V-Key Cryptographic Module

FIPS 140-2 Non-Proprietary Security Policy
Level 1 Validation

V-Key Pte. Ltd.

Version 2.3.16
02 August, 2016

Table of Contents

Revision History

Date	Revision	Description
July 08, 2015	2.3.1	Divided services (Section 4.2) into Approved, Allowed and Non-Approved services.
July 22, 2015	2.3.2	Modified description of protection mechanisms for the keystore (Section 9.5) to be consistent with the implementation. Updated the state diagram (B.1) and state transition conditions and events (B.3). Modified description of error state (B2.6) to be consistent with the implementation.
July 30, 2015	2.3.3	Reviewed and updated Table 4. Updated state diagram and the transitions Updated information about Allowed methods for key transport.

Date	Revision	Description
August 12, 2015	2.3.4	Revised Figure 1 for cryptographic boundary. Added information for Random Number Generator about the AES implementation and seeds. Added detail about configuration items Added distinction between VM_KDF_HMAC_SHA1 and VM_KDF_HMAC_SHA256. Detailed information about Error State and Exit State.
August 18, 2015	2.3.5	Added information about Target Audience Revised wording for RNG (Section 9.2) and caveat about RSA encryption strength (Section 9.3). Revised Figure 1. Revised Section 3. Corrected some typos.
August 31, 2015	2.3.6	Minor change in Introduction Moved caveat about RSA encryption strength from Section 9.3 to Table 2 Added clarification in B.3 for the output event of self-tests Revised module ports and interface in Section 3. Revised Figure 1 and its description in Section 2.1. Revised Table 1 and Table 3 for Triple-DES after enabling testing with three identical keys.

Date	Revision	Description
September 9, 2015	2.3.7	Revised Section 2.3 and 2.4. Revised API in Tables of Section 4.2. Updated information about test environment in Section 2.6. Some minor changes.
October 1, 2015	2.3.8	Detailed B.2.4 with information about data and control interface. Added “Show Status” to Table 4. Revised description for Storage Root Key. Moved password-based key derivation from Table 1 to Table 3. Added changes due to support for RSA key size up to 4096 bits.
October 21, 2015	2.3.9	Revised description for Storage Root Key to add password-based protection. Revised Appendix A for PBKDF2, RSA and KDF.
December 17, 2015	2.3.10	Added VM_SYS_INIT to Table 1. Detailed information about seed generation in Section 9.2. Added Apple iPhone 5S running on iOS 7.0.4 to Test Environment. Detailed information about key zeroization in Section 9.6
December 24, 2015	2.3.11	Moved RNG from Table 1 to Table 3. Various changes due to transition of ANSI X9.31 based RNGs into disallowed status. Added a section for Security Rules.
May 12, 2016	2.3.12	Added Table 5 as CSP summary.

Date	Revision	Description
		Added clarification that the non-approved RNG cannot be used for key generation. Added reference to Appendix A in Table 1. Removed Triple-DES Two Key entry from Table 1, removed Triple-DES and RSA entries from Table 3. Added clarification about allowed RSA key sizes. Removed PBKDF2 from Table 4. Modified Section 2.5 about Modes of Operation. Modified Section 9.5 about key storage Some minor format and wording changes.
May 25, 2016	2.3.13	Removed references to AES/Triple-DES key wrapping and unwrapping in Section 9.3 and Table 5.
July 10, 2016	2.3.14	Renamed Table 3, renamed KDF to KBKDF. Revised CSP Table in Section 9. Revised roles for operators. Added clarification for Test Environment. Added clarification regarding RNG and key generation.
July 15, 2016	2.3.15	Added clarification regarding requirement for no source code modification in porting the Module to another environment. Changed “hashing key” to “derivation key” as type of KBKDF key in Table 5.
August 02, 2016	2.3.16	Added clarification regarding key transport

Date	Revision	Description
		in Section 9.3 by referencing Table 2. Added clarification regarding services outside of the cryptographic boundary in Table 3. Added version information of the module.

1. INTRODUCTION

The V-Key cryptographic module (the “Module”) is a software module residing within a virtual machine, V-OS, which provides a secure sandboxed operating environment for the Module. The Module is designed to run within this sandbox to provide separation from hosting operating system in order to secure the critical security parameters (CSPs) processed and stored within Module.

The V-Key cryptographic module is based on the CTaoCrypt cryptography library used by CyaSSL [1], and enhances them using the protecting sandboxed operating environment.

This is the security policy documentation for the V-Key cryptographic module to meet FIPS 140-2 Level 1 requirements.

1.1. Audience

This security policy describes the V-Key cryptographic module with respect to FIPS 140-2 requirements, and is required as a part of the FIPS 140-2 validation process. The companion document, V-Key User Guide, is a reference for Crypto Officers provisioning and administrating the V-Key cryptographic module for use by end-users, and a reference for Users developing secure mobile apps by utilizing the V-Key cryptographic module.

The security policy is intended for the following audience

- Developers working on the release
- NVLAP Accredited FIPS 140-2 testing laboratories
- NIST - Cryptographic Module Validation Program (CMVP)
- Administrators and Users of the cryptographic module.

1.2. Document Organization

This Security Policy document is one part of the complete V-Key FIPS 140-2 Submission Package. The Submission Package contains:

- *FIPS 140-2 Security Policy, v2.3.15*: this document
- Design Specifications and Functional Specifications: See *V-Key User Guide, v2.3.10*
- Crypto Officer and User Guidance documentation: See *V-Key User Guide, v2.3.10*
- Algorithm Certificates: See Appendix A of this document

- Finite State Machine: See Appendix B of this document.

2. MODULE SPECIFICATION

The V-Key cryptographic module is a software module that provides confidentiality, integrity, and message digest services. The Module has been designed and implemented to meet FIPS 140-2 Level 1 requirements.

2.1. Cryptographic Boundary

The Module (Software Version 3.6.0 as crypto_kernel.bin) is a software-only cryptographic library, but for the purposes of the FIPS 140-2 validation, it is considered a multiple-chip standalone cryptographic module.

The Module is designed to run on V-OS Virtual Machine to provide separation from the mobile operating system. The logical cryptographic boundary for the Module contains the V-Key Cryptographic Module and a reduced operating system (see Figure 1).

The Wrapper API (which is specified in the User Guide) serves as a wrapper layer on top of V-Key Cryptographic Module. It is not part of the logical cryptographic boundary. For ease of implementation without security impact, it provides API for calling applications (e.g. mobile apps) under iOS and Android respectively.

V-OS Interface serves as an interface between the Wrapper and the V-Key cryptographic module. It translates Wrapper API calls into function calls of V-Key cryptographic module running within the virtual machine, whereas it provides all parameters of a function call via the stack of the virtual machine to the Module.

The Module is programmed in C, then compiled and assembled into obfuscated microcode (i.e. obfuscated virtual machine instructions). The virtual machine interpreter interprets the microcode and acts as a protecting sandboxed operating environment on top of hosting mobile operating systems.

The physical cryptographic boundary for the Module is defined by the mobile phone hardware of the system executing the application. This system hardware includes the central processing unit(s), cache and main memory (RAM), system bus, and ports to peripherals including disk drives and permanent mass storage devices, network interface cards, keyboard and console and any terminal devices.

2.2. Module Block Diagram

The block diagram in Figure 1 shows the physical and logical boundary of the Module. The dashed line shows the logical cryptographic boundary for the Module.

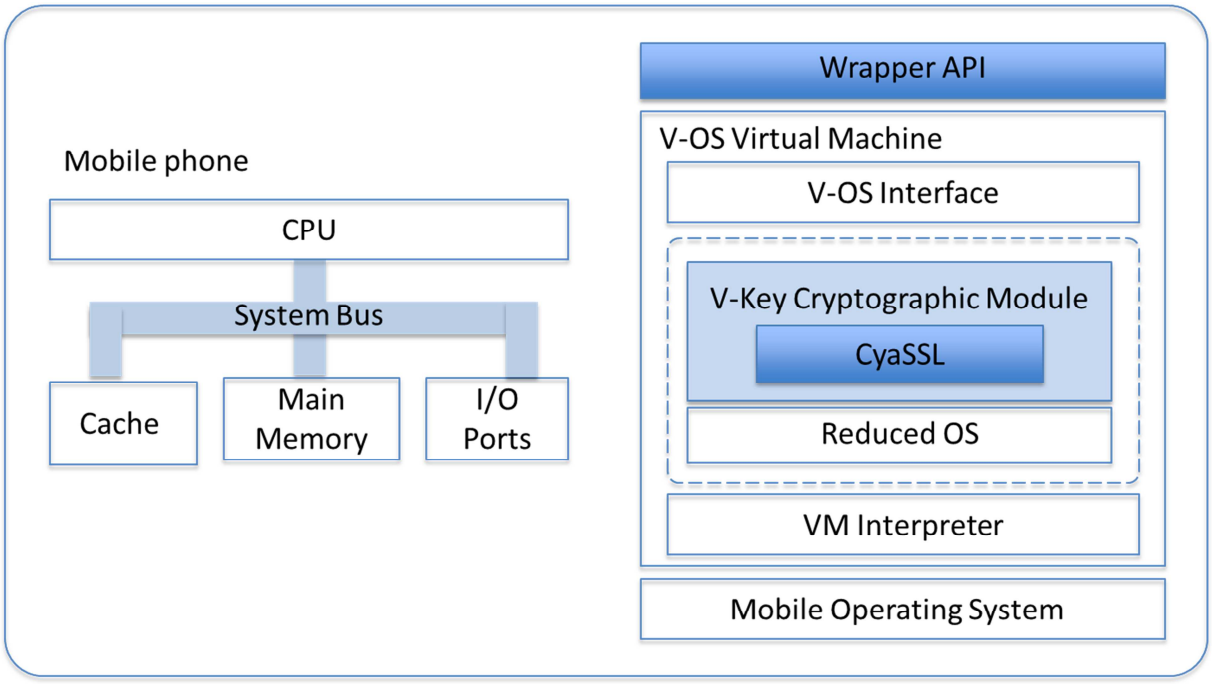


Figure 1 Physical and logical boundary of the V-Key cryptographic module

2.3. Approved Cryptographic Algorithms

The list of implemented Approved cryptographic algorithms and security functions are listed in Table 1 of Section 4.2. Not Approved but Allowed algorithms and security functions are listed in Table 2 of Section 4.2.

2.4. Non-Approved Cryptographic Algorithms

Non-Approved cryptographic services provided by the V-Key cryptographic module are listed in Table 3 of Section 4.2. Performing these services using the Module will invalidate FIPS compliance.

2.5. Modes of Operation

The Module supports a FIPS Approved mode of operation and a non-FIPS Approved mode of operation. The Module always boots into the Approved mode of operation and runs Module's self-tests.

The Module is in the Approved mode of operation when any of the Approved or Not-Approved but Allowed cryptographic functions (as listed in Table 1 and Table 2) are invoked by the calling application.

The Module is in the non-FIPS Approved mode of operation, when any of the Non-Approved cryptographic functions (as listed in Table 3) are invoked by the calling application.

Critical Security Parameters (CSPs) are not impacted by Non-Approved cryptographic functions, since CSPs are not accessed by these functions. Transitions to or from the FIPS Approved mode of operation do not require re-execution of the Module's self-tests.

2.6. Test Environment

As allowed by FIPS 140-2 Implementation Guidance G.5 (IG G.5), the validation status of the Cryptographic Module is maintained when operated on any general purpose computer (GPC) provided that the GPC uses the specified single-user operating system on the validation certificate, or another compatible single-user operating system such as those internally tested by V-Key which are listed below. According to IG G.5, this is only applicable if no source code modification (e.g., changes, additions, or deletions of code) is required to recompile the Cryptographic Module for porting to another operational environment.

The Module was tested by the Security Testing Laboratory (CST) on a Samsung Galaxy S4 running Android 4.4.2 with V-OS 3.6.0.

The Module was installed and internally tested by V-Key on the following computer systems. V-OS was installed on all platforms, with the V-Key Cryptographic Module operating in the V-OS Virtual Machine.

- Apple iPhone 4 running on iOS 6.0.1
- Apple iPhone 4S running on iOS 6.1.3
- Apple iPhone 5 running on iOS 6.1.4
- Apple iPhone 5C running on iOS 7.0.6
- Apple iPhone 5S running on iOS 7.0.4, iOS 7.1 and iOS 7.1.1
- Apple iPhone 6 running on iOS 8.0
- Samsung Note 3 running on Android 4.3
- Samsung Galaxy S4 and S5 running on Android 4.4.2
- HTC One M8 running on Android 4.4.2

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

3. MODULE PORTS AND INTERFACES

The V-Key cryptographic module is a software cryptographic implementation running on top of a virtual machine. No hardware or firmware components are included. The physical ports are those of conventional mobile devices it runs on, for instance, keyboard and monitor. The following describes the logical interfaces.

Data Input	All data input to be processed by the Module is passed via the stack to the Module.
Data Output	All data output returned to the caller as updated data in memory pointed to by some of the parameters.
Control Input	All control input to be processed by the Module is passed via the stack to the Module.
Status Output	All status output returned to the caller as return codes.

4. ACCESS CONTROL, ROLES, SERVICES AND AUTHENTICATION

4.1. Roles

The Module supports a User role and a Crypto Officer role. The User role is assumed by the local user of the Module. This role can access all services through calling API provided by the Module except module installation and uninstallation. The Crypto Officer role is supported for the installation and uninstallation of the module. The Crypto Officer role can perform Module Power On, Self-Tests and query Module's Status.

Role	**Authorized Services**
User	All except module installation and uninstallation.
Crypto Officer	Module installation and uninstallation, Module Power On, Self-Tests and Show Status

The Module does not support a *Maintenance* role which is assumed to perform physical maintenance and/or logical maintenance services (e.g., hardware/software diagnostics).

4.2. Services

The services provided by the Module are listed in Table 1 for Approved algorithms and key lengths, in Table 2 for Not Approved but Allowed algorithms and key lengths, and in Table 3 for Non-Approved algorithms and key lengths. The *API Reference* in the document *V-Key User Guide* provides detailed description of purpose, inputs and outputs of each service. All services listed in Table 1 and Table 2 are for the user role.

Service Type	Algorithm – Refer Appendix A for Details	NIST/FIPS Publication	API Functions
Symmetric Key	AES	FIPS 197	VM_AES_ECB_ENC VM_AES_ECB_DEC VM_AES_CBC_ENC VM_AES_CBC_DEC VM_AES_ECB_ENC_ALIAS VM_AES_ECB_DEC_ALIAS VM_AES_CBC_ENC_ALIAS VM_AES_CBC_DEC_ALIAS Encryption and decryption in ECB and CBC mode. Cryptographic keys of 128, 192 and 256 bits are supported.
	Triple-DES	FIPS 46-3	VM_DES3_CBC_ENC VM_DES3_CBC_DEC VM_DES3_CBC_ENC_ALIAS VM_DES3_CBC_DEC_ALIAS Three-Key Triple DES: A key bundle shall consist of three different keys.
Asymmetric Key	RSA	FIPS 186-4 NIST SP800-56B NIST SP800-131A	VM_RSA_SHA1_VERIFY_ALIAS VM_RSA_SHA256_SIGN_ALIAS VM_RSA_SHA256_VERIFY_ALIAS RSA signature generation and signature verification. As padding schemes, RSASSA-PKCS1-v1.5 is supported for signing. For signature generation, key lengths of minimum 2048 bits and maximum 3072 bits must be used. For signature verification, key length of 1024 bits is allowed for legacy use.
Secure Hash Standard	SHA-1 SHA-256	FIPS 180-4	VM_SHA1 VM_SHA256
Message	HMAC-	FIPS 198	VM_HMAC_SHA1 VM_HMAC_SHA256

Service Type	Algorithm – Refer Appendix A for Details	NIST/FIPS Publication	API Functions
Authentication	SHA1 HMAC-SHA256		VM_HMAC_SHA1_ALIAS VM_HMAC_SHA256_ALIAS Key length must be ≥ 112 bits.
Key Establishment	Key Derivation Using Pseudo-random Functions (KBKDF)	NIST SP800-108	VM_KDF_HMAC_SHA1 VM_KDF_HMAC_SHA256 VM_KDF_HMAC_SHA1_ALIAS VM_KDF_HMAC_SHA256_ALIAS Derives a key from an existing key based on HMAC-SHA1 or HMAC-SHA256 in Counter mode with location of counter before fixed input data.
Module Power-On	N/A	N/A	VM_SYS_INIT Starts the Module within the V-OS virtual machine, and triggers power-on self-tests. The function returns either an error code or a non-negative return code indicating that self-tests are successful.
Self-Tests	N/A	N/A	VM_SELF_TEST VM_DO_MODULE_INTEGRITY_TEST Performs Module's self-tests and integrity test on power-up and on demand, and returns the outcome of self-tests.
Show Status	N/A	N/A	VM_GET_POWERUPTEST_STATUS Shows the outcome of Module's self-tests and integrity tests. With any failure in Module's self-tests or cryptographic operations, the Module exits. Otherwise, the Module is in the Normal User State.

Table 1 Approved cryptographic services provided by the V-Key cryptographic module

Service Type	Algorithm	NIST Publication	API Functions
Asymmetric Key	RSA	FIPS 140-2 Implementation Guidance IG 7.1 NIST SP800-56B Section 7.2.2 for RSA OAEP	VM_RSA_PUBKEY_ENC_ALIAS VM_RSA_PRIKEY_DEC_ALIAS VM_RSA_OAEP_ENC_ALIAS VM_RSA_OAEP_DEC_ALIAS Using key size of 2048 or 3072 bits, they can be used in Allowed methods for key transport[1]. RSA (key wrapping; key establishment) methodology provides between 112 and 128 bits of encryption strength.

Table 2 Allowed cryptographic services provided by the V-Key cryptographic module

[1] Allowed methods for key transport are “Any key encapsulation scheme employing an RSA-based key methodology that uses key lengths specified in SP 800-131A as acceptable or deprecated.” (FIPS 140-2 Implementation Guidance IG 7.1). RSA-OAEP is implemented as specified in SP 800-56B section 7.2.2.

Service Type	Algorithm	NIST Publication	API Functions
Random Number Generators	ANSI X9.31 Appendix A.2.4 using AES with a 128-bit key.	See FIPS 140-2 Annex C: Approved Random Number Generators	VM_RNG_BYTES VM_FIPS_RNG_BYTES
Key Establishment	Password-Based Key Derivation Function 2 (PBKDF2)	NIST SP800-132, see FIPS 140-2 Annex D: Approved Key Establishment Techniques	VM_KDF_PBKDF2 Generates a key of the specified length based on the given password and salt. This service is performed outside of the cryptographic boundary of the Module[2].

Table 3 Non-Approved and not-allowed cryptographic services provided by the V-Key cryptographic module

Table 4 identifies for each service the CSPs, type of available accesses and authorized role(s) in which the service can be performed.

Role	Service	Critical Security Parameters	Types of Access
User	AES encryption and decryption	Secret symmetric key	Write Execute
User	Triple-DES encryption and decryption	Secret symmetric key	Write Execute
User	RSA decryption and signature generation	Private key	Write Execute

[2] The Module provides access to this service, however the actual service is performed outside of the cryptographic boundary of the Module. The Module only passes parameters provided in the call to V-OS to perform the processing.

Role	Service	Critical Security Parameters	Types of Access
User	RSA encryption and signature verification	Public key	Write Execute
User	SHA hash generation	N/A	N/A
User	HMAC-SHA1, HMAC-SHA256 generation	Secret key	Write Execute
User	Key Derivation Using Pseudorandom Functions	Key derivation key (the input key), resulting derived key, input data containing information related to the derived key.	Write Execute
Crypto Officer/User	Self-Tests	Module Integrity Key	Execute
Crypto Officer/User	Show Status	N/A	Execute
Crypto Officer/User	Module Power-On	Module Integrity Key	Execute

Table 4 CSPs, access types, and authorized roles of provided services

4.3. Identification and Authentication

Within the constraints of FIPS 140-2 Level 1, the Module does not implement user authentication. Roles are implicitly selected by the operator based on the API executed. Implicit role selection is summarized in Table 4.

5. FINITE STATE MODEL

The Module implements the finite state model detailed in Appendix B.

6. PHYSICAL SECURITY

Not applicable, as the Module is implemented entirely in software. The physical security is provided solely by the host platform, and as such the Module is not subject to the physical security requirements of the FIPS 140-2 standard.

7. OPERATIONAL ENVIRONMENT

The V-Key cryptographic module runs inside a virtual machine, V-OS, which runs on top of the mobile operating system iOS and Android. All cryptographic software, cryptographic keys and CSPs, and control and status information of the Module are under the control of the virtual machine.

The Module is restricted to a Single Operator Mode of Operation. It is intended to be used in a client environment, where each client application is associated with exactly one instance of the Module. Thus, under the control of V-OS, each instance of the Module can only be accessed by one application process. This prevents access by other processes to plaintext private and secret keys, CSPs, and intermediate key generation values during the time the cryptographic module is operational. Additionally, all CSPs generated upon Module provision never leave the V-OS sandbox as plaintext.

The integrity of the Module is verified through the Power-Up Self-Test described in Section 9 “Self-Tests”. To further protect the Module from unauthorized disclosure and modification, the microcode of the Module is encrypted and it is only decrypted into the memory space of the virtual machine at runtime.

Upon module uninstallation, the microcode of the Module and the encrypted keystore file in the file system shall be wiped by the end user to prevent reverse engineering and exposure of CSPs. This, combined with a proprietary set of virtual machine codes, prevents an attacker from analysing the virtual machine processing and contents.

8. SECURITY RULES

This section documents the security rules which are enforced by the Module, and under which the Module shall operate:

1. Within the constraints of FIPS 140-2 Level 1, the Module does not implement user authentication. It relies on the hosting operating system for user and operator authentication.
2. The Module is implemented entirely in software. The physical security is provided solely by the host platform, and as such the Module is not subject to the physical security requirements of the FIPS 140-2 standard.
3. The Module does not provide security mechanisms to mitigate any specific attacks outside the scope of FIPS 140-2 Level 1 requirements.
4. The Module automatically conducts a set of power-up self-tests. If any of the tests fails, the Module enters the Error State, and an error code will be returned indicating self-test failure. The Module does not provide any cryptographic service while in the Error State. If all of the tests pass, the Module enters the Normal User State. The Module provides cryptographic services only in the Normal User State.
5. At any time in the Normal User State, the operator of the Module (both the User and the Cryptographic Officer) is able to perform power-up self-tests.
6. This Module does not provide key generation.
7. The end user of the Module is responsible for zeroizing CSP and the microcode of the Module after uninstallation.

9. CRYPTOGRAPHIC KEY MANAGEMENT

All CSPs used by the Module are summarized in Table 5. The Module does not support key generation (section 9.2). All cryptographic keys are stored in a keystore (section 9.5), and keys can be zeroized (section 9.6)

CSP	Type	Key Sizes	Modes	Usage
AES Encryption and Decryption Key	Symmetric Secret Key	128, 192, 256 bits	ECB, CBC	Encryption, Decryption,
Triple-DES Encryption Key and Decryption Key	Symmetric Secret Key	192 bits	CBC	Encryption, Decryption,
RSA Decryption and Signature Generation Key	Private Key	2048, 3072 bits	N/A	RSA Decryption, Signature Generation, RSA-based Key Transport
RSA Encryption and Signature Verification Key	Public Key	2048, 3072 bits, 1024 bits for signature verification as legacy use	N/A	RSA Encryption, Signature Verification, RSA-based Key Transport
HMAC Key	Hashing Key	>= 112 bits	N/A	Message Authentication
KBKDF Key	Derivation Key	>= 112 bits	N/A	Key Derivation Using Pseudorandom Functions
Module Integrity Key	Hashing Key	128 bits	N/A	Embedded within the Module, used for Module Integrity Test using HMAC-SHA1
Storage Integrity Key	Hashing Key	128 bits	N/A	Embedded within the Module, used for Keystore Integrity Test using HMAC-SHA256
Key Encrypting Key	Symmetric Secret Key	128 bits	AES-CBC	Embedded within the Module, used to encrypt and decrypt keys in the keystore

Table 5 Critical Security Parameters (CSPs)

9.1. Random Number Generators

The Module implements the Random Number Generator (RNG) using AES with a 128-bit key as specified in ANSI X9.31 Appendix A.2.4. The data output from the RNG passes the continuous random number generator test (see section 10.2.4).

Note that RNGs specified in ANSI X9.31 Appendix A.2.4 are disallowed after 2015. Thus, the RNG provided by this Module is a non-approved cryptographic service (see Table 3). This Module does not provide an Approved RNG in a FIPS mode of operation.

128-bit seeds are retrieved from */dev/random* of the native device which collects entropy from different sources. Seeds are only retrieved for the initialization of the random number generator. In order to make sure that no seeds are the same, a delay of 1ns is added between two consecutive retrievals, and a test for inequality is performed. An error code is thrown in case the inequality test fails.

For day time information required by the RNG, a combination of the current time as number of seconds elapsed since 00:00 hours, Jan 1, 1970 UTC, and processor uptime in nanoseconds are used.

9.2. Key Generation

Key generation is not provided in either FIPS Approved or non-FIPS Approved mode of operation by this Module.

9.3. Key Establishment

The Module provides the following Approved key establishment methods as specified in Annex D to the FIPS 140-2 standard.

- Key derivation: The Module supports key derivation from an existing key as specified in SP 800-108 as well as password-based key derivation as specified in NIST SP 800-132. Note that password-based key derivation provided by this Module is non-approved in a FIPS mode of operation.

The Module provides the following Allowed key transport methods

- Key transport: The Module supports key transport using RSA-based key methodology (see Table 2).

It is the Crypto Officer's responsibility to choose sufficient key lengths for key encryption keys in order to protect the cryptographic keys to be transported. Keys may only be transported under equivalent or higher key strengths. The NIST SP800-57, Part 1 consists of Table 2, which compares security strengths for approved asymmetric and symmetric algorithms at different key lengths.

9.4. Key Entry and Output

Keys are entered into the Module in plaintext through Module's API, for instance, through API to perform encryption and decryption, or through API to enter a key into the keystore.

Note that output of the Random Number Generator as provided by this Module cannot be used as input to key entry API.

For key output, the Module supports key transport scheme using RSA OAEP (KTS-OAEP-Basic) as specified in SP800-56B section 9.2. The sender encrypts the keying material using the receiver's public key with the RSA-OAEP padding. Upon receiving the encrypted key, the receiver decrypts the ciphertext using its private key with RSA-OAEP (see key transport in section 9.3).

9.5. Key Storage

The Module stores all cryptographic keys in a keystore. The keys are imported into the Module using the API provided by the Module (e.g. VM_KEYSTORE_ADDKEY). Key binaries stored in the keystore are encrypted with an embedded key encrypting key using AES-CBC. Each key in the keystore is identified by a key alias, which is referenced by the caller application when invoking cryptographic operations.

In order to persistently save the keystore in the file system, the Module serializes the content of the keystore. To protect the integrity of the keystore file, the Module stores the HMAC-SHA256 value of the serialized keystore using an embedded storage integrity key.

Whenever the keystore file is loaded into runtime memory, the Module verifies the integrity of the file by using the storage integrity key. Only if the keystore has not been tampered with, the Module de-serializes the content of the keystore.

9.6. Key Zeroization

The Module stores keys in a *KeyStore* object while the keys are in use in runtime memory. The API VM_KEYSTORE_DELKEY is provided to delete a specific key from the keystore, and the Module zeroizes the memory used by this key (i.e., the affected memory will be overwritten with

zeros). VM_KEYSTORE_CLEAR is used to zeroize all keys in the keystore. Both APIs can only be called when the Module is in the Normal User State.

When performing encryption or decryption using any key stored in the keystore, the encrypted key in the keystore gets decrypted. The decrypted key is zeroized before the encryption or decryption function returns.

As described in section 9.5, the keystore in the file system is encrypted. It is the responsibility of the user to securely delete the keystore file from the file system.

10. SELF-TESTS

When initialized, the V-Key cryptographic module automatically performs the following self-tests. If any test fails, the Module will enter the Error State (see finite state model of the Module in Appendix B) which will prevent use of the Module. From the Error State, the Module automatically enters the Module Exit State. When entering the Error State or Exit State, error messages “Entering the Module Error State” and “Exiting the Module” are displayed.

Whenever a cryptographic service is called in the Module Error State, the error message “Module is in Error State!” is displayed and the error code VM_RESULT_ERROR (-990) is returned to the caller. Whenever a cryptographic service is called in the Module Exit State, the error message “Already exited V-OS, unable to run crypto function” is displayed, and the error code VM_EXIT(-994) is returned to the caller.

The Module provides an API, VM_SELF_TEST, which allows for executing the self-tests on demand for periodic testing of the module, and VM_GET_POWERUPTEST_STATUS which returns the outcome of self-tests.

10.1. Power-up Tests

When the microcode of the Module is loaded into the V-OS virtual machine, a suite of power-up self-tests is performed automatically (i.e. without any intervention from an operator) to ensure the integrity and correct operation of the provided cryptographic services. This section describes the power-up self-tests implemented by the Module.

10.1.1. Cryptographic Algorithm Test

The power-up self-tests for the following algorithms use a Known Answer Test (KAT), where a cryptographic value is calculated and compared with a stored, previously determined answer:

a) AES-ECB encrypt/decrypt (128-bit, 192-bit, and 256-bit keys)

b) AES-CBC encrypt/decrypt (128-bit, 192-bit, and 256-bit keys)

c) Triple DES encrypt/decrypt

d) SHA-1 / SHA-256

e) HMAC-SHA-1 / HMAC-SHA-256

f) Random number generators based on ANSI X9.31 AES

g) Key derivation using pseudorandom functions in counter mode

h) Password-based key derivation

i) Pairwise consistency test for RSA encrypt/decrypt (2048-bit modulus *n*)

j) Pairwise consistency test for RSA-SHA1 and RSA-SHA256 signature generation and verification (2048-bit modulus *n*)

10.1.2. Software Integrity Test

The HMAC-SHA1 value of the Module is pre-computed by the vendor. The header part of the Module's microcode has a reserved area of 20 bytes to store this value. The key for HMAC is embedded in the Module. During the software integrity self-test, the Module reads the microcode (in binary representation) from the memory, and applies HMAC-SHA1 to the binary excluding the HMAC-SHA1 value in the header section. The computed value is compared with the value stored in the header for verification.

10.2. Conditional Tests

The Module performs several conditional tests automatically as specified in FIPS 140-2. These tests cannot be disabled by users.

10.2.1. Pair-wise Consistency Test

Pairwise consistency test is performed for RSA encryption and decryption as well as signature generation and verification as part of the power-up tests.

For RSA-based encryption and decryption, the Module encrypts a message with the public key, and verifies that the ciphertext differs from the plaintext. If the two values are equal, the test fails. If the two values differ, the Module decrypts the resulting ciphertext with the private key, and verifies that the decrypted value equals the original message. If the two values are not equal, the test fails.

For RSA-based signature generation and verification, the Module encrypts a message with the public key, and verifies that the ciphertext differs from the plaintext. If the two values are equal, the test fails. If the two values differ, the Module decrypts the resulting ciphertext with the private key, and verifies that the decrypted value equals the original message. If the two values are not equal, the test fails.

10.2.2. Software/Firmware Load Test

Not applicable, as the Module does not load external software or firmware.

10.2.3. Manual Key Entry Test

Not applicable, as the Module does not allow keys to be manually entered.

10.2.4. Continuous Random Number Generator Test

The Module implements the Random Number Generators based on AES as specified in ANSI X9.31 Appendix A.2.4. Note that RNGs specified in ANSI X9.31 Appendix A.2.4 are disallowed after 2015. Each time the RNG is called, the continuous RNG test as specified in FIPS 140-2 section 4.9.2 is performed to test for failure to a constant value.

The RNG based on AES generates blocks of 128 bits. The first *n* bits (*n* = 128) generated after initialization are not used. They are stored for comparison against the next *n* bits. The test compares each subsequently generated *n* bits against the previously generated *n* bits. The test fails if any two compared *n*-bit sequences are equal.

10.2.5. Bypass Test

Not applicable, as the Module does not implement a bypass capability.

10.3. Critical Function Tests

The Module does not implement any critical function tests.

11. DESIGN ASSURANCE

V-Key uses best practices during the design, deployment, and operation of the V-Key cryptographic module to provide assurance that the functional requirements and specifications are realized in the implementation.

11.1. Configuration Management

The configuration management system includes following configuration items

- The source code of the V-Key Cryptographic Module and reduced OS. Both are within the logical cryptographic boundary.
- The source code of CyaSSL
- The source code of the Wrapper
- The source code of V-OS Virtual machine
- Security policy
- User guide

Both the source code and the associated documentation are managed using a V-Key version control system running on private GitHub source code repository with access control implemented. All changes are tracked and versioned. Whenever a new version of a file is stored in the configuration management system, it is labelled with a unique version number. These changes are also linked to the respective work items (stories and defects) in a JIRA issue management system for traceability. Builds are automated through Jenkins to achieve continuous integration. A set of automated tests are also run as part of each build, with test results reflected in Jenkins. Released product versions are stored in Subversion.

11.2. Delivery and Operation

The companion document, *V-Key User Guide*, documents the procedures for secure installation, initialization, and start-up of the V-Key cryptographic module.

11.3. Development

The companion document, *V-Key User Guide*, provides the functional and design specifications. It describes the cryptographic module, the external ports and interfaces of the module, and the purpose of the interfaces. The source code is annotated with comments to depict the

correspondence of the components to the design of the module. Enhancements and defects are defined as work items in the JIRA issue management system, which is linked to a Confluence team collaboration system where requirements can be discussed with stakeholders. Architectures and designs are discussed in Confluence, with finalized ones exported and stored in Subversion. Test cases and automation code are also stored in Subversion. These artifacts are all linked to the respective work items in JIRA.

11.4. Guidance Documents

The companion document, *V-Key User Guide* contains the *API Reference*, the *Crypto Officer Guidance* and the *User Guidance* documentation. The API Reference describes the purpose, input and output parameters of all Approved cryptographic services and functions provided by the Module. The Crypto Officer Guidance is a reference for Crypto Officers provisioning and administrating the V-Key cryptographic module for use by end-users. Together with the API Reference, the User Guidance explains the proper and complete use of the Module.

12. MITIGATION OF OTHER ATTACKS

The Module does not provide security mechanisms to mitigate any specific attacks outside the scope of FIPS 140-2 Level 1 requirements.

13. References

[1] CyaSSL Embedded SSL Library (wolfSSL), http://www.yassl.com/yaSSL/Products-cyassl.html.

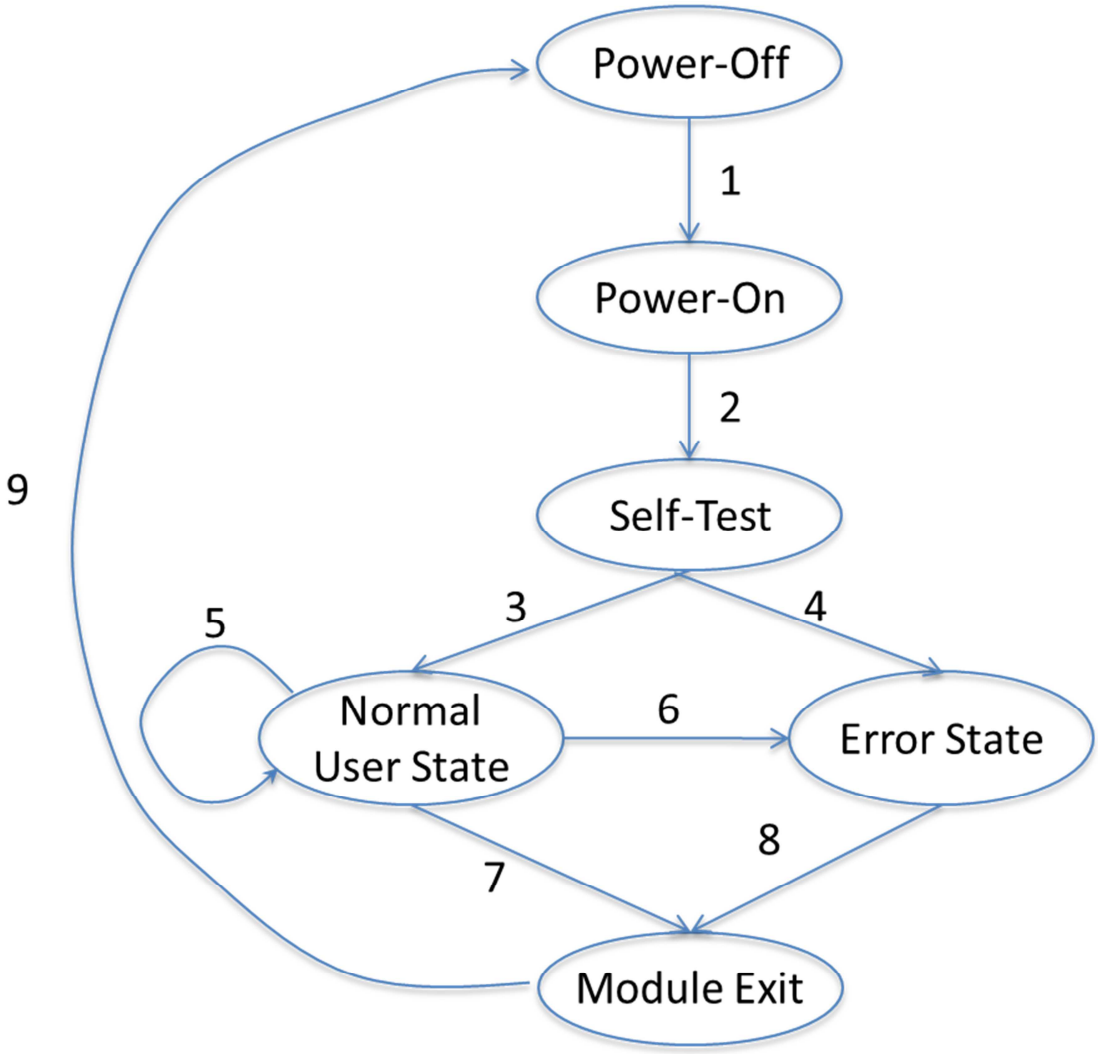
APPENDIX A. ALGORITHM CERTIFICATES

Algorithm	Cert#	Description
AES	3679	AES encryption and decryption in ECB and CBC mode, with key size 128, 192 and 256 bits.
Triple DES	2057	3-Key Triple DES encryption and decryption in CBC mode.
SHS	3093	SHA-1 (BYTE-only) SHA-256 (BYTE-only)
HMAC	2425	HMAC-SHA-1 (Key Sizes Ranges Tested: K<B, K=B, K>B) HMAC-SHA-256 (Key Sizes Ranges Tested: K<B, K=B, K>B)
RSA	1900	ALG[RSASSA-PKCS1_V1_5]; SIG(gen); SIG(ver); MOD([1024], [2048], [3072]); SHA([SHA-1], [SHA-256])
KBKDF	74	Based on HMAC-SHA1 and HMAC-SHA256 in Counter Mode (with location of counter before fixed input data)

APPENDIX B. FINITE STATE MACHINE MODEL

This Appendix describes the Finite State Model (FSM) for an application using the V-Key cryptographic module. The Module can be in only one state at a time.

B.1. State Diagram



B.2. State Description

B.2.1. Power-off

The Module is in the *Power-Off* state, when it has not been loaded into memory.

B.2.2. Power-On

During loading and starting the Module, if any error occurs (for instance, the firmware file cannot be found or the Module cannot be started within the virtual machine) an error code will be returned.

Only when the firmware has been loaded into memory and the Module has been successfully started within the virtual machine, the Finite State Machine is instantiated, and the Module enters into the *Power-On* state.

After entering the *Power-On* state, the Module immediately transitions to the *Self-Test* state.

B.2.3. Self-Test

The Module conducts the power-up self-tests as described in Section 10.1 during this state. If all of the tests pass, the Module transitions to the *Normal User State*. If any of the tests fails, the Module transitions to the *Error State*.

B.2.4. Normal User State

The Module enters the *Normal User State*, if all power-up self-tests have executed successfully. In this state, the Module is idle, waiting for an operation call through the defined API. The Module remains in this state if the operation returns successfully. If the operation fails, the Module either remains in this state or transitions to the *Error State* depending on the specific error.

In this state, the Data Input Interface provides the data input for the corresponding control input. The Control Input Interface implicitly defines the role (Crypto Officer or User) being used. The Data Output Interface provides data output returned to the caller for the corresponding control input. The Status Output Interface provides return codes returned to the called.

B.2.5. Error State

Initially, the Module enters the *Error State* through failure in one or more power-up self-tests. The Module can also enter the *Error State* through failure in loading a serialized keystore (see Section 9.5) from the file system in the *Normal User State.* Loading of keystore fails, when the Module detects a tamper of the keystore integrity.

Once the Module is in the *Error State*, no cryptographic service can be invoked. The Module returns the error code VM_RESULT_ERROR (-990), whenever it receives a service call in the Error State.

From the *Error State*, the Module subsequently enters the *Module Exit* state. The Module needs to be restarted for further processing.

B.2.6. Module Exit

The Module enters the *Module Exit* state, when the user exits the application from either the *Normal User State* or the *Error State*.

Once the Module is in the *Module Exit* state, no cryptographic service can be invoked. The Module returns the error code VM_EXIT (-994), whenever it receives a service call in the *Module Exit* state.

B.3. State Transition Conditions and Events

	Current State	**Input Events**	**Output Events**	**Next State**
1	Power-Off	Initiate Module load and start	Load Success code	Power-On
2	Power-On	Automatic transition	No output	Self-Test
3	Self-Test	All power-up self-tests run successfully	Self-test success. A non-negative return code indicating that all power-up self-	Normal User State

			tests are successful[3] .	
4	Self-Test	Any power-up self-tests fails	Self-test failure. A negative error code indicating the failure.	Error State
5	Normal User State	Run cryptographic operation either successfully or failed	Operation success or operation failure	Normal User State
6	Normal User State	Failure during execution of cryptographic operation	Return error code to the calling function. Display error message “Entering the Module Error State”.	Error State
7	Normal User State	User exits the application	Operation success.	Module Exit
8	Error State	Error code returned from cryptographic operations	Display error message “Exiting the Module”	Module Exit
9	Module Exit	Function to unload the Module invoked	Module unloaded from the memory	Power-Off

[3] Power-on self-tests are triggered by the function VM_SYS_INIT which starts the Module within the V-OS virtual machine. The function returns either an error code or a non-negative return code indicating that self-tests are successful.