



Treasure Cryptographic Module FIPS 140-2 Security Policy

Treasure Cloud Pte Ltd.

Prepared by jtsec Beyond IT Security S.L.

Version: 1.14

Table of content

| | | |
|-----|--|----|
| 1 | Introduction..... | 4 |
| 1.1 | Overview..... | 4 |
| 1.2 | Document Organization..... | 4 |
| 2 | Module Specification | 5 |
| 2.1 | Modes of Operation and Security Functions | 5 |
| 2.2 | Block Diagram..... | 6 |
| 2.3 | Critical Security Parameters | 7 |
| 2.4 | Ports and Interfaces..... | 8 |
| 3 | Roles, Authentication and services..... | 9 |
| 3.1 | Roles and Authentication | 9 |
| 3.2 | Services | 9 |
| 4 | Physical Security..... | 11 |
| 5 | Operational Environment..... | 12 |
| 5.1 | Tested Configuration | 12 |
| 5.2 | Operation Rules..... | 12 |
| 6 | Cryptographic Key Management | 13 |
| 6.1 | Random Number Generation..... | 13 |
| 6.2 | Key Generation..... | 13 |
| 6.3 | Key Entry and Output | 13 |
| 6.4 | Key Storage..... | 13 |
| 6.5 | Key Zeroization..... | 13 |
| 7 | Electromagnetic Interference/Electromagnetic Compatibility (EMI/EMC) | 14 |
| 8 | Self-Tests..... | 15 |
| 8.1 | Power-Up Self-Test | 15 |
| 8.2 | Conditional Self-Test..... | 15 |
| 9 | Mitigation of Other Attacks..... | 17 |

| | | |
|------|--|----|
| 10 | Design Assurance | 18 |
| 10.1 | Configuration Management | 18 |
| 10.2 | Configuration Items Identification Method | 18 |
| 11 | Crypto Officer and User Guidance | 19 |
| 11.1 | Secure Distribution | 19 |
| 11.2 | Integrity and Confidentiality Assurance | 19 |
| 11.3 | Installation Instructions and Initialization | 19 |
| 11.4 | Secure operation..... | 22 |
| 12 | Glosary and Abbreviations | 23 |

1 INTRODUCTION

1.1 OVERVIEW

This document is the non-proprietary FIPS 140-2 Security Policy for the *Treasure Cryptographic Module (Software Version 1.5, Hardware Version: Intel Core i7-6600U)*. The *Treasure Cryptographic Module* will also be referred to as “the module” through the document. This Security Policy specifies the security rules under which the module should operate to meet FIPS 140-2 level 1 requirements.

The module is classified by FIPS 140-2 as a software-hybrid, multi-chip standalone embodiment module. The software component of the module is library providing a C-language application programming interface (API) for use by another application which statically links with it. The hardware component of the module is CPU which supports AES-NI instruction set, which is invoked for AES operations performed by the module.

The FIPS 140-2 security levels for the module are as follow:

| Security Requirements | Security Level |
|---|----------------|
| Cryptographic Module Specification | 1 |
| Cryptographic Module Ports and Interfaces | 1 |
| Roles, Services, and Authentication | 1 |
| Finite State Model | 1 |
| Physical Security | 1 |
| Operational Environment | 1 |
| Cryptographic Key Management | 1 |
| EMI/EMC | 1 |
| Self-Tests | 1 |
| Design Assurance | 1 |
| Mitigation of Other Attacks | N/A |

Table 1: Security Requirement

1.2 DOCUMENT ORGANIZATION

This security policy is one part of the FIPS 140-2 submission package. The submission package contains:

- Security policy: This document.
- Algorithm certificates: see Section “2.1 Modes of Operation and Security Functions”.
- Functional specification and design documentation: See Sections “2.2 Block Diagram” and “2.4 Ports and Interfaces” and the document “Treasure FIPS 140-2 Functional Specification-1.2.pdf”
- User guide: See Section “11 Crypto Officer and User Guidance”
- Finite state model: See the document “Treasure FIPS 140-2 Finite State Model-1.2.pdf”
- Configuration item list: See the document “Treasure FIPS 140-2 Configuration Item List-1.12.pdf”

2 MODULE SPECIFICATION

The logical cryptographic boundary of the module is the discrete block of data and instructions generated from the Treasure object module source code; a single file named fipsanister.o. This object module contains the FIPS enabled operations (AES, ECC, HMAC, RSA, cipher_common, digest, rand, and cipher_key_tools) and the modified version of the OpenSSL 2.0.16 source code which has been compiled to run in intel SGX environment and has been modified to support the algorithms provided by ADV (Treasure Data Vault).

The module is to be run on a general-purpose computer that consists of multiple components, so the physical cryptographic boundary is the general-purpose computer where the module is installed. This includes the central processing unit(s), the cache, the main memory, disk drives, network interface cards and peripherals.

2.1 MODES OF OPERATION AND SECURITY FUNCTIONS

The module can only be operated in a FIPS 140-2 Approved mode. The module supports the following Approved security functions:

| Algorithm | Modes | Certificate |
|---------------------|---|-----------------|
| [FIPS 186-4] RSA2 | GenKey(ANSI X9.31, 2048/3072 bits), SigGenPKCS1.5, SigVerPKCS1.5 (2048/3072 and SHA-224 SHA-256, SHA-384, SHA-512 sizes to signature verification). The module also supports GenKey, SigGenPKCS1.5, SigVerPKCS1.5 (4096 bits) but they have not been CAVP tested. | #C608 |
| FIPS 186-2] RSA | SigGenPKCS1.5 (4096 and SHA-224 SHA-256, SHA-384, SHA-512 sizes to signature generation). | #C608 |
| [FIPS 197] AES | 128/192/256-bit [SP 800-38C] CCM, [SP 800-38D] GCM, CBC, CTR, OFB, ECB, CFB | #C608 |
| [FIPS 186-4] ECDSA | <ul style="list-style-type: none"> - GenKey: curves (P-224, P-256, P-384, P-521) - PKV: curves (P-224, P-256, P-384, P-521) - SigGen: (P-224 [SHA-224, SHA-256, SHA-384, SHA-512], P-256 [SHA-224, SHA-256, SHA-384, SHA-512], P-384 [SHA-224, SHA-256, SHA-384, SHA-512], P-512 [SHA-224, SHA-256, SHA-384, SHA-512]) - SigVer: (P-224 [SHA-224, SHA-256, SHA-384, SHA-512], P-256 [SHA-224, SHA-256, SHA-384, SHA-512], P-384 [SHA-224, SHA-256, SHA-384, SHA-512], P-521 [SHA-224, SHA-256, SHA-384, SHA-512]) | #C608 |
| [FIPS 198] HMAC | SHA-1, SHA-224, SHA-256, SHA-384, SHA-512 | #C608 |
| [FIPS 180-4] Digest | SHA-1, SHA-224, SHA-256, SHA-384, SHA-512 | #C608 |
| [SP 800-90] DRBG | Hash DRBG (SHA-1, SHA-224, SHA-256, SHA-384, SHA-512) | #C608 |
| [SP 800-133] CKG | Resulting Symmetric keys and seeds used for asymmetric key generation are an unmodified output from an Approved DRBG. | Vendor Affirmed |

Table 2: Modes of operation and security functions

| Algorithm | Caveat | Use |
|--------------------|--|--|
| RSA Key Wrapping | Provides between 112 and 150 bits of encryption strength | Key Establishment |
| NDRNG ¹ | | Used to provide seed input into the module's Approved DRBG |

Table 3: Non-Approved but Allowed Algorithm Implementations

There are no Non-Approved security functions supported by the module, because the module is always operating in FIPS Approved mode.

2.2 BLOCK DIAGRAM

The following block diagram depicts the information flows between the module and outside equipment through the input/output interfaces defined in section 2.4 Ports and Interfaces.

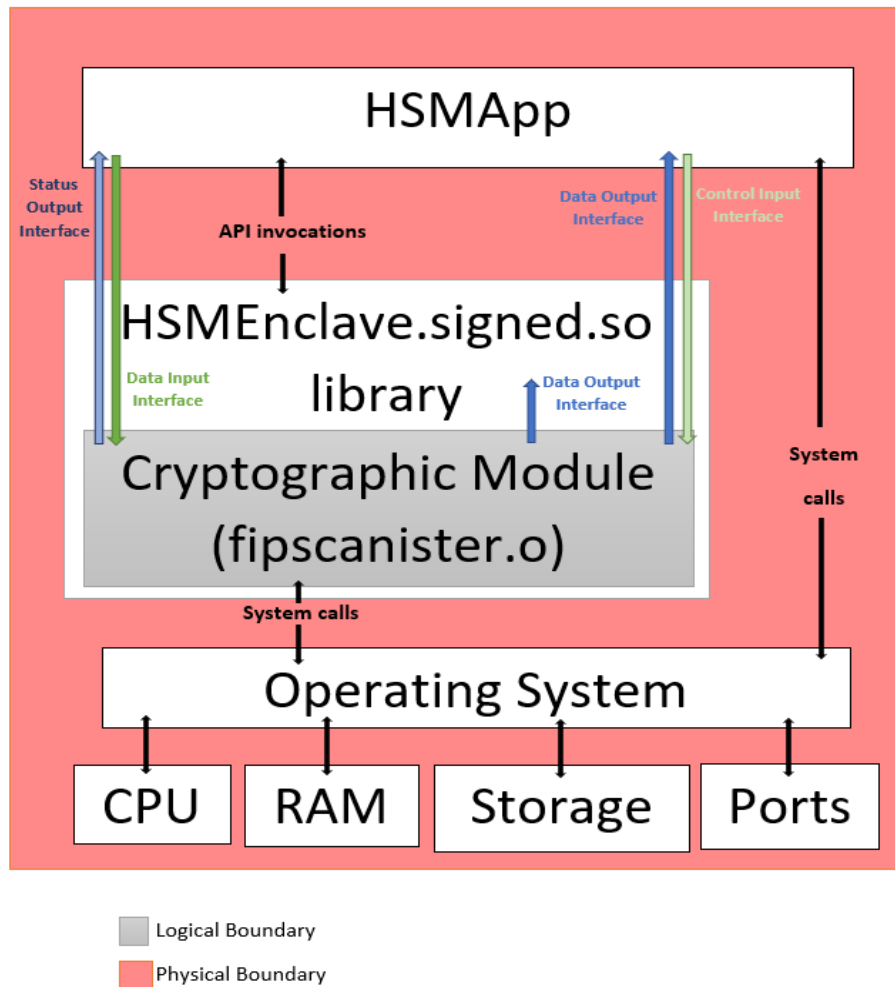


Figure 1: Module block diagram

¹ The module's NDRNG produces an estimated 128 bits of entropy



Figure 2: Picture of the Intel® Core™ i7-6600U Processor

2.3 CRITICAL SECURITY PARAMETERS

Critical Security Parameters (CSPs) encompass all symmetric keys and asymmetric private keys whose disclosure or modification can compromise the security of the cryptographic module. The main responsibility of the protection of the CSPs relies on the host hardware and software which is outside of the boundary of this module using the methodology specified in section “6.4 Key Storage”.

The following table describes the different types of CSPs used by the module and their description:

| CSP | Description |
|----------------|--|
| RSA SGK | RSA (2048/3072/4096 bits) signature generation key |
| RSA DK | RSA (2048/3072/4096 bits) decryption key |
| AES EDK | AES (128/192/256 bits) encrypt/decrypt key |
| HMAC Key | HMAC (256/384/512/1024 bits) key |
| EC SGK | ECDSA (224/256/384/521 bits) signature generation key |
| Hash_DRBG CSPs | V (440/888 bits), C (440/888 bits), entropy input and seed by default depending on the used SHA size |

Table 4: List of CSPs used by the module

The following table shows a complete list of the public keys used by the module:

| Public key | Description |
|------------|-------------|
| | |

| | |
|---------|---|
| RSA SVK | RSA (2048/3072/4096 bits) signature verification key |
| RSA EK | RSA (2048/3072/4096 bits) encryption key |
| EC SVK | ECDSA (224/256/384/521 bits) signature verification key |

Table 5: List of the public keys used by the module

2.4 PORTS AND INTERFACES

The physical ports of the module are the same as the general-purpose computer on which it is executing. The module provides a logical interface via a C-language application program interface (API). FIPS 140-2 interfaces are mapped to the module’s logical interface as follows:

| FIPS-140-2 Interface | Physical Interface | Logical Interface | Description |
|----------------------|---|----------------------------------|---|
| Data input | Network port, Serial port, USB port, SCSI/SATA Controller | Input parameters to API calls | The module has one input interface that is mapped with the API input parameters to all cryptographic functions. |
| Data output | Network port, Serial port, USB port, SCSI/SATA Controller | Output parameters from APO calls | The module has one output interface that is mapped to the API output parameters. |
| Control input | Network port, Serial port, USB port, Power button | API Function calls | The module has one control input interface that is mapped with all API function calls. |
| Status output | Network port, Serial port, USB port, Graphics controller | Return values from API calls | The module has one status output interface which is mapped with all API status output parameters and return codes |
| Power Input | General purpose computer power | Not Applicable | Not Applicable |

Table 6: Module interfaces

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

The control of the physical ports is outside module scope. When the module is performing self-tests, or is in an error state, all data output via the data output interface is inhibited.

3 ROLES, AUTHENTICATION AND SERVICES

3.1 ROLES AND AUTHENTICATION

As it is required, the module supports the User and Crypto Officer roles. Only one role can be active at a time as the module has been built to disable threading (by using the compilation option “no-threads” in build_openssl_fips_module.sh). Therefore, the module does not allow concurrent operators. In addition, the cryptographic module does not support authentication mechanisms.

The following table describes each of the two operator roles support by the module. The roles are implicitly assumed upon the invocation of the module services:

| Role | Authorized Services |
|----------------|--|
| User | All the services, related to the module loading and calling any of the API functions, except the module installation |
| Crypto Officer | Secure installation of the Module on the host computer system and calling of any API functions |

Table 7: Users role and authorized services

3.2 SERVICES

After the crypto officer has performed the module installation, each user (User role and crypto officer role) can use the following services and Keys/CSPs depending its type of access by using the specified API function:

| Authorized Services | Roles | Description | Keys and CSPs | API function | Access |
|-----------------------|----------|--|---|--|--------|
| Module Initialization | CO | Used to initialize the module. | N/A | FIPS_module_mode_set() | N/A |
| Self-test | User, CO | Used to perform self-tests. | N/A | FIPS_selftest() | N/A |
| Generate key | User, CO | Used to generate symmetric and asymmetric keys. | RSA SGK, RSA DK, AES EDK, HMAC Key, EC SGK, RSA SVK, RSA EK, EC SVK | aes_keygen() rsa_gen_keypair() ec_gen_keypair() hmac_keygen() | W |
| Encrypt | User, CO | Used to encrypt a block of data with RSA or AES. | RSA EK, AES EDK | aes_encrypt() aes_ecb_256_encrypt() rsa_public_encrypt() | RX |
| Decrypt | User, CO | Used to decrypt a block of data with RSA or AES | RSA DK, AES EDK | aes_decrypt() aes_ecb_256_decrypt() rsa_private_decrypt() | RX |

| | | | | | |
|--------------------------|----------|---|-----------------|-----------------------------|-----|
| Generate MAC | User, CO | Used to generate HMAC | HMAC Key | hmac_sign() | RX |
| Generate signature | User, CO | Used to generate RSA or EC signatures. | RSA SGK, EC SGK | rsa_sign() ec_sign() | RX |
| Verify MAC | User, CO | Used to verify HMAC | HMAC Key | hmac_verify() | RX |
| Verify signature | User, CO | Used to verify RSA or EC signatures. | RSA SVK, EC SVK | rsa_verify() ec_verify() | RX |
| Digest | User, CO | Used to generate a SHA-1 or SHA-2 message digest. | N/A | get_digest() | N/A |
| Random number generation | User, CO | Used to generate a random number | Hash_DRBG CSPs | rgen() | RWX |
| Zeroize | User, CO | Used to destroy all CSPs. | All CSPs | delete_object() | W |
| Get status | User, CO | Used to get the current status of the module | N/A | FIPS_get_module_state() | N/A |

Table 8: Description of authorized services

4 PHYSICAL SECURITY

The module is a software-hybrid module that operates on a multi-chip standalone platform which conforms to the Level 1 requirements for physical security. The hardware portion is entirely contained within a hard plastic production-grade enclosure which corresponds to the laptop enclosure that surrounds the cryptographic module.

5 OPERATIONAL ENVIRONMENT

The cryptographic module operates on Ubuntu 18.04 LTS 64-bit running on the Intel® Core™ i7-6600U CPU @ 2.60GHz × 4 processor, hence the module's operational environment is modifiable. The operating system segregates user processes into separate process spaces. Each process space is logically separated from all other processes by the operating system software and hardware, preventing unauthorized access from other running processes. The Module functions entirely within the process space of the calling application in a single thread satisfying the requirement for a single user mode of operation.

5.1 TESTED CONFIGURATION

The following table shows the tested configuration that must be deployed in a Lenovo Thinkpad T460s with the following specifications:

| Element | Specifications |
|--------------|---|
| Laptop Model | Lenovo Thinkpad T460s |
| CPU | Intel® Core™ i7-6600U CPU @ 2.60GHz × 4 |
| Memory | 24GB DDR4 |
| Storage | SATA SSD (512 GB) |

Table 9: Tested configuration to be deployed

5.2 OPERATION RULES

Once the application is loaded into memory, the module is initialized to operate in FIPS mode that is its only mode of operation complying with the following rules:

1. The module is initialized in the FIPS mode of operation using the `FIPS_module_mode_set()` function call.
2. The replacement or modification of the module by unauthorized users is prohibited.
3. The operating system enforces authentication method(s) to prevent unauthorized access to the Module services.
4. Before performing any cryptographic operation, the system status must be set to READY which means that the power-up self-test has been successfully completed.
5. The output interface is inhibited if the module state is `FIPS_STATE_POWER_ON`, `FIPS_STATE_SELFTEST` or `FIPS_STATE_ERROR`.
6. All Critical Security Parameters (CSP) are verified as correct and are securely generated, stored and destroyed.
7. All host system components that can contains sensitive cryptographic data (main memory, system bus and disk storage) must be located in a secure environment.
8. The unauthorized reading, writing, or modification of the address space of the Module is prohibited.
9. The operating system is the responsible for multitasking operations so that other processes cannot access the address space of the process containing the Module.
10. The user shall not link multi-threaded applications to the Module API.

6 CRYPTOGRAPHIC KEY MANAGEMENT

6.1 RANDOM NUMBER GENERATION

The module uses an SP 800-90A compliant Hash_DRBG for the generation of random numbers. The module also uses a NDRNG as the source of entropy for DRBG seeds.

To perform the random number generation, the module calls the **RAND_bytes()** function for symmetric and asymmetric key generation.

6.2 KEY GENERATION

The module uses the DRBG defined in Section “6.1 Random Number Generation” for the creation of random data, which is used to generate symmetric keys and seeds for RSA or ECDSA key pair generation. The module does not return intermediate key generation values.

6.3 KEY ENTRY AND OUTPUT

The module does not support manual key entry or intermediate key generation output. The module supports the entry of keys to it via API input parameters in plaintext form. The module also supports the output of keys via API output parameters in plaintext form. The module does not enter or output keys in plaintext format outside of its physical boundary

6.4 KEY STORAGE

Public and private keys are provided to the module by the calling process, and are destroyed when released by the appropriate API function calls. The module does not perform persistent storage of keys.

6.5 KEY ZEROIZATION

After using CSPs, symmetric keys and asymmetric keys, the memory where they are temporally stored is automatically zeroized by calling the **SAFE_FREE()** function which replaces its content with 0s.

The calling application is responsible for calling the **delete_object()** zeroization function, which uses the **compare_header()** to verify that the ID and type of the key matches properly before zeroizing it. The zeroization function overwrite the memory occupied by keys with 0s and deallocates the memory with the regular memory deallocation operating system call.

7 ELECTROMAGNETIC INTERFERENCE/ELECTROMAGNETIC COMPATIBILITY (EMI/EMC)

The software runs in a platform that conforms to the EMI/EMC requirements specified by 47 Code of Federal Regulations, Part 15, Subpart B, Unintentional Radiators, Digital Devices, Class A.

8 SELF-TESTS

8.1 POWER-UP SELF-TEST

The module starts performing the power-up self-test after being loaded into memory and initialized. Once the module starts the performance of the power-up self-tests, it changes states from the “No Operative” state to the “Test” state. The power-up self-tests are composed of an integrity test of the software using HMAC-SHA1 and the KAT (Known Answer Tests), which are a group of tests based on calculating a cryptographic value and comparing it with a stored previously determined answer. In the case of ECDSA, it is tested using a pair-wise consistency test.

To execute the power-up self-test, the module calls the **FIPS_selftest()** function which performs the integrity test of the module and the KAT for each of the Approved algorithms detailed in the following table:

| Algorithm | Description |
|----------------|--|
| Integrity test | Software integrity test using HMAC-SHA-1. The associated function to this test is FIPS_check_incore_fingerprint() . |
| AES | Known answer test. Separate encryption and decryption test using a 128 bits key and ECB mode. The associated function to this test is FIPS_selftest_aes() . |
| AES CCM | Known answer test. Separate encryption and decryption test using a 192 bits key. The associated function to this test is FIPS_selftest_aes_ccm() . |
| AES GCM | Known answer test. Separate encryption and decryption test using a 256 bits key. The associated function to this test is FIPS_selftest_aes_gcm() . |
| RSA | Known answer test. By signing and verifying with 2048 bits key and SHA-256. The associated function to this test is FIPS_selftest_rsa() . |
| ECDSA | Pair-wise consistency test. By signing and verifying using a P-224 curve with SHA-512. The associated function to this test is FIPS_selftest_ecdsa() . |
| HMAC | Known answer test for HMAC-SHA-1, HMAC-SHA-224, HMAC-SHA-256, HMAC-SHA-384 and HMAC-SHA-512. The associated function to this test is FIPS_selftest_hmac() . |
| SHA-1 | Known answer test. Message digest using SHA-1. The associated function to this test is FIPS_selftest_sha1() . All other SHS implementations are test as a part of the HMAC KAT. |
| DRBG | Known answer test. SP 800-90A HASH_DRBG: SHA-1, SHA-224, SHA-256, SHA-384 and SHA-512. The associated function to this test is FIPS_selftest_drbg() . |

Table 10: Power-Up self-test description

Power-on self tests return 1 if all self tests succeed, and 0 if not. If a self-test fails, the module enters the error state and all data output is inhibited. During self-tests, cryptographic functions cannot be performed until the tests are complete. If a self-test fails, subsequent invocation of any cryptographic function calls will fail. The only way to recover from a self-test failure is by power-cycling the module.

8.2 CONDITIONAL SELF-TEST

The module performs the following conditional self-tests:

| Algorithm | Description |
|-----------|-------------|
| | |

| | |
|-----------|---|
| RSA | Pairwise consistency test for both Sign/Verify and Encrypt/Decrypt. |
| ECDSA | Pairwise consistency test for Sign/Verify. |
| Hash_DRBG | -Continuous random number generator test as specified by SP 800-90A. - SP 800-90A DRBG Health Tests: Instantiate, Reseed, Generate, and Uninstantiate. |
| NDRNG | Continuous Random Number Generation Test |

Table 11: Conditional self-test description

In the event of a DRBG self-test failure the calling application must uninstantiate and re-instantiate the DRBG per SP 800-90A requirements.

9 MITIGATION OF OTHER ATTACKS

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

10 DESIGN ASSURANCE

10.1 CONFIGURATION MANAGEMENT

Configuration management for the module's source code is provided by the Git source code management system. Git provides configuration item version control, change control, flaw remediation tracking and the tracking of source code revisions. The source code is maintained in a private Git repository with write access restricted to authorized developers.

10.2 CONFIGURATION ITEMS IDENTIFICATION METHOD

The internal versioning of the source code files is performed by Git automatically and the assigned version and revision are used internally to control the code development, so that it must not be confused with the final released version of the code that is assigned manually to each source file to allow the customer to identify the version. The version will be assigned with the following format "Version: X.Y (Date)", where X is the version number, Y is the revision number and Date is the release date of the module.

Regarding each associated module documentation, they are manually versioned by appending the version and revision to their filename as follow: Document-X.Y.

Each associated module documentation file is manually assigned a version number which is stated as part of the file name which uses the following naming convention:

- Naming: Name-X.Y, where Name is the unique name of the related document, and X.Y is the version and revision of the document. Every new document starts with version v1.0.
- Version Update: When the document is modified and this modification implies major changes, then the X number is incremented. However, if changes and modifications imply minor changes, then the Y number is incremented.

11 CRYPTO OFFICER AND USER GUIDANCE

11.1 SECURE DISTRIBUTION

The Treasure Data Vault application, which the module is integrated in, is available for download from the vendor website over HTTPS. A hash and signature of the download are provided to allow the customer to verify the integrity of the application before proceeding with the secure installation and configuration.

11.2 INTEGRITY AND CONFIDENTIALITY ASSURANCE

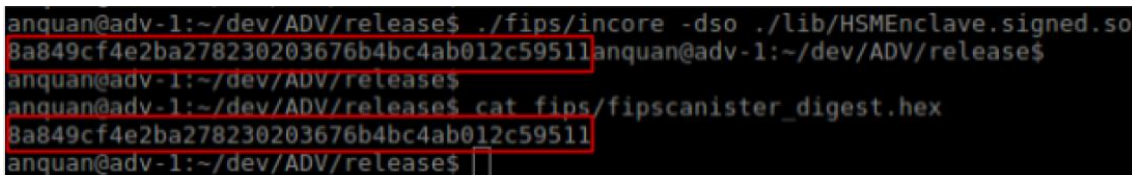
Due to the module being contained within the Treasure Data Vault Application, the only way to verify its integrity is during the first run of the application.

As it is specified in the section below, during the build process of the HSMApp, the fipsanister signature is stored in the fipsanister_digest.hex file which will be used to verify the integrity of the module.

To perform the validation, the crypto officer needs to use the incore utility to generate the signature of the fipsanister inside the HSMEnclave.signed.so by running the following command being in the 'release' directory:

```
./fips/incore -dso ./lib/HSMEnclave.signed.so.
```

Once the crypto officer has obtained the signature of the fipsanister inside the HSMEnclave.signed.so, he can compare it with the content of the fipsanister_digest.hex file generated during the build process:



```
anquan@adv-1:~/dev/ADV/release$ ./fips/incore -dso ./lib/HSMEnclave.signed.so
8a849cf4e2ba278230203676b4bc4ab012c59511anquan@adv-1:~/dev/ADV/release$
anquan@adv-1:~/dev/ADV/release$
anquan@adv-1:~/dev/ADV/release$ cat fips/fipsanister_digest.hex
8a849cf4e2ba278230203676b4bc4ab012c59511
anquan@adv-1:~/dev/ADV/release$
```

Figure 3: Integrity verification of the module

11.3 INSTALLATION INSTRUCTIONS AND INITIALIZATION

This section details the steps that must be followed by the Crypto Officer to proceed with the secure installation and initialization of the module after checking that its version is 1.5.

As is detailed in section “2 Module Specification” the fipsanister.o is composed of FIPS enabled ciphers and the OpenSSL source code which complies with the standard FIPS 140-2. The fipsanister.o is part of the Treasure Data Vault library (HSMEnclave.signed.so).

The ADV application has some dependencies that must be installed prior to proceeding with the application installation, this is because ADV is a security service that runs on Intel’s Software Guard Extension (SGX) technology. The needed dependencies are listed below:

1. Intel SGX SDK for Linux (<https://github.com/01org/linux-sgx>)
2. Intel SGX driver for Linux (<https://github.com/01org/linux-sgx-driver>)
3. FIPS Enabled SGXSSL for Linux (OpenSSL 1.0.2q ported by Treasure for Intel SGX Environment)
4. Boost C++ Library (<http://www.boost.org>) used for logging, encoding/decoding, communication, etc.

The following command and instruction must be used to proceed with the installation of the listed dependencies:

SGX SDK and SGX driver installation:

Step 1: Configure the system with the SGX hardware enabled option.

Step 2: Check if matching kernel headers are installed:

```
dpkg-query -s linux-headers-$(uname -r)
```

Step 3: Install matching headers:

```
sudo apt-get install linux-headers-$(uname -r)
```

Step 4: Build the Intel SGX driver by executing the following command in the driver path:

```
make
```

Step 5: Install the intel SGX driver with:

```
sudo mkdir -p "/lib/modules/"`uname -r`"/kernel/drivers/intel/sgx"  
sudo cp isgx.ko "/lib/modules/"`uname -r`"/kernel/drivers/intel/sgx"  
sudo sh -c "cat /etc/modules | grep -Fxq isgx || echo isgx >> /etc/modules"  
sudo /sbin/depmod  
sudo /sbin/modprobe isgx
```

Step 6: Install the required tools to build the Intel(R) SGX SDK:

```
sudo apt-get install build-essential ocaml automake autoconf libtool wget python libssl-dev  
sudo apt-get install libssl-dev libcurl4-openssl-dev protobuf-compiler libprotobuf-dev debhelper
```

Step 7: Download the prebuilt binaries by running the download_prebuilt.sh script:

```
./download_prebuilt.sh
```

Step 8: Build the Intel SGX SDK and SGX PSW:

```
make
```

Step 9: Build the Intel SGX SDK Installer

```
make sdk_install_pkg
```

Step 10: Build the Intel SGX PSW Installer

```
make deb_pkg  
make deb_sgx_enclave_common_dev_pkg
```

Step 11: Install the required tool to use Intel(R) SGX SDK:

```
sudo apt-get install build-essential python
```

Step 12: Install the Intel SGX SDK:

```
source ${sgx-sdk-install-path}/environment
```

```
cd linux/installer/bin  
./sgx_linux_x64_sdk_${version}.bin
```

Step 13: Install the prerequisites for SGX PSW:

```
sudo apt-get install libssl-dev libcurl4-openssl-dev libprotobuf-dev
```

Step 14: Install the Intel SGX PSW:

```
cd linux/installer/deb  
sudo dpkg -i ./libsgx-urts_${version}-${revision}_amd64.deb ./libsgx-enclave-common_${version}-${revision}_amd64.deb
```

Step 15: Download source code from dynamic-application-loader-host-interface project. In the source code folder, build and install the JHI service using the following commands

```
sudo apt-get install uuid-dev libxml2-dev cmake pkg-config libsystemd-dev cmake .;make;sudo make  
install;sudo systemctl enable jhi
```

FIPS Enabled SGXSSL for Linux:

Step 16: After installing the Intel SGX SDK, Intel PSW, Intel SGX driver and PERL. Download OpenSSL package into openssl_source/ directory. (tar.gz package, e.g. openssl-1.0.2q.tar.gz)

Step 17: cd to Linux/ directory and run the following commands to install Intel SGXSSL libraries:

```
make all  
sudo make install
```

Boost and C++ Library installation:

Step 18: Install the library:

```
sudo apt-get install build-essential libboost-all-dev libc++-dev libc++unit-dev
```

After installing the dependencies, the module can be built and installed, however it's important to consider that the module will be compiled and installed internally during the build and installation of the HSMAApp.

The HSMAApp can be installed in three different modes: pre-release mode, debug mode or production mode. When built in production mode, the signing material (hash of HSMEnclave) has been generated using the Intel SGX tool (sgx_sign) to sign the enclave software and then the signature is attached to the HSMEnclave.so to generate the HSMEnclave.signed.so (This process can only be performed by whitelisted signers as the vendor). In addition, this signing process is only necessary to ensure the security of the enclave software, thus it does not affect to the module operation and security.

To proceed with the build and installation using the production mode, it is necessary to perform the following steps:

Step 19: Generate the signing material by running the following script:

```
./build_fips.sh clean prod
```

Step 20: At this point, the signing material (HSMEnclave_hash.hex) has been generated in ./release/lib. Now bring it to the signing machine which contains the signer's key that has been whitelisted by Intel. Sign the signing material/s (HSMEnclave_signature.hex) and then bring it back to this build machine (./release/lib). Then run the production release script:

```
./prod_release.sh <pubkey> <hsm_signature> <ra_signature>
```

This script will build and install the HSMApp application, HSMEnclave.signed.so library, which contains the module, and the FIPS Validation Suite. When it is executed, the HSMEnclave.signed.so (in this case the HSMEnclave.so) is loaded with all its libraries (ADV Ciphers, OpenSSL source code compliant with FIPS 140-2, Intel SGX libraries, SGXSSL, etc.) and the included fips_premain.c which will be used to perform the FIPS integrity test.

During the build process, the HSMApp computes the fingerprint of fipsanister.o and writes the signature file in the variable HMAC_SHA1_SIG defined in the fips_premain.c and also creates a file named as fipsanister_digest.hex to store its value.

Once the signature file has been generated, the build_fips.sh script passes it to the second build process to set it when the HSMApp loads the HSMEnclave.signed.so (in this case the HSMEnclave.so).

Finally, the HSMEnclave.signed.so (in this case the HSMEnclave.so) is loaded and initializes the module in FIPS mode by executing the **fips_set_mode()** function which starts the power-up self-test by calling the **FIPS_selftest()** function defined in the fips_post.c file.

11.4 SECURE OPERATION

After the Crypto Officer installs and initializes the module as described in the previous section, if all the power-up self-test (listed in section “8.1 Power-Up Self-Test”) are performed successfully, then the module will be in READY state and will allow the User and Crypto Officer to use the authorized services and the API functions detailed in the section “3.2 Services” related to any cryptographic operation, key zeroization, obtaining the module status or performing a self-test.

To interact with the module, the User and Crypto Officer can use the defined ports in the section “2.4 Ports and Interfaces” without any additional measure or especial behavior, due to the module is always operating in FIPS mode and, in addition, it does not return any private secret or key component.

Apart from that, the module generates IVs internally using the Approved DRBG which are at least 96-bits in length. In the event that module power is lost and restored, the calling application must ensure that any AES-GCM keys used for encryption or decryption are re-distributed.

12 GLOSARY AND ABBREVIATIONS

| | |
|--------------|--|
| ADV | Treasure Data Vault |
| AES | Advanced Encryption Standard |
| API | Application Programming Interface |
| CBC | Cipher Block Chaining |
| CFB | Cipher Feedback |
| CO | Crypto Officer |
| CPU | Central processing Unit |
| CSP | Critical Security Parameter |
| CTR | Counter |
| DK | Decryption Key |
| DRBG | Deterministic Random Bit Generator |
| ECB | Electronic Codebook |
| ECC | Elliptic Curve Cryptographic |
| ECDSA | Elliptic Curve Digital Signature Algorithm |
| EDK | Encrypt/Decrypt Key |
| EMI | ELECTROMAGNETIC INTERFERENCE |
| EMC | ELECTROMAGNETIC COMPATIBILITY |
| FSM | Finite State Model |
| GCM | Galois/Counter Mode |
| HMAC | Hash-based Message Authentication Code |
| KAT | Known Answer Test |
| NDRNG | Non-Deterministic Random Number Generator |
| OFB | Output Feedback |
| RAM | Random Access Memory |
| SHA | Secure Hash Algorithm |
| SGK | Signature Generation Key |
| SGX | Software Guard Extension |
| SVK | Signature Verification Key |
| URI | Uniform Resource Identifier |