Non-proprietary Security Policy for

FIPS 140-2 Validation

Secure Kernel Code Integrity (skci.dll) in Microsoft Windows 10 Pro Windows 10 Enterprise Windows 10 Enterprise LTSB Windows Server 2016 Standard Windows Server 2016 Datacenter Windows Storage Server 2016

DOCUMENT INFORMATION

Version Number	1.0
Updated On	December 12, 2016

CHANGE HISTORY

Date	Version	Updated By	Change
12 DEC 2016	1.0	Tim Myers	First release to validators

Table of Contents

1 Introduction

Secure Kernel Code Integrity (SKCI) running in the Virtual Secure Mode (VSM) of the Hyper-V hypervisor will only grant execute access to physical pages in the kernel that have been successfully verified. Executable pages will not have write permission outside of Hyper-V. Therefore, only verified code can be executed.

Throughout this document, Secure Kernel Code Integrity may also be referred to as SKCI and SKCI.DLL.

SKCI 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.

The Operational Environments (OEs) are:

1. Windows 10 Enterprise Anniversary Update (x64) running on a Microsoft Surface Pro 3 - Intel Core i7 with AES-NI and PCLMULQDQ and SSSE 3
2. Windows 10 Enterprise Anniversary Update (x64) running on a Microsoft Surface Pro 4 – Intel Core i5 with AES-NI and PCLMULQDQ and SSSE 3
3. Windows 10 Enterprise Anniversary Update (x64) running on a Microsoft Surface Book – Intel Core i7 with AES-NI and PCLMULQDQ and SSSE 3
4. Windows 10 Enterprise Anniversary Update (x64) running on a Dell Precision Tower 5810MT - Intel Xeon with AES-NI and PCLMULQDQ and SSSE 3
5. Windows 10 Enterprise Anniversary Update (x64) running on a HP Compaq Pro 6305 - AMD A4 with AES-NI and PCLMULQDQ and SSSE 3
6. Windows 10 Pro Anniversary Update (x64) running on a Microsoft Surface Pro 3 - Intel Core i7 with AES-NI and PCLMULQDQ and SSSE 3
7. Windows 10 Pro Anniversary Update (x64) running on a Microsoft Surface Pro 4 - Intel Core i5 with AES-NI and PCLMULQDQ and SSSE 3
8. Windows 10 Pro Anniversary Update (x64) running on a Microsoft Surface Book - Intel Core i7 with AES-NI and PCLMULQDQ and SSSE 3
9. Windows 10 Pro Anniversary Update (x64) running on a Dell Precision Tower 5810MT - Intel Xeon with AES-NI and PCLMULQDQ and SSSE 3
10. Windows 10 Enterprise LTSB Anniversary Update (x64) running on a Dell Precision Tower 5810MT - Intel Xeon with AES-NI and PCLMULQDQ and SSSE 3
11. Windows 10 Enterprise LTSB Anniversary Update (x64) running on a Dell XPS 8700 - Intel Core i7 with AES-NI and PCLMULQDQ and SSSE 3
12. Windows Server 2016 Standard Edition running on a HP Compaq Pro 6305 - AMD A4 with AES-NI and PCLMULQDQ and SSSE 3
13. Windows Server 2016 Standard Edition running on a Dell PowerEdge R630 Server - Intel Xeon with AES-NI and PCLMULQDQ and SSSE 3
14. Windows Server 2016 Datacenter Edition running on a Dell PowerEdge R630 Server - Intel Xeon with AES-NI and PCLMULQDQ and SSSE 3
15. Windows Storage Server 2016 running on a Dell PowerEdge R630 Server - Intel Xeon with AES-NI and PCLMULQDQ and SSSE 3

herein referred to as Windows 10 OEs.

1.1 List of Cryptographic Module Binary Executables

SKCI.DLL – Version 10.0.14393 for Windows 10 OEs

1.2 Brief Module Description

SKCI is a dynamically-linked library used to verify the integrity of executable code pages in the kernel.

1.3 Validated Platforms

The SKCI component listed in Section 1.1 was validated using the machine configurations specified in the list of Windows 10 OEs.

1.4 Cryptographic Boundary

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

2 Security Policy

SKCI is in a FIPS mode of operation when the following rules are followed:

- SKCI is supported on Windows 10 OEs.
- Windows 10 OEs are operating systems supporting a "single user" mode where there is only one interactive user during a logon session.
- SKCI 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.
- SKCI operates in FIPS mode of operation only when used with the FIPS approved version of Windows 10 OEs Code Integrity (CI.DLL) validated to FIPS 140-2 under Cert. # 2935 operating in FIPS mode.

The following diagram, Figure 1, illustrates the master components of the SKCI module:

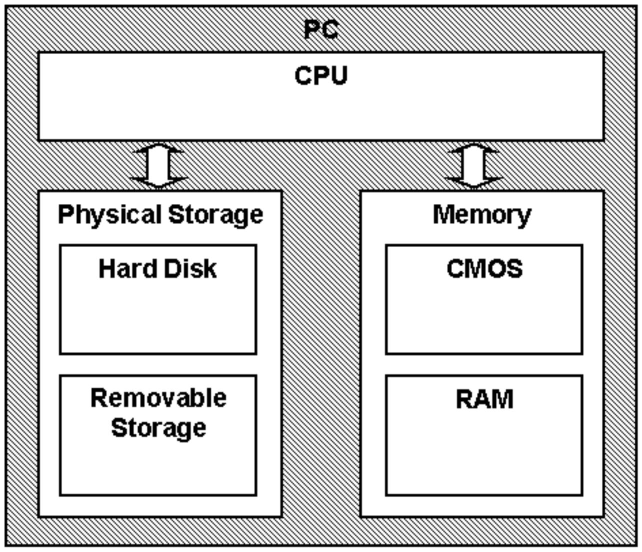

Figure 1

- SKCI's main service is to verify the integrity of executable code pages in the kernel. In addition to this service, SKCI also provides status services. These status services indicate whether the integrity checks passed.
- All services implemented within SKCI are available to the Crypto officer role. The Crypto officer role is assumed by the operating system processes that will invoke executable page image verification in SKCI.

2.1 FIPS 140-2 Approved Algorithms

SKCI implements the following FIPS 140-2 Approved algorithms:

- 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. # 2193)
- FIPS 180-4 SHS SHA-1, SHA-256, SHA-384, and SHA-512 (Cert. # 3347)

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

2.2 Non-Approved Algorithms

SKCI also includes a legacy implementation of MD5, which is not allowed for usage in FIPS mode. 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 SKCI.

2.4 Machine Configurations

SKCI 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 the Code Integrity (CI.dll), which is protected by an RSA signature with a 2048-bit key and SHA-256 message digest, before loading it into memory. Code Integrity verifies the origin and integrity of SKCI before it is loaded into memory and executed. Code Integrity also ensures SKCI has been 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.

4 Ports and Interfaces

4.1 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 kernel. These functions are explained further in the subsequent subsections.

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

4.1.1 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.

As the power-on (startup) function of SKCI, SkciInitialize() 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, SkciInitialize() returns STATUS_INVALID_IMAGE_HASH.

4.1.2 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.

4.1.3 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.

4.1.4 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.

4.1.5 SkciValidateDynamicCodePages

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

4.1.6 SkciFinalizeSecureImageHash

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

4.1.7 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.

4.1.8 SkciFreeImageContext

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

4.1.9 SkciTransferVersionResource

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

4.2 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.

4.3 Status Output Interface

The Status Output Interface for SKCI also consists of the 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).

4.4 Data Input Interface

The Data Input Interface for SKCI also consists of the 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.

4.5 Data Output Interface

The Data Output Interface for SKCI also consists of the export functions. Data is returned to the function’s caller via output parameters.

5 Specification of Roles

SKCI supports both User and Cryptographic Officer roles (as defined in FIPS 140-2). Both roles have access to all services implemented in SKCI 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.

6 Services

SKCI’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)

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

Secure Kernel Code Integrity

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

Table 1

Service	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 the secure kernel loader starts to load a binary executable file.
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.

The following table maps services and export functions.

Table 2

Service	Export Functions
Verify the integrity of binary executable code	SkciInitialize SkciCreateCodeCatalog SkciCreateSecureImage SkciValidateImageData SkciValidateDynamicCodePages SkciFinalizeSecureImageHash SkciFinishImageValidation SkciFreeImageContext SkciTransferVersionResource
Provide Show Status services that indicate whether the integrity checks passed	SkciInitialize SkciCreateCodeCatalog SkciCreateSecureImage SkciValidateImageData SkciValidateDynamicCodePages SkciFinalizeSecureImageHash SkciFinishImageValidation SkciFreeImageContext SkciTransferVersionResource
Provide Self-Test services	SkciInitialize
Legacy certificate chain authentication (non-FIPS approved service)	SkciInitialize SkciCreateCodeCatalog SkciCreateSecureImage SkciValidateImageData SkciValidateDynamicCodePages SkciFinalizeSecureImageHash SkciFinishImageValidation SkciFreeImageContext SkciTransferVersionResource

6.1 Verification of Integrity Service

SKCI verifies the integrity of digitally signed drivers, Dynamic-linked Libraries (DLLs), and other binary executables.

6.2 Show Status Services

The status information is returned to the caller as the return value from the function. The User / Cryptographic Officer does not have any direct access to the return value, but rather, they may observe the failure of applications or services to load.

6.3 Self-Test Services

Secure Kernel Code Integrity automatically executes Self-Tests upon being loaded, which provides the User / Cryptographic Officer assurance that the module is operating properly. Upon failing a Self-Test, this module will fail to load and return an error indicator (as described in section 4.1.1) which may be observed by the User / Cryptographic Officer as a failure of applications or services to load. The Self-Test functionality is described in Section 10 Self-Tests.

6.4 Service Inputs / Outputs

The User / Cryptographic Officer does not have access to the service inputs and outputs that are specified in Section 0 Ports and Interfaces.

7 Operational Environment

The operational environment for SKCI is the Windows 10 OEs running on the software and hardware configurations listed in Section 1.3 - Validated Platforms.

8 Authentication

SKCI does not implement any authentication services. The User and Cryptographic Officer roles are assumed implicitly by booting the Windows operating system.

9 Cryptographic Key Management

SKCI 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

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.

9.1 Cryptographic Keys

The SKCI crypto module uses the following cryptographic keys:

Table 3

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

9.2 Critical Security Parameters

The SKCI crypto module does not contain any Critical Security Parameters (CSPs).

9.3 Access Control Policy

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

10 Self-Tests

10.1 Power-On Self-Tests

SKCI 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.

The integrity of SKCI itself is protected by an RSA signature with a 2048-bit key and SHA-256 message digest, which is verified by Code Integrity (CI.DLL) before SKCI 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

SKCI 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 10 OEs. The various methods of delivery and installation for each product are listed in the following table.

Table 4

Product	Delivery and Installation Method
Windows 10 Pro, Windows 10 Enterprise, Windows Enterprise LTSB, Windows Server 2016 Standard, Windows Server 2016 Datacenter	• Pre-installed on the computer by OEM • Download that updates to Windows 10 • Enterprise IT deployment
Surface Book, Surface Pro 4, Surface Pro 3, Windows Storage Server 2016	• Pre-installed by the OEM (Microsoft)

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: https://www.microsoft.com/en-us/howtotell/default.aspx

The installed version of Windows 10 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.

12 Mitigation of Other Attacks

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

Table 5

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.

Table 6

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:

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

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

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

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

15.1 How to Verify Windows Versions

The installed version of Windows 10 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.

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: 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.