInitialize function.

Function	Description
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 (SHA2-256) • SHS (SHA2-384) • SHS (SHA2-512) 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 (SHA2-256) • SHS (SHA2-384) • SHS (SHA2-512) If this validation process fails, the module is not valid and the module is not loaded. Both the trusted file image hash and trusted page hashes are signed using the RSA signature algorithm with PKCS#1 v1.5 padding.
CiValidateImageData	After calling CiValidateImageHeader to obtain the set of trusted per-page hashes of an image file, CiValidateImageData() is used to check the integrity of each page by computing the hash value of the page. If the computed hash matches the identified trusted hash, then CiValidateImageData confirms the integrity of the page. Otherwise, CiValidateImageData returns STATUS_INVALID_IMAGE_HASH.
CiQueryInformation	This function returns state data about whether code integrity is being enforced and whether test signing is enabled.
CiSetFileCache	For a verified file, this function 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.
CiGetFileCache	For an input file, this function returns the previously validated signature level and the thumbprint of the signing certificate. This check was done during a previous validation, and this function is just returning a cached result.
CiHashMemory()	This function passes supplied data to MinCrypK_HashMemory and returns the hash of that data.

Function	Description
KappxIsPackageFile	This routine takes an input file object and parses out the full package name associated with the corresponding package. Package association is established based on the normalized path corresponding to the file object.
CiCompareSigningLevels	This routine determines if the source signing level is applicable for the target. For example, a source of Microsoft is not valid for a target of Windows, but vice-versa is valid.
CiValidateFileAsImageType	This routine determines if a PE file is signed appropriately for the specified image type. The file must be mapped as a view to a data section or be a copy of that view.
CiRegisterSigningInformation	This routine sets a signer, in addition to those already configured, for a specified signing level. Only those levels that are enabled for runtime configuration will accept signers supplied to this routine.
CiUnregisterSigningInformation	This routine removes a previous registered runtime signing information. Once a registration handle has been unregistered, it must be discarded.
CiInitializePolicy	This routine is called to get the configuration of CI for this boot and return the list of address ranges to be protected by Patch Guard.
CipQueryPolicyInformation	This routine returns information about the Code Integrity policy state.
CiValidateDynamicCodePages	This routine validates the contents of code pages generated dynamically.
SIPolicyQuerySecurityPolicy	This routine queries the secure setting for specific provider's <Key,Value> pair.
CiRevalidateImage	This routine determines if previously validated images must be validated again.
CiSetUnlockInformation	This function sets unlock information for Code Integrity.
CiGetBuildExpiryTime	This routine determines the expiry time of the build. Zero time means the build never expires, which is true for production and test builds. Flight builds expire when the certificate that signs CI expires.
CiGetStrongImageReference	Note – this function is only included in the callback structure if HVCI is enabled. This routine returns the handle to a secure image.
CiReleaseContext	Note – this function is only included in the callback structure if HVCI is enabled. This routine closes a validation context.
CiHvciSetImageBaseAddress	Note – this function is only included in the callback structure if HVCI is enabled. This routine changes the base address of an image that is using secure relocations.

Table 13: Functions Accessed Using Callback Structure Provided by the CiInitialize Function

4 Roles, Services, and Authentication

4.1 Authentication Methods

N/A for this module.

4.2 Roles

The module claims a single role, Cryptographic Officer (CO). All services are accessible by this role. The module is a library used solely by the Windows kernel and does not interact with the user through any service. The module's functions are fully automatic and not configurable.

Name	Type	Operator Type	Authentication Methods
Cryptographic Officer (CO)	Role	CO	None

Table 14: Roles

4.3 Approved Services

The module provides approved services only. The indicator for each service is the successful completion of the service. The table below provides additional indicator details to enable the operator to verify each service's successful completion.

Name	Description	Indicator	Inputs	Outputs	Security Functions	SSP Access
Perform Cryptographic Algorithm Self-Tests	The module provides a power-up self-test service that is automatically executed.	Self-test success is indicated by module and algorithm availability; failure is indicated by an error.	Executed when the CiInitialize() function is called and input parameters are provided.	If any self-test fails, the module will not load and a failure status, STATUS_INVALID_IMAGE_HASH, is returned and the computer will not complete the boot process. Otherwise, STATUS_SUCCESS is returned and the boot process completes.	DigSig1 DigSig-Legacy SHS1	Cryptographic Officer (CO)
Perform Pre-Operational Software Integrity Test [EVM]	The pre-operational integrity test is executed by the [EVM] OS Loader (IG 1.A Documentation Requirements 5).	Integrity test success is indicated by module and algorithm availability; failure is indicated by an error.	This service is fully automatic and executed before the Code Integrity module is loaded into memory.	The availability of the module is the service output. See section 10 Self-Tests for more information.	DigSig2 SHS2	Cryptographic Officer (CO)

Name	Description	Indicator	Inputs	Outputs	Security Functions	SSP Access
Perform Zeroization	Zeroizes cryptographic material.	SSPs are zeroized. See Section 9.3 SSP Zeroization Methods for more information.	Procedural zeroization executed by the user as described in section 9.3 Zeroization Methods	The SSPs in non-volatile storage are zeroized.	DigSig-Legacy DigSig1 DigSig2 SHS1 SHS2	Cryptographic Officer (CO) - Embedded RSA Signatures for Code Integrity to Evaluate : Z - Embedded X.509 Certificate for CI.DLL (This is not an SSP): Z - Hash Value for CI.DLL (This is not an SSP): Z - Hash Values for Code Integrity to Evaluate: Z
Show Status	Implicit in module initialization and status messages returned with function calls.	Operational status is indicated by successfully initializing the module using CiInitialize() and success status messages using the binary integrity verification functions.	Executed whenever an integrity check function is completed, specifically when one of the following functions is called and input parameters are provided - for more information on the functions, see 3.4 Additional Information: CiQueryInformation() CiInitialize() CiInitializePolicy() CipQueryPolicyInformation() CiRevalidateImage() All exported functions	Status is output across the module's Status Output logical interface.	None	Cryptographic Officer (CO)

Name	Description	Indicator	Inputs	Outputs	Security Functions	SSP Access
Show Version	Provides the module version number.	Version information is provided in the Portable Executable (PE) header of each module binary. The PE header contains Windows-specific fields such as the major and minor version (10.0.22621.1 for Windows 11 and 10.0.20348.1668 for Windows Server). See the PE Format public documentation published on https://learn.microsoft.com for more information..	Module binary file.	Version information is embedded in the PE header of the binary.	None	Cryptographic Officer (CO)

Name	Description	Indicator	Inputs	Outputs	Security Functions	SSP Access
Verify the Integrity of Executable Code	This service is called by Windows to verify the integrity of digitally signed device drivers and other critical executable components of the operating system.	Results of the integrity check are returned via the functions detailed in Section 3.4 Additional Information.	Executed whenever a binary executable is loaded, specifically when one of the following functions is called and input parameters are provided - for more information on the functions, see 3.4 Additional Information: CiGetPEInformation() CiVerifyHashInCatalog() CiCheckSignedFile() CiFindPageHashesInCatalog() CiFindPageHashesInSignedFile() CiFreePolicyInfo() CiSetTrustedOriginClaimId() CiValidateFileObject() CiValidateImageHeader() CiValidateImageData() CiSetFileCache() CiGetFileCache() CiHashMemory() KappxIsPackageFile() CiCompareSigningLevels() CiValidateFileAsImageType() CiRegisterSigningInformation() CiUnregisterSigningInformation() CiValidateDynamicCodePages() SIPolicyQuerySecurityPolicy() CiSetUnlockInformation() CiGetBuildExpiryTime() CiGetStrongImageReference() CiReleaseContext() CiHvciSetImageBaseAddress()	The cryptographic operation is completed, and status is returned.	DigSig1 DigSig-Legacy SHS1	Cryptographic Officer (CO) - Embedded RSA Signatures for Code Integrity to Evaluate : W,E - Hash Values for Code Integrity to Evaluate: W,E

Table 15: Approved Services

4.4 Non-Approved Services

N/A for this module.

4.5 External Software/Firmware Loaded

The module does not load external software or firmware.

5 Software/Firmware Security

The secure installation, generation, and startup procedures of this module are part of the secure installation, configuration, and startup procedures of the Windows operating systems named in Section 2.2 Tested and Vendor Affirmed Module Version and Identification.

In the Windows operating system, all kernel-mode modules, including CI.DLL, are loaded into the Windows Kernel (ntoskrnl.exe) 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.

5.1 Integrity Techniques

Windows uses several mechanisms to provide integrity verification depending on the stage in the boot sequence and also on the hardware and configuration.

The [EVM] Windows OS Loader module (IG 1.A Documentation Requirements 5) checks the integrity of the CI.DLL component of the Code Integrity module before it is loaded. The EVM uses approved RSA SigVer (FIPS 186-4, CAVP certificates A3767 and A3768) and SHA2-256 (FIPS 180-4, CAVP certificates A4008 and A4009) to perform the integrity check. To perform the module integrity test on demand, the user may restart the computer. See the table below for more information on the [EVM] Windows OS Loader module.

The Code Integrity module is then, in certain configurations, responsible for checking the integrity of every other user and kernel mode system executable as they are loaded. Refer to Section 11.1 Installation, Initialization, and Startup Procedures for more information.

The following table provides the details of the [EVM] Windows OS Loader (IG 1.A Documentation Requirements 5).

EVM Name	CMVP Certificate	Version Details
Windows OS Loader (Windows 11 v 22H2 and Windows Server 2022)	#5405	Windows 11 version 22H2, build 10.0.22621.1 Windows Server 2022, build 10.0.20348.1668

Table 16: EVM Details

The figure below shows the Integrity Chain of trust for the Windows builds and modules in scope for this validation.

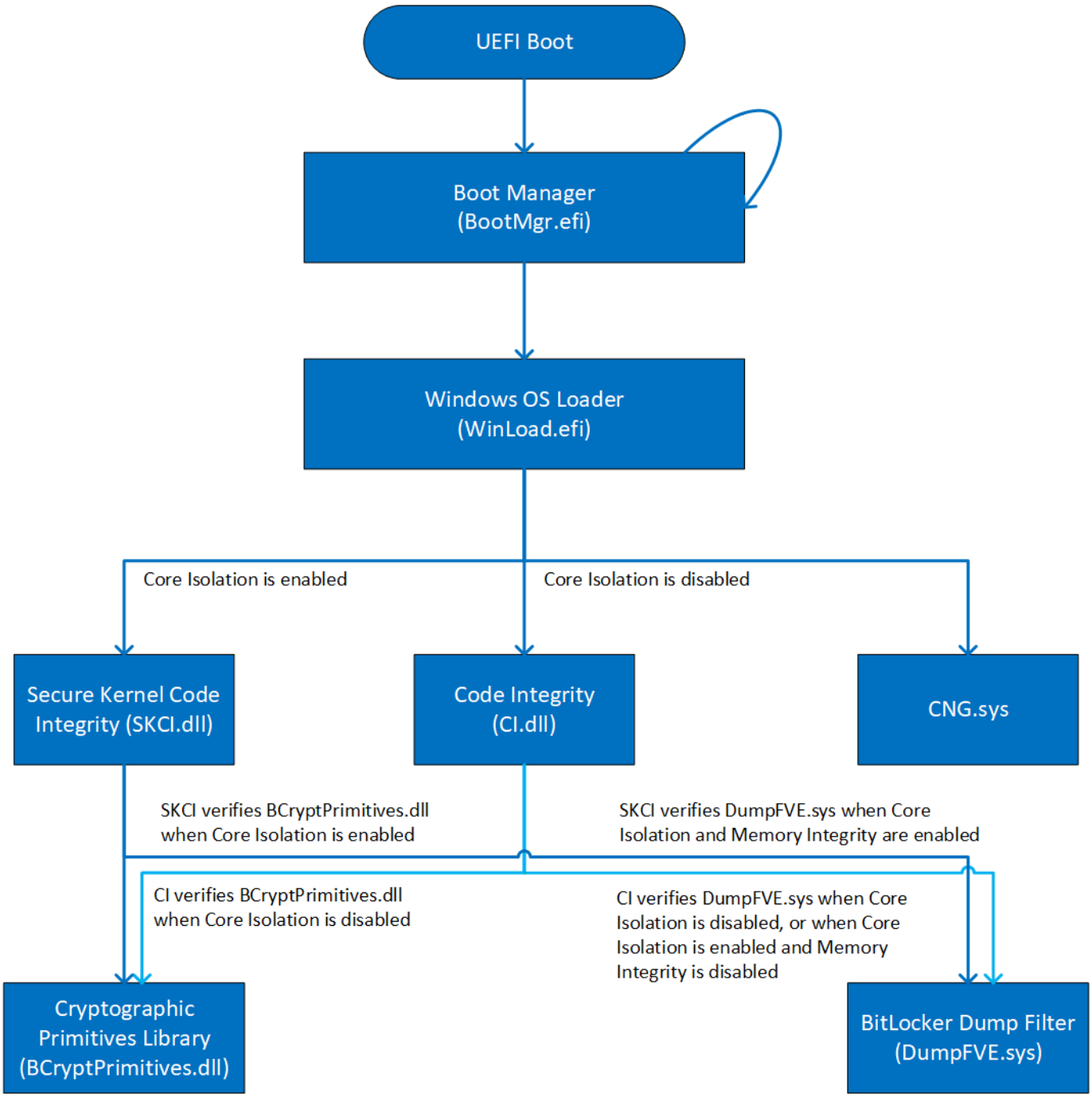

Figure 2: Integrity Chain of Trust

5.2 Initiate on Demand

To initiate the integrity test on demand, the operator may restart the computer.

6 Operational Environment

6.1 Operational Environment Type and Requirements

Type of Operational Environment: Modifiable

How Requirements are Satisfied:

The modifiable operational environment for the module is the Windows operating system running on a supported hardware platform, as listed in Section 2.2 Tested and Vendor Affirmed Module Version and Identification. The module is invoked by the Windows kernel as a fully automatic service with no user interaction.

7 Physical Security

N/A for this module.

8 Non-Invasive Security

N/A for this module.

9 Sensitive Security Parameters Management

9.1 Storage Areas

Storage Area Name	Description	Persistence Type
Hard Disk	Persistent SSPs are stored on the operating system volume (see: Hard Disk in the block diagram).	Static
RAM	Volatile SSPs are temporarily stored in the computer's memory while the module is powered-on (see: RAM in the block diagram).	Dynamic

Table 17: Storage Areas

9.2 SSP Input-Output Methods

The table below lists the input method for SSPs input into the module (Hash Values for Code Integrity to Evaluate and Embedded RSA Signatures for Code Integrity to Evaluate). The SFI or Algorithm column is blank as the module does not use any SFIs or algorithms when the SSP is input.

Name	From	To	Format Type	Distribution Type	Entry Type	SFI or Algorithm
Input from non-encrypted non-volatile storage	Hard Disk	RAM	Plaintext	Manual	Electronic	

Table 18: SSP Input-Output Methods

9.3 SSP Zeroization Methods

Zeroization Method	Description	Rationale	Operator Initiation
Procedural zeroization	Operators may choose to overwrite SSPs in non-volatile storage by reformatting and overwriting the storage media for the operating system. Operators may zeroize SSPs in volatile storage by powering off the host General Purpose Computer (GPC).	Reformatting and overwriting the OS volume at least once ensures adequate zeroization. Operators may choose to overwrite multiple times at their discretion. SSPs in volatile storage are effectively overwritten with zeros when power is lost.	Operators may use the Format command together with the /P parameter which specifies the number of overwrite passes. For more information on the command, operators may type "format /?" at the command prompt. Operators may power off or reboot the host GPC to zeroize SSPs stored in volatile RAM.

Table 19: SSP Zeroization Methods

9.4 SSPs

The tables below list the keys and SSPs used by the module. Per the CMVP, public keys and file hashes used for the module integrity check are not considered SSPs and, as such, have no input method assigned and are categorized as neither PSP nor CSP. Blank cells are either not applicable or optional according to the CMVP.

Name	Description	Size - Strength	Type - Category	Generated By	Established By	Used By
Embedded RSA Signatures for Code Integrity to Evaluate	RSA signatures that Code Integrity uses to verify the authenticity of digitally signed drivers and binary files.	Size: 1024 (legacy use only), 2048, 3072, 4096 bits - Strength: 80 (legacy use only), 112, 128, and 152 bits	Asymmetric Public Key - PSP	Generated external to the module by the Microsoft Windows build process for the Code Integrity catalog.		DigSig-Legacy DigSig1
Embedded X.509 Certificate for CI.DLL (This is not an SSP)	Public key used for RSA PKCS #1 (v1.5) integrity verification of CI.DLL.	Size: 2048-bit - Strength: 112 bits	Asymmetric Public Key - Neither	Generated external to the module by the Microsoft Windows build process.		DigSig2
Hash Value for CI.DLL (This is not an SSP)	File hash used to verify the integrity of CI.DLL.	Size: 256 bits - Strength: 128 bits	File Hash - Neither	Generated external to the module by the Microsoft Windows build process.		SHS2
Hash Values for Code Integrity to Evaluate	File hashes that Code Integrity uses to verify the integrity of digitally signed drivers and binary files.	Size: 160 (legacy use only), 256, 384, or 512 bits - Strength: 80 (legacy-use only), 128, 192, or 256 bits	File Hash - PSP	Generated external to the module by the Microsoft Windows build process for the Code Integrity catalog.		SHS1

Table 20: SSP Table 1

Name	Input - Output	Storage	Storage Duration	Zeroization	Related SSPs
Embedded RSA Signatures for Code Integrity to Evaluate	Input from non-encrypted non-volatile storage	RAM :Plaintext	Until zeroized	Procedural zeroization	
Embedded X.509 Certificate for CI.DLL (This is not an SSP)		Hard Disk :Plaintext		Procedural zeroization	
Hash Value for CI.DLL (This is not an SSP)		Hard Disk :Plaintext		Procedural zeroization	
Hash Values for Code Integrity to Evaluate	Input from non-encrypted non-volatile storage	RAM :Plaintext	Until zeroized	Procedural zeroization	

Table 21: SSP Table 2

9.5 Transitions

The following transition timelines apply to the approved algorithms named in the bullets below:

- The SHA-1 algorithm will become disallowed for applying cryptographic protection starting January 1, 2031, however, its use for legacy digital signature verification will continue to be allowed.
- RSA with a security strength of less than 128 bits will become deprecated starting January 1, 2031, however, its use for legacy digital signature verification will continue to be allowed.
- FIPS 186-4 has been superseded by FIPS 186-5. This transition began on July 25, 2023, and concluded on February 3, 2024. Although testing for this module was completed against FIPS 186-4, it claims compliance with FIPS 186-5 for RSA signature verification – see Section 2.7.1 FIPS 186-4 and 186-5 for more information.

10 Self-Tests

Windows performs tests automatically to ensure integrity and correct functionality. The module will not perform cryptographic functions while in its self-test or error states. If a self-test fails, the module enters the error state and future cryptographic function calls fail. If the self-test passes, cryptographic functions are available for use. As the module has separate CAVP certificates for Windows 11 and Windows Server 2022, the self-test tables below list two identical rows for each algorithm self-test, one for Windows 11 and one for Windows Server 2022.

10.1 Pre-Operational Self-Tests

The Windows OS Loader module (EVM) checks the integrity of CI.DLL before it is loaded. The algorithms used for the pre-operational self-tests pass their own algorithm self-tests before integrity verification is performed. See section 5.1 Integrity Techniques for details on the EVM and how the module's integrity is checked.

Algorithm or Test	Test Properties	Test Method	Test Type	Indicator	Details
RSA SigVer (FIPS186-4) (A3767)	RSA PKCS#1v1.5 with 2048-bit key and SHA2-256 (Windows 11 version 22H2)	Software Integrity	SW/FW Integrity	The Perform Cryptographic Algorithm Self-Tests service runs.	[EVM] signature verification.
RSA SigVer (FIPS186-4) (A3768)	RSA PKCS#1v1.5 with 2048-bit key and SHA2-256 (Windows Server 2022)	Software Integrity	SW/FW Integrity	The Perform Cryptographic Algorithm Self-Tests service runs.	[EVM] signature verification.

Table 22: Pre-Operational Self-Tests

10.2 Conditional Self-Tests

The module performs the conditional cryptographic algorithm self-tests automatically when the module is loaded into memory, after the pre-operational software integrity tests described above have completed. The module implements Known Answer Test (KAT) functions each time the module is loaded by the Windows kernel and CiInitialize is called. If any self-test fails, the module will not load and a failure status, STATUS_INVALID_IMAGE_HASH, is returned, and the computer will not complete the boot process. Otherwise, STATUS_SUCCESS is returned and the boot process completes.

Algorithm or Test	Test Properties	Test Method	Test Type	Indicator	Details	Conditions
[EVM] RSA SigVer (FIPS186-4) (A3767)	RSA PKCS#1v1.5 with 2048-bit key and SHA2-256 (Windows 11 version 22H2)	KAT	CAST	The Perform Cryptographic Algorithm Self-Tests service runs.	[EVM] signature verification.	The self-tests are run prior to pre-operational integrity test during module initialization.
[EVM] RSA SigVer (FIPS186-4) (A3768)	RSA PKCS#1v1.5 with 2048-bit key and SHA2-256 (Windows Server 2022)	KAT	CAST	The Perform Cryptographic Algorithm Self-Tests service runs.	[EVM] signature verification.	The self-tests are run prior to pre-operational integrity test during module initialization.
[EVM] SHA2-256 (A4008)	256 bits (Windows 11 version 22H2)	KAT	CAST	The Perform Cryptographic Algorithm Self-Tests service runs.	[EVM] Secure hash	The self-tests are run prior to pre-operational integrity test during module initialization.
[EVM] SHA2-256 (A4009)	256 bits (Windows Server 2022)	KAT	CAST	The Perform Cryptographic Algorithm Self-Tests service runs.	[EVM] Secure hash	The self-tests are run prior to pre-operational integrity test during module initialization.
RSA SigVer (FIPS186-4) (A4008)	RSA PKCS#1v1.5 with 2048-bit key and SHA2-256 (Windows 11 version 22H2)	KAT	CAST	STATUS_SUCCESS returned to CiInitialize.	Signature verification	Run at every module initialization (when CiInitialize is called) after the module integrity is verified.
RSA SigVer (FIPS186-4) (A4009)	RSA PKCS#1v1.5 with 2048-bit key and SHA2-256 (Windows Server 2022)	KAT	CAST	STATUS_SUCCESS returned to CiInitialize.	Signature verification	Run at every module initialization (when CiInitialize is called) after the module integrity is verified.
SHA-1 (A4008)	160 bits (Windows 11 version 22H2)	KAT	CAST	STATUS_SUCCESS returned to CiInitialize.	Secure hash	Run at every module initialization (when CiInitialize is called) after the module integrity is verified.
SHA-1 (A4009)	160 bits (Windows Server 2022)	KAT	CAST	STATUS_SUCCESS returned to CiInitialize.	Secure hash	Run at every module initialization (when CiInitialize is called) after the module integrity is verified.
SHA2-256 (A4008)	256 bits (Windows 11 version 22H2)	KAT	CAST	STATUS_SUCCESS returned to CiInitialize.	Secure hash	Run at every module initialization (when CiInitialize is called) after the module integrity is verified.
SHA2-256 (A4009)	256 bits (Windows Server 2022)	KAT	CAST	STATUS_SUCCESS returned to CiInitialize.	Secure hash	Run at every module initialization (when CiInitialize is called) after the module integrity is verified.

Algorithm or Test	Test Properties	Test Method	Test Type	Indicator	Details	Conditions
SHA2-512 (A4008)	512 bits (Windows 11 version 22H2)	KAT	CAST	STATUS_SUCCESS returned to CiInitialize.	Secure hash	Run at every module initialization (when CiInitialize is called) after the module integrity is verified.
SHA2-512 (A4009)	512 bits (Windows Server 2022)	KAT	CAST	STATUS_SUCCESS returned to CiInitialize.	Secure hash	Run at every module initialization (when CiInitialize is called) after the module integrity is verified.

Table 23: Conditional Self-Tests

10.3 Periodic Self-Test Information

The tables below present the periodic self-test information for the module. The set of self-tests presented in tables 23 and 24 below is identical to the set of self-tests presented in tables 21 and 22 above. The Windows OS Loader module (EVM) checks the integrity of CI.DLL before it is loaded. See section 5.1 Integrity Techniques for details on the EVM and how the module's integrity is checked.

Algorithm or Test	Test Method	Test Type	Period	Periodic Method
RSA SigVer (FIPS186-4) (A3767)	Software Integrity	SW/FW Integrity	Pre-operational integrity test is run at every module initialization before the CASTs (Windows 11 version 22H2).	Manual on-demand (operator initiated by rebooting the computer)
RSA SigVer (FIPS186-4) (A3768)	Software Integrity	SW/FW Integrity	Pre-operational integrity test is run at every module initialization before the CASTs (Windows Server 2022).	Manual on-demand (operator initiated by rebooting the computer)

Table 24: Pre-Operational Periodic Information

Algorithm or Test	Test Method	Test Type	Period	Periodic Method
[EVM] RSA SigVer (FIPS186-4) (A3767)	KAT	CAST	Conditional CAST is run at every module initialization prior to module's integrity check (Windows 11 version 22H2).	Manual on-demand (operator initiated by rebooting the computer)
[EVM] RSA SigVer (FIPS186-4) (A3768)	KAT	CAST	Conditional CAST is run at every module initialization prior to module's integrity check (Windows Server 2022).	Manual on-demand (operator initiated by rebooting the computer)

Algorithm or Test	Test Method	Test Type	Period	Periodic Method
[EVM] SHA2-256 (A4008)	KAT	CAST	Conditional CAST is run at every module initialization prior to module's integrity check (Windows 11 version 22H2).	Manual on-demand (operator initiated by rebooting the computer)
[EVM] SHA2-256 (A4009)	KAT	CAST	Conditional CAST is run at every module initialization prior to module's integrity check (Windows Server 2022).	Manual on-demand (operator initiated by rebooting the computer)
RSA SigVer (FIPS186-4) (A4008)	KAT	CAST	Conditional CASTs are run at every module initialization after the module integrity is verified (Windows 11 version 22H2).	Manual on-demand (operator initiated by rebooting the computer)
RSA SigVer (FIPS186-4) (A4009)	KAT	CAST	Conditional CASTs are run at every module initialization after the module integrity is verified (Windows Server 2022).	Manual on-demand (operator initiated by rebooting the computer)
SHA-1 (A4008)	KAT	CAST	Conditional CASTs are run at every module initialization after the module integrity is verified (Windows 11 version 22H2)	Manual on-demand (operator initiated by rebooting the computer)
SHA-1 (A4009)	KAT	CAST	Conditional CASTs are run at every module initialization after the module integrity is verified (Windows Server 2022)	Manual on-demand (operator initiated by rebooting the computer)
SHA2-256 (A4008)	KAT	CAST	Conditional CASTs are run at every module initialization after the module integrity is verified (Windows 11 version 22H2)	Manual on-demand (operator initiated by rebooting the computer)
SHA2-256 (A4009)	KAT	CAST	Conditional CASTs are run at every module initialization after the module integrity is verified (Windows Server 2022)	Manual on-demand (operator initiated by rebooting the computer)

Algorithm or Test	Test Method	Test Type	Period	Periodic Method
SHA2-512 (A4008)	KAT	CAST	Conditional CASTs are run at every module initialization after the module integrity is verified (Windows 11 version 22H2).	Manual on-demand (operator initiated by rebooting the computer)
SHA2-512 (A4009)	KAT	CAST	Conditional CASTs are run at every module initialization after the module integrity is verified (Windows Server 2022).	Manual on-demand (operator initiated by rebooting the computer)

Table 25: Conditional Periodic Information

10.4 Error States

Name	Description	Conditions	Recovery Method	Indicator
Boot Fail	Boot failure	The error state occurs if the module integrity test fails, if any of the cryptographic algorithm self-tests fail, or if any of the binary integrity checks fail.	Restart the computer	Boot failure blue screen error message

Table 26: Error States

10.5 Operator Initiation of Self-Tests

To perform the module self-tests on demand, including running the approved services Perform Pre-operational Software Integrity Test [EVM] and Perform Cryptographic Algorithm Self-Tests, the user may reboot the computer.

11 Life-Cycle Assurance

11.1 Installation, Initialization, and Startup Procedures

The Windows 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 version downloaded from Windows Update. An inspection of authenticity can be made by following the guidance at this Microsoft web site: https://www.microsoft.com/en-us/howtotell/default.aspx.

For Windows Updates, the client only accepts binaries signed by Microsoft certificates. The Windows Update client only accepts content whose SHA2 hash matches the SHA2 hash specified in the metadata. All metadata communication is done over a Transport Layer Security (TLS) port. Using TLS ensures that the client is communicating with the real server and so prevents a malicious TLS server from communicating to the TLS client. The version and digital signature of new cryptographic module must be verified to match the version that was validated. See Section 11.2 Administrator Guidance for details on how to do this.

Module initialization occurs automatically as part of the Windows boot process. The finite state model diagram below visualizes the initialization process along with other module states. Every state of the module can transition to the power-off state through power-cycle/rebooting the host machine.

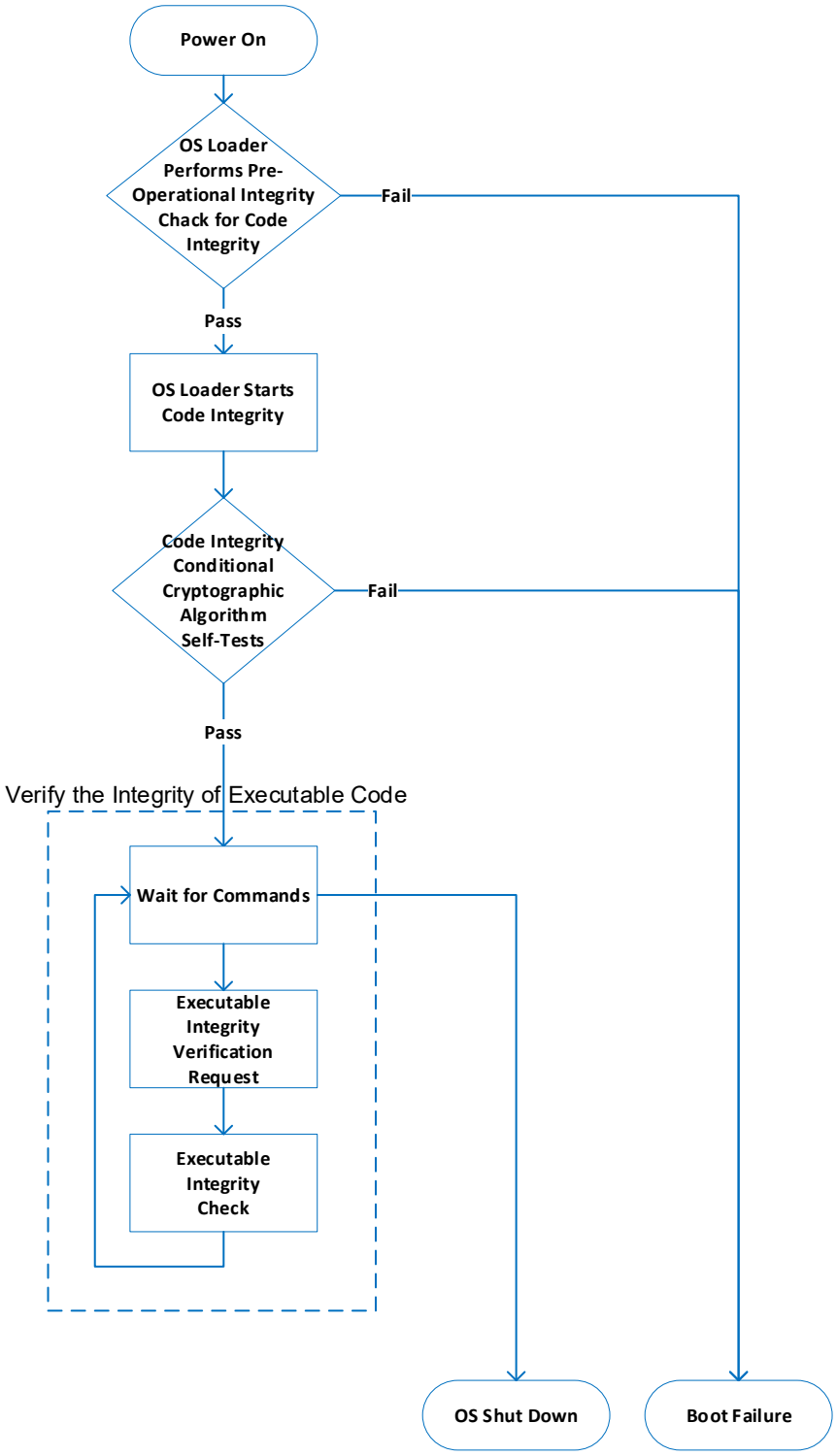

Figure 3: Finite State Model

11.2 Administrator Guidance

The installed version of Windows must be checked to match the version that was validated. See Section 11.2.1 Verifying the Installed Windows Version below for details on how to do this. To sanitize the module, the operator should reformat the hard drive or wipe the device as part of unenrollment for Azure Active Directory.

11.2.1 Verifying the Installed Windows Version

The following methods may be used to check the installed version of Windows against the version number listed in 2.2 Tested and Vendor Affirmed Module Version and Identification.

Using the Windows command prompt or Windows PowerShell (local or remote):

- Open a command prompt or PowerShell window.
- At the prompt, type **systeminfo** and press the Enter key.
- Near the top of the output, information like the following is displayed. The OS Version field lists the installed Windows version. Compare this version number against the version number listed in 2.2 Tested and Vendor Affirmed Module Version and Identification.

```
OS Name:                   Microsoft Windows 11 Enterprise
OS Version:                10.0.xxxxx N/A Build xxxxx
OS Manufacturer:           Microsoft Corporation
```

For Windows installations without a user interface, e.g., Windows Server with the Core Installation option, a server management solution may also be used to validate the installed Windows version. For example, the Overview page of Windows Admin Center Server Manager lists the version number under the Operating System category. For more information, see Manage Servers with Windows Admin Center.

11.2.2 Verifying the Cryptographic Module Version and its Signature

To confirm the version number or digital signature of the module, locate the module binary or binaries named in 2.2 Tested and Vendor Affirmed Module Version and Identification in their default installation location. The list below identifies the default install locations for the Windows cryptographic module binaries for a system where Windows has been installed on the C: drive.

- ci.dll - C:\Windows\System32\

To validate the module version number, use Windows Explorer or PowerShell (local or remote).

- Using Windows Explorer:
  - Open Windows Explorer and navigate to the folder where the binary is installed, referencing the list above for the correct location.
  - Find the file in the folder and **right click** on the file's icon.
  - Select **Properties** from the context menu.
  - Select the **Details** tab.
  - Compare the version number in the **File version** field against the version identified in 2.2 Tested and Vendor Affirmed Module Version and Identification.

- Using PowerShell:
  - Open a **PowerShell** window.

- Use the **Get-ItemProperty cmdlet** together with the path of the cryptographic module binary identified above, formatting the output as a list. For example, if the binary path is `C:\Windows\System32\ci.dll`, the PowerShell command is:

  ```
  Get-ItemProperty -Path "C:\Windows\System32\ci.dll" | Format-List
  ```

- The cmdlet will return output that summarizes file property details, including the version number. If the version number listed in the **VersionInfo / ProductVersion** field matches one of the version numbers identified in 2.2 Tested and Vendor Affirmed Module Version and Identification, then the module version has been verified.
- Full documentation for the Get-ItemProperty cmdlet may be found at: https://learn.microsoft.com/en-us/powershell/module/microsoft.powershell.management/get-itemproperty.

To validate the Windows digital signature for the module binary, use Windows Explorer or PowerShell (local or remote).

- Using Windows Explorer:
  - Open Windows Explorer and navigate to the folder where the binary is installed, referencing the list at the beginning of this section for the correct location.
  - Find the file in the folder and **right click** on the file's icon.
  - Select **Properties** from the context menu.
  - Select the **Digital Signatures** tab.
  - In the **Signature** list, select the **Microsoft Windows** signer.
  - Click the **Details** button.
  - 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.

- Using PowerShell:
  - Open a **PowerShell** window.
  - Use the **Get-AuthenticodeSignature cmdlet** together with the path the path of the cryptographic module binary identified at the beginning of this section. For example, if the binary path is `C:\Windows\System32\ci.dll`, the PowerShell command is:

    ```
    Get-AuthenticodeSignature -FilePath "C:\Windows\System32\ci.dll"
    ```

- The cmdlet will return output that summarizes signature details. If the signature is valid, the **Status** field will show "`Valid`" and the **StatusMessage** field will show "`Signature verified.`"
- Full documentation for the Get-AuthenticodeSignature cmdlet may be found at: https://learn.microsoft.com/en-us/powershell/module/microsoft.powershell.security/get-authenticodesignature.

11.3 Non-Administrator Guidance

The module implements a single role only, Cryptographic Officer. See the Administrator Guidance above.

11.4 Design and Rules

The module is a software cryptographic module that provides cryptographic services within the Operational Environments listed in Section 2.2 Tested and Vendor Affirmed Module Version and Identification. The other sections of this Security Policy provide additional details on the design of the module and its rules of operation.

12 Mitigation of Other Attacks

12.1 Attack List

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

Algorithm	Protected Against	Mitigation
SHA-1	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.

Table 27: Mitigation of Other Attacks

13 Standards References

- FIPS 140-3, Security Requirements for Cryptographic Modules, https://csrc.nist.gov/publications/detail/fips/140/3/final
- FIPS 180-4, Secure Hash Standard (SHS), https://csrc.nist.gov/publications/detail/fips/180/4/final
- FIPS 186-4, Digital Signature Standard (DSS), https://csrc.nist.gov/publications/detail/fips/186/4/final
- FIPS 186-5, Digital Signature Standard (DSS), https://csrc.nist.gov/pubs/fips/186-5/final