Virtru Corporation

VirtruCrypto - FIPS JavaScript Module

Document Version 1.2.5 - for Module Version 1.2.0

FIPS 140-2 Non-Proprietary Security Policy

February 2, 2023

1. Introduction

Federal Information Processing Standards (FIPS) Publication 140-2 — Security Requirements for Cryptographic Modules specifies requirements for cryptographic modules to be deployed in a Sensitive but Unclassified environment. The National Institute of Standards and Technology (NIST) and Canadian Centre for Cyber Security (CCCS) Cryptographic Module Validation Program (CMVP) run the FIPS 140 program. The NVLAP accredits independent testing labs to perform FIPS 140 testing; the CMVP validates modules meeting FIPS 140 validation. Validated is the term given to a module that is documented and tested against the FIPS 140 criteria.

More information is available on the CMVP website at http://csrc.nist.gov/groups/STM/cmvp/index.html.

About this Document

This non-proprietary Cryptographic Module Security Policy for VirtruCrypto - FIPS JavaScript Module provides an overview of the product and a high-level description of how it meets the overall Level 1 security requirements of FIPS 140-2.

Notices

2. FIPS 140-2 Security Level

The following table lists the level of validation for each area in FIPS 140-2:

FIPS 140-2 Section Title	Validation Level
Cryptographic Module Specification	1
Cryptographic Module Ports and Interfaces	1
Roles, Services & Authentication	1
Finite State Model	1
Physical Security	N/A
Operational Environment	1
Cryptographic Key Management	1
Electromagnetic Interference / Electromagnetic Compatibility	1
Self-Tests	1
Design Assurance	1
Mitigation of Other Attacks	N/A
Overall Level	1

Table 1 – FIPS 140-2 Security Levels

3. Cryptographic Module Specification

The VirtruCrypto module is a multi-chip standalone cryptographic module that operates with:

- A commercially available JavaScript engine that supports WebAssembly (WASM), such as a browser (e.g., Chrome) or JavaScript runtime environment (e.g., node).
- A commercially available general-purpose computer hardware.
- A commercially available Operating System (OS) that runs on the computer hardware.

The VirtruCrypto module was tested on the following operational environments on the general-purpose computer (GPC) platforms detailed below. In testing, it was run under node with WASM enabled, on version 12.x of node:

Operational Environment	Processor Family	Computer Model
Windows 10	Intel (R) Xeon(R) Silver 4108 CPU (non-PAA)	HP Z8

Table 2 – Tested Operational Environments

As per FIPS 140-2 Implementation Guidance G.5, compliance is maintained for other versions of the respective operational environments where the module binary is unchanged. FIPS 140-2 compliance is maintained on any GPC provided that the GPC uses the specified single user operating system/mode specified on the validation certificate, or another compatible single user operating system. No claim can be made as to the correct operation of the module or the security strengths of the generated keys if any source code is changed and the module binary is reconstructed.

The module will run on various hardware and OS while running in a JavaScript engine while implementing the same code within the scope of the boundary (see Figure 1) with regards to the FIPS 140-2 Level 1 requirements and, thus, the vendor may affirm JavaScript engines running on other platforms as well[1]. Compatible platforms/OEs can be described as those with a single user operational environment, where each user application and JavaScript 'program' runs in a virtually separated independent space, with access enforcement. Operating systems such as Linux, MacOS, and Windows and browsers such as Chrome provide such an operational environment.

The module only operates in FIPS mode.

3.1 Cryptographic Boundary

The VirtruCrypto cryptographic module boundary is a WebAssembly binary made up of three components: a C-wrapper, Boring SSL library (limited to the cryptographic operations defined in this Security Policy), and an initialization function to verify a HMAC-SHA-256 hash against the WebAssembly binary to ensure integrity.

Figure 1 shows the logical relationship of the cryptographic module components.

[1] CMVP makes no statement as to the correct operation of the module or the security strengths of the generated keys when ported to an operational environment which is not listed on the validation certificate (see IG G.5).

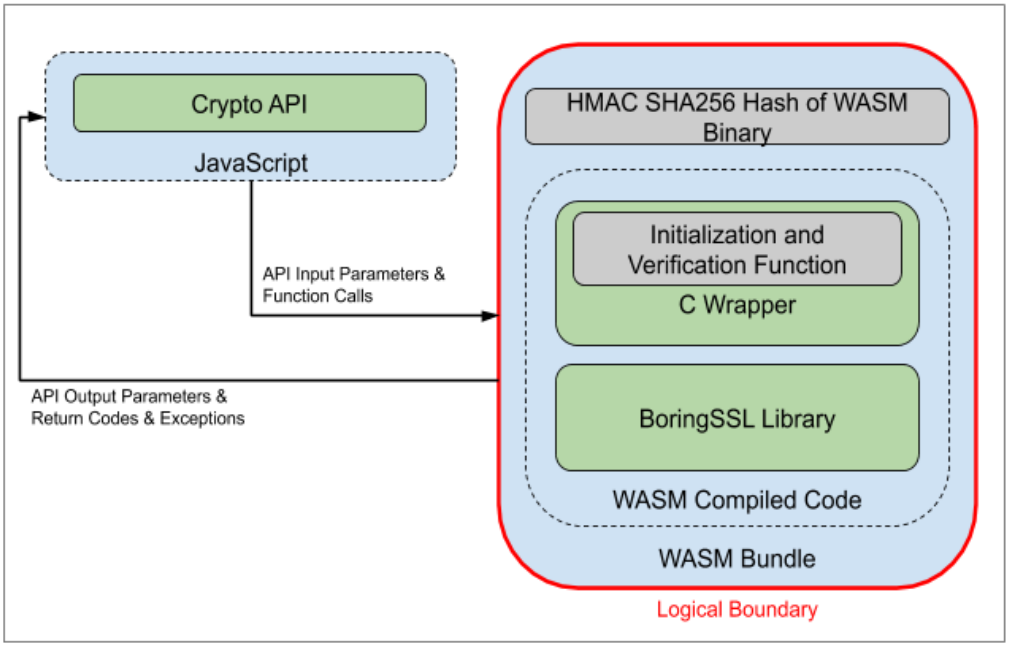

Figure 1 - Logical & Physical Boundaries

The VirtruCrypto API closely follows the functionality of the Web Cryptography API, W3C 1/26/2017. WebCrypto functions all return a JavaScript Promise, with the actual result value passed on success or generating an exception on failure.

3.2 Software Specifications

The VirtruCrypto cryptographic module provides services to JavaScript computer language users in the form of a JavaScript library with packaged WebAssembly. The same binary is used across different hardware and OS combinations as the JavaScript engine will absorb the differences of the computer and hardware and OS.

The interface into the VirtruCrypto cryptographic module is via Application Programming Interface (API) method calls. These method calls provide the interface to the cryptographic services, for which the parameters and return codes provide the control input and status output.

4. Cryptographic Module Ports and Interfaces

I/O	Logical Interface	Physical Interface
Data Input	API Input Parameters	Physical Ports of the Platform
Data Output	API Output Parameters	Physical Ports of the Platform

	& Return Codes	
Control Input	API Function Calls & Input Parameters	Physical Ports of the Platform
Status Output	API Output Parameters & Return Codes and/or Exceptions	Physical Ports of the Platform
Power Input	N/A	Physical Ports of the Platform

Table 3 – Module Ports and Interfaces

Control of the physical ports is outside module scope as this is a software module. However, when the module is not initialized or is in an error state, all output on the module's logical data output interfaces is inhibited.

5. Roles, Authentication and Services

The cryptographic module implements both User and Crypto Officer (CO) roles. The module does not support user authentication. The User and CO roles are implicitly assumed by the entity accessing services implemented by the module. A user is considered the owner of the thread that instantiates the module and, therefore, only one concurrent user is allowed.

The following abbreviations are used in Table 4: 'X' for Execute, 'R' for Read, 'W' for Write.

Service	**Approved security functions**	**Access to CSP**	**CSP**	**Crypto Officer**	**User**
Initialization	Initialization	N/A	N/A	x	x
Status	Show Status	N/A	N/A	x	x
	Perform Self Test on Demand	N/A	N/A	x	x
Symmetric (AES)	Key Generation	W	AES Key	x	x
	Key Import	W		x	x
	Key Export	R		x	x
	Encrypt	X		x	x
	Decrypt	X		x	x

Service	Approved security functions	Access to CSP	CSP	Crypto Officer	User
	Key Zeroize	W		x	x
Hashing & Message Auth	Hashing	N/A	N/A	x	x
	Key Generation	W	HMAC Key	x	x
	Key Import	W		x	x
	Key Export	R		x	x
	Message Authentication	X		x	x
	Key Zeroize	W		x	x
Digital Signature	Key Pair Generation	W	RSA Private Key	x	x
	Key Pair Import	W		x	x
	Key Pair Export	R		x	x
	Sign	X		x	x
	Verify	X		x	x
	Key Zeroize	W		x	x
Asymmetric (RSA)	Key Pair Generation	W	RSA Private Key	x	x
	Key Pair Import	W		x	x
	Key Pair Export	R		x	x
	Encrypt			x	x
	Key Wrap	X		x	x
	Decrypt			x	x

Service	Approved security functions	Access to CSP	CSP	Crypto Officer	User
	Key Unwrap	X		x	x
	Key Zeroize	W		x	x
Entropy	Generate	W	DRBG	x	x
	Zeroize	W		x	x

Table 4 – Roles and Services

6. Physical Specifications

The cryptographic module is comprised of software only and does not claim any physical security.

7. Operational Environment

This module is to be run in a single user operational environment, where each user application runs in virtually separated independent space. Note that modern Operating Systems such as UNIX, Linux and Windows provide such an operational environment.

8. Cryptographic Algorithms and Key Management

8.1. Approved Cryptographic Algorithms

The module implements the following FIPS 140-2 Approved algorithms:

Category	Algorithm	Standard	FIPS Approved	Cert #
Block Ciphers	AES-256 (CBC, GCM, GMAC)	FIPS 197, SP 800-38D	x	A2602
Hash Functions	SHA-256	FIPS 180-4	x	A2602
Message Authentication	HMAC-SHA-256	FIPS 198-1	x	A2602
Digital Signature (SigGen and SigVer (using	RSA PKCS #1 v1.5 Signature (2048-bit)	PKCS #1 v2.1	x	A2602

SHA2-256), KeyGen)				
DRBG	AES-CTR DRBG (256)	NIST SP 800-90A	x	A2602
Entropy	ENT (P)	NIST SP 800-90B	x	N/A
Cryptographic Key Generation	CKG[2]	NIST SP 800-133	x	Vendor Affirmed

Table 5 – Supported Approved Algorithms and Standards

Algorithm	Usage
RSA Key Wrapping, Non-SP 800-56B compliant	Key establishment methodology provides between 112 and 256 bits of encryption strength

Table 6 – Supported Non-Approved but Allowed Algorithms and Standards

Algorithm	Key	Key Size	Security Strength
AES	Key	256 bits	256
HMAC	Key	256 bits	256
RSA Signature	Key Pair	2048 bits	112
RSA Key (Key Transport)	Key Pair	2048 bits	112
DRBG	Key	256 bits	256

Table 7 – Key, Key Size, and Security Strength

8.2. Cryptographic Key Management

The table below provides a list of all Private keys and CSPs used by the module:

Key/CSP Name	Key Description	Generated/Input	Output
CTR_DRBG Key	256 bits	Internally Generated	None
CTR_DRBG V (seed)	256 bits	Internally Generated	None

[2] Symmetric key and asymmetric seed generation in accordance with SP 800-133rev2 and IG D.12.

CTR_DRBG Entropy Input	384 bits	Internally Generated	None
AES-GCM Key	AES (256) encrypt/decrypt/ generate/verify key	Internally Generated or input via API in plaintext	If permitted during import or generation, output via API in plaintext
AES-CBC Key	AES (256) encrypt/decrypt/ generate/verify key	Internally Generated or input via API in plaintext	If permitted during import or generation, output via API in plaintext
HMAC Key	Keyed hash key (256)	Internally Generated or input via API in plaintext	If permitted during import or generation, output via API in plaintext
RSA Key (Key Transport)	RSA (2048 bits) decryption (private key transport) key	Internally Generated or input via API in plaintext	If permitted during import or generation, output via API in plaintext

Table 8 – Key Management

Generated Keys are compliant to SP 800-133.

The Module supports internal IV generation using the module's Approved DRBG. The generated IV is at least 96 bits in length as per the NIST SP 800-38D, Section 8.2.2 requirements. The approved DRBG generates outputs such that the (key, IV) pair collision probability is less than 2^{-32} as per FIPS 140-2 IG A.5 Scenario 2 and NIST SP 800-38D.

8.3. Public Keys

The table below provides a complete list of the Public keys used by the module:

Public Key Name	Key Description
RSA Key (Key Transport)	RSA (2048 bits) encryption (public key transport) key
RSA Signature Verification Key	RSA (2048 bits) signature verification public key

Table 9 - Public Keys

8.4. Key Generation

The module supports generation of RSA key pairs as specified in Section 5 of NIST SP 800-133. Specifically, the module generates the pair as the key-pair owner. For signature use, FIPS 186 Section 5.1 applies. The module uses a generalized version of the algorithm given in FIPS 186-4 Appendix B.3, and additionally runs FIPS checking which includes validating the public key per SP 800-89 5.3.3 and performing a pair-wise consistency check per FIPS 140-2 4.9.2.

For generation of keys for AES-GCM and AES-CBC, 256 random bits are generated internally from the DRBG. The module also allows for key input via API and as such, the module provides no assurance of minimum strength or security of input keys.

The module employs a Validated NIST SP 800-90B entropy source for key generation. The DRBG is without df since the output of the conditioner is full entropy. The NRBG is a draft-SP800-90C XOR Construction NRBG formed by XORing a seed with a DRBG output. The module requests a minimum number of 128 bits of entropy from its Operational Environment per each call. Therefore, the caveat “The module generates cryptographic keys whose strengths are modified by available entropy”, as per IG 7.14, scenario 1b applies.

8.5. Key Storage

The cryptographic module does not perform persistent storage of keys. Keys and CSPs are passed to the module by the calling application, or the calling application instructs the module to generate them. The keys and CSPs are stored in memory in plaintext. Keys and CSPs residing in internally allocated data structures (during the lifetime of a series of API calls) can only be accessed using the module defined API. The JavaScript engine and/or operating system are responsible for protecting memory and process space from unauthorized access.

8.6. Key Zeroization

The module is passed keys as part of a function call from a calling application and does not store keys persistently. The calling application is responsible for parameters passed in and out of the module. The Operating System and the calling application are responsible to zeroize all keys.

The DRBG is zeroed when the module is removed from memory and memory is reclaimed by the OS.

The output data path is provided by the data interfaces and is logically disconnected from processes performing key generation, import, or zeroization. No key information will be output through the data output interface when the module zeroizes keys.

9. Self-tests

9.1 Power-On Self-Tests

Power-on self-tests are run upon the initialization of the module and do not require operator intervention to run. If any of the tests fail, the module will not initialize. The module will enter an error state and no services can be accessed.

9.1.1 Integrity Tests

The module implements an integrity test using a validation HMAC-SHA-256 check. This is implemented by the initialization function being fed the bytes of the module (the WebAssembly bytes) and these are hashed to compare against a previously stored HMAC-SHA-256 (computed under a hardcoded key) which was computed and added to the package at build time.

9.1.2 Known Answer Tests

The module implements known answer tests for:

- AES-GCM (encryption and decryption. Key size: 256-bits)
- AES-GMAC (encryption and decryption. Key size: 256-bits)
- AES-CBC (encryption and decryption. Key size: 256-bits)
- SP 800-90B CTR_DRBG (Key size: 256-bits)
- RSA (signature generation/signature verification and encryption/decryption. Key size: 2048-bit)
- SHA (SHA-256)
- ENT (P) SP800-90B Start-Up Health Tests
    - CHT superset of APT and RCT (Vendor Defined)[3]

9.1.3 On Demand Self Tests

On-demand self-tests may be run by the operator calling the defined API during operation of the module. If any of these tests fail, the module will enter an error state, where no services can be accessed by the operators. The module can be re-initialized to clear the error and resume FIPS mode of operation. When initiated, the Known Answer Tests listed above are run.

9.2 Conditional Self-Tests

These tests are run during operation of the module. If any of these tests fail, the module will enter an error state, where no services can be accessed by the operators. The module can be re-initialized to clear the error and resume operation. The module performs the following conditional self-tests:

[3] Refer to Section 2.2 and 5 in Entropy Report.

- Pairwise Consistency Test - during RSA Key Pair generation
- DRBG Health Tests - Performed on DRBG, per SP 800-90A Section 11.3. Required per IG C.1.
- Continuous RNG Test – Performed on Entropy Input as defined in FIPS 140-2 section 4.9.2

9.3 Failure of Self-Tests

Failure of the self-tests places the cryptographic module in the error state, wherein no cryptographic operations can be performed.

10. Mitigation of Other Attacks

The module is not designed to mitigate against other attacks outside of the scope of FIPS 140-2.

11. Guidance and Secure Operation

The module is intended to be run in a single user operational environment, where each user application and JavaScript 'program' runs in a virtually separated independent space, with access enforcement. Operating systems such as Linux and browsers such as Chrome provide such an operational environment.

To initialize the module, call `initialize(initArray,initArray.length,hash)` from a JavaScript library, and pass in the bytes of the module as the Uint8Array `initArray`, as well as the pre-calculated hash to validate against. Check the return code of this function to ensure it is non-zero.

In order to operate securely, it is the user's responsibility to:

- Manage and protect key material;
- Provide correctly formed and secure high-entropy keys when importing key material;
- Provide accurate data and comparison hash to the initialize function;
- Securely generate and provide IVs/nonces to functions requiring them;
- Avoid reusing key material or IVs/nonces;
- Call `release()` on all keys when no longer needed;
- Zeroize and destroy all plaintext when no longer needed; and
- Destroy the JavaScript engine process and allow the OS to reclaim memory once no longer needed.

12. Acronym List

AES	Advanced Encryption Standard
API	Application Programming Interface
CAVP	Cryptographic Algorithm Validation Program
CO	Cryptographic Officer
DRBG	Deterministic Random Bit Generator
ECC	Elliptic Curve Cryptography
FIPS	Federal Information Processing Standard
IV	Initialization Vector
NDRNG	Non Deterministic Random Number Generator
RNG	Random Number Generator
RSA	Rivest, Shamir, and Adleman Algorithm
WASM	WebAssembly