ASIGRA Inc.

AsigraEncModule Encryption Library

(Software Version 1.0)

FIPS 140-2 Non-Proprietary

Security Policy

Table of Contents

1. Introduction

1.1 Overview

This is a non-proprietary Federal Information Processing Standard (FIPS) 140-2 Security Policy for ASIGRA Inc.'s AsigraEncModule. The AsigraEncModule ("Cryptographic Module" or "Module") is a cryptographic library for C++ language users providing various hash algorithms, encryption algorithms and random number generation. This Security Policy specifies the rules under the AsigraEncModule must operate. These security rules are derived from the requirements of FIPS 140-2 and related documents

1.2 Purpose

This Security Policy is a required document as part of the FIPS 140-2 submission. The FIPS 140-2 validation submission is composed of the following parts:

- Security Policy
- Crypto Officer and User Guide
- Design Assurance Document
- API Reference
- Source Code

1.3 Security level

The Cryptographic Module meets the following security level (according to the FIPS 140-2 security requirements sections):

Section No.	*Area Title*	*Level*
1	Cryptographic Module Specification	1
2	Cryptographic Module Ports and Interfaces	1
3	Roles, Services, and Authentication	1
4	Finite State Model	1
5	Physical Security	N/A
6	Operational Environment	1
7	Cryptographic Key Management	1
8	Electromagnetic Interference/Electromagnetic Compatibility	1
9	Self-Tests	1
10	Design Assurance	1
11	Mitigation of Other Attacks	N/A

2. Cryptographic Module Specification

The AsigraEncModule is a multi-chip standalone software cryptographic module that operates with the following components:

- a commercially available general-purpose computer hardware
- a commercially available Operating System (OS) that runs on the computer hardware

As a software cryptographic module, the AsigraEncModule provides access to cryptographic services by the usage of "functions" that are invoked by module users. Invoking the functions of the module represents the mechanism through which users gain access to the cryptographic services offered by the module. As such, the term "function" and "service" are used interchangeably throughout this document.

2.1 Software Environment

The module is offered as a dynamic library for the following Operating Systems:

- Linux, running on a x86-compatible CPU
- Linux, running on a x64-compatible CPU
- Microsoft Windows, running on a x86-compatible CPU
- Microsoft Windows, running on a x64-compatible CPU
- Macintosh OS X, running on a x86-compatible CPU

2.3 Master component list

The AsigraEncModule encryption library is a software-only module. The software part is composed of one item: the dynamic library containing the module. Depending on the operating system, this is named:

- ***asigraencmodule.dll*** on Windows (32 or 64 bit versions)
- ***libasigraencmodule.so*** on Linux (32 or 64 bit versions)
- ***libasigraencmodule.dylib*** on Mac OS X

The AsigraEncModule encryption library is used on a general purpose computer. The actual hardware and O/S builds for these general purpose computers vary. The library was specifically tested during the FIPS 140-2 validation on the following hardware/software environments:

O/S Type	***Environment***
Windws 32-bit	Microsoft Windows Server 2003, Enterprise Edition, 5.2.3790 Service Pack 2, Build 3790 2GHz Intel CPU, 1GB RAM
Windows 64-bit	Microsoft Windows Server 2003 x64, Standard Edition, 5.2.3790, Service Pack 1, Build 3790 Dual-Core Intel Xeon at 2.4 GHz, 3 GB RAM
Linux 32-bit	RedHat Enterprise Linux 5, Update 6 3GHz Intel P4 CPU, 2GB RAM

O/S Type	*Environment*
Linux 64-bit	RedHat Enterprise Linux 5 x64, Update 6, 2 x Quad Core Intel Xeon at 2.5 GHz, 8GB RAM
Mac OS X	Mac OS X, 10.5 2 x Dual-Core Intel Xeon at 2GHz, 2 GB RAM

2.4 Module Interfaces

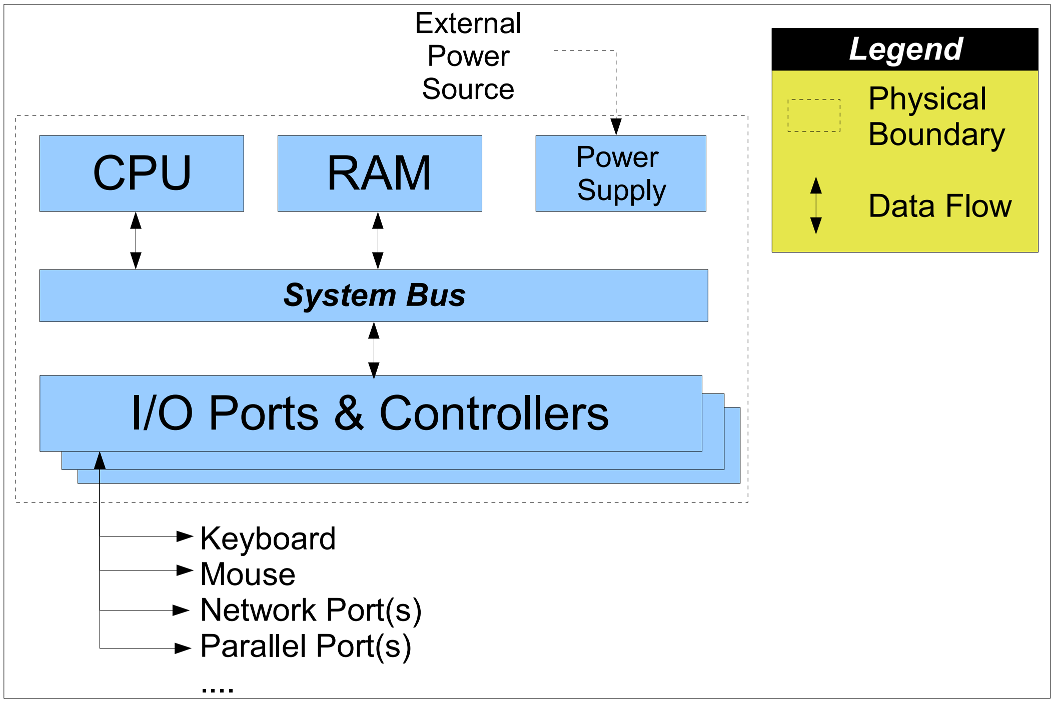
The Cryptographic Module's physical interfaces consist of the keyboard, mouse, monitor, CD-ROM drive, floppy drive, serial ports, COM ports, USB ports, network adapters and any other I/O hardware components part for the general-purpose computer hardware on which the module operates. However, the module sends/receives data entirely through the underlying logical interface, a C++ API documented in the Cryptographic Module API Reference.

The following table represents the a mapping between the logical and the physical interfaces of the cryptographic module:

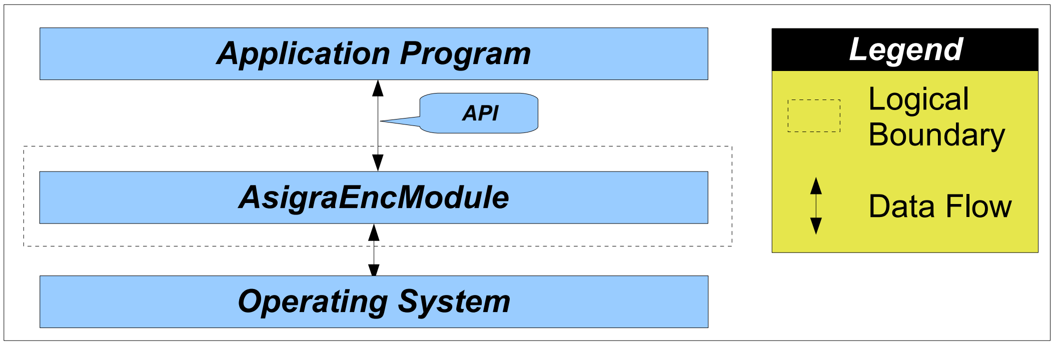
Logical Interface	*Physical Interface Mapping (Standard PC)*	*Cryptographic Module Mapping (API)*
Data Input Interface	Keyboard/mouse/CD-ROM/DVD-ROM, floppy drive, serial/parallel/USB/network ports	Arguments for a function that specify module data input
Data Output Interface	Floppy drive, monitor, serial / parallel / USB / network ports	Arguments for a function that specify where the function results are stored
Control Input Interface	Keyboard/mouse/CD-ROM/DVD-ROM, floppy drive, serial/parallel/USB/network ports	API functions are used to initialize the module and control the operation of the module
Status Output Interface	Floppy drive, monitor, serial / parallel / USB / network ports	Return values of the function calls and arguments for that specify where to store error information.
Power Interface	Power switch	Not Applicable

2.5 Module Block Diagram

The following block diagram show the physical hardware ports for the data input, control input, data output and status output.



The following block diagram displays the logical interface to the Cryptographic Module, illustrating the use of the API for data input/output and control input/status output.



As a software cryptographic module, the *cryptographic boundary* for the module on a general purpose computer is the computer's case, which physically encloses the complete set of hardware and software.

2.6 Approved mode of operation

The module implements and provides access only to FIPS-approved cryptographic functions. As a result, the module always operates in an approved mode of operation. This "FIPS-mode" is the only mode of operation for the module.

3. Roles, Services and Authentication

3.1 Roles and Services

The User and Crypto Officer roles are implicitly assumed by the entity that can access the services implemented by the Cryptographic Module. The following table defines authorized services that are available to each user role:

Role	*Authorized Services*
Crypto User	• Library services as defined below
Crypto Officer	• Library services as defined below • Library installation

The following is a list of services that are provided to the above roles (named "Library services" in the table above):

- Library initialization by initializing a consistency check
- Status checking (library version retrieval, checking the success/failure of the last consistency check)
- Cryptographic services offered by the library: encryption, message digest, random number generation, zeroize and self-test

As a library and as allowed by FIPS 140-2 Level 1 requirements, the module does not support user identification or authentication for these roles.

3.2 Authentication

The Cryptographic Module does not provide identification or authentication mechanisms that would distinguish between Crypto user and Crypto Officer roles. The roles are implicitly assumed by the services that are accessed and can be differentiated by assigning installation and configuration tasks to the Crypto Officer.

3.3 Services

The Cryptograhic Module offers only *approved cryptographic algorithms*. The following is a list of cryptographic algorithms provided:

Cryptographic Service	*Algorithm*	*Standard*	*NIST Certificate*	*Implementation Function & Returned Interface*
Symmetric Cipher	AES – CBC and ECB – with 128, 192 or 256 bit keys	FIPS 197	#968	Encryptor_AES128 -> IEncDataEncryptor Decryptor_AES128 -> IEncDataDecryptor Encryptor_AES192 -> IEncDataEncryptor Decryptor_AES192 -> IEncDataDecryptor Encryptor_AES256 -> IEncDataEncryptor Decryptor_AES256 -> IEncDataDecryptor
Message Digest	SHA-1 SHA-224 SHA-256 SHA-384 SHA-512	FIPS 180-2, including Change Notice 1	#938	Digest_SHA1 -> IDataDigester Digest_SHA224 -> IDataDigester Digest_SHA256 -> IDataDigester Digest_SHA384 -> IDataDigester Digest_SHA512 -> IDataDigester
	HMAC-SHA-256	FIPS 198	#541	Digest_SHA_HMAC -> IDataDigester
Random Number Generation	ANSI X9.31 Appendix A, 2.4 Using AES	ANSI X9.31	#546	RNG_AnsiX931_AES -> IencRandomDataGenerator

In addition to the cryptographic services, the module provides the following additional control & status checking functionality (functions directly callable by an operator):

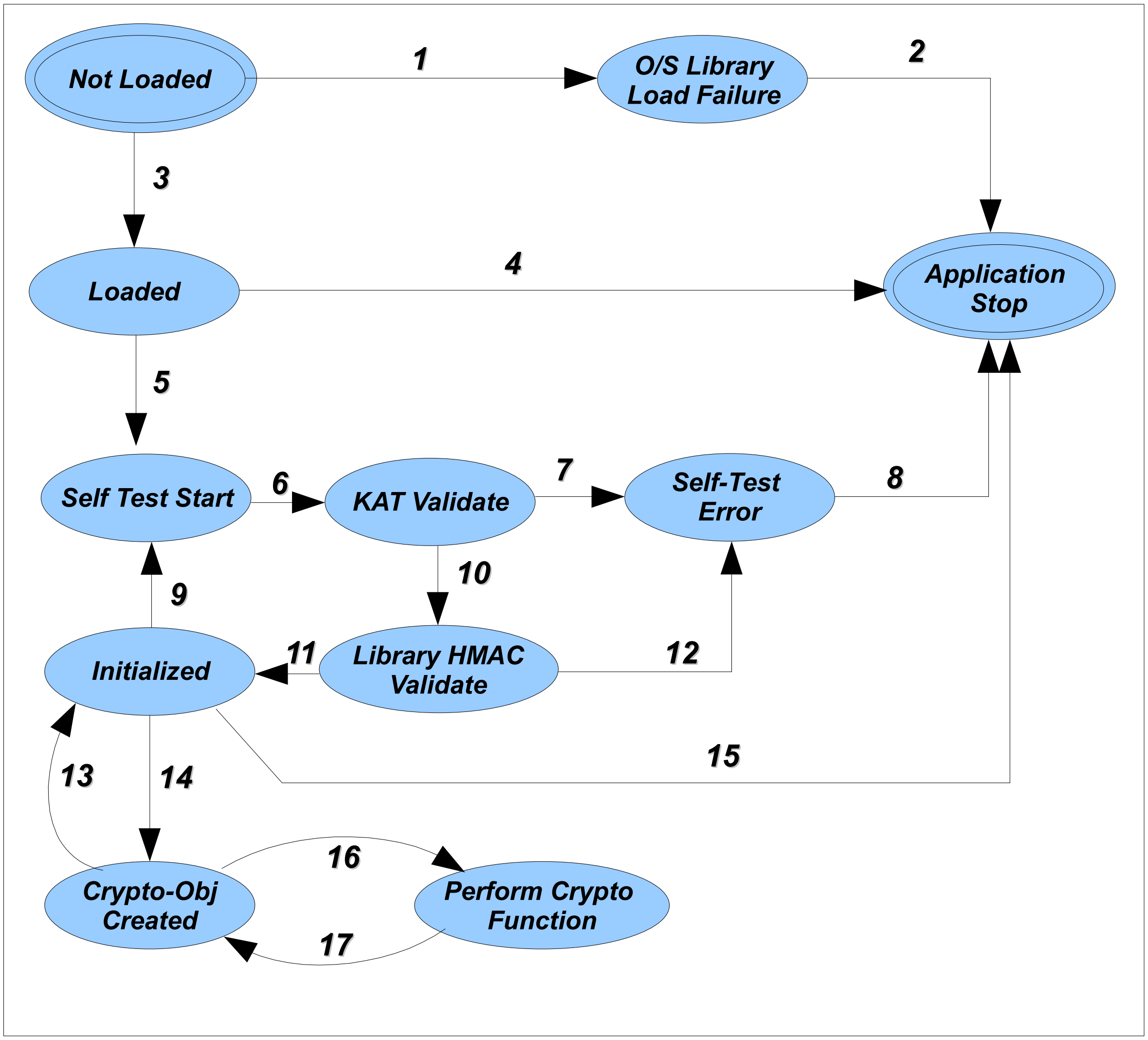
Provided service	*Implementation Function*		*Keys & CSP*	*Access Type*	*Role(s)*
Library status checking ("Show status" functionality of the module)	GetVersion SelfTestStatusOK	-> check library version -> check for self-test errors	No CSP	Not Applicable	Crypto Officer Crypto User
Self-test initiation (for initial and on-demand self-test)	PerformSelfTest	-> initiate a self-test of the library	HMAC Key	R/E	Crypto Officer Crypto User
Zeroize	Release	-> clears the CSP and releases allocated memory	All keys except HMAC	W	Crypto Officer Crypto User

The following table identifies CSPs and type of available access for the supported services. As per section 3.1, the cryptographic module does not support user identification or authorization for user roles. As such, the same level of access to cryptographic services is granted to all user roles:

Service	*Cryptographic Keys and CSPs*	*Types of Access (e.g. RWE)*	*Role(s)*
AES Encryption and Decryption	Secret Key	RW	Crypto Officer Crypto User
SHA-1, SHA-224, SHA-256, SHA-384, SHA-512 Hashing	Checksum	RW	Crypto Officer Crypto User
HMAC-SHA-256	Secret key and Checksum	RW	Crypto Officer Crypto User
ANSI X9.31 RNG	Seed Value and Seed Key	No operator access to the seed key or value	Crypto Officer Crypto User
Library Installation	HMAC Key	W	Crypto Officer

4. Finite State Model

The diagram below represents the finite state model of the Cryptographic Module. The Module can be only in one state at a time. Transitions between states are a result of various software events or actions. Operations specific to the Crypto Officer are outside the scope of this state model.



4.1 State Description

State	*Description*
Not Loaded	The module is in this state when it has not been loaded into memory. This state corresponds to the "*Power Off*" state defined in FIPS 140-2
Loaded	In this state, the module has been loaded into memory however the power-on self-test has not been performed. The module services are not available at this point until the module self-test is executed.

State	*Description*
O/S Library Load Failure	The state represents a failure to load the module in memory, such as "file not found" or "incompatible library type".
Self Test Start	The Module Entered a self-testing mode. This mode is entered either after the library loading (mandatory initial self-test) or when the user performs an "on-demand self test".
KAT Validate	The Module is performing Known Answer Tests for the provided cryptographic functions. A known answer test will be performed for every cryptographic algorithm implemented by the module (AES – all key lengths, SHA – all sizes, SHA-1-HMAC and RNG)
Library HMAC Validate	The Module will compute library-module HMAC-checksum for the entire dynamic library code (excluding the compiled-in HMAC value and compile-specific timestamps and checksums). This HMAC covers the data and executable code for all the module components.
Initialized	This state corresponds to the "*Power On*" state defined in FIPS 140-2. This state is entered to after the power-on self-test has been completed successfully. The module cryptographic services can be used while the module is in this state.
Self-Test Error	The Module enters this state if at one of the self-test checks has failed. Once the module entered this state, the cryptographic services provided by the module are not available anymore and any attempts to use a cryptographic services offered by the module will result in an error.
Application Stop	The module enters the Application Stop state when the application program using the module terminates either normally or abnormally.
Crypto-Obj Created	A cryptographic object of the library is created and the corresponding CSP has been initialized and resides in the Cryptographic Module memory.
Perform Crypto Function	A cryptographic function is applied using one of the the cryptographic objects created using the library API.

4.2 State Transition Description

Transition #	*Description*
1	Transition from the "Not Loaded" state into the "Load Error" state as a result of a O/S Specific library load failure. O/S provides failure code and message.
2	Application Termination as a result of a Cryptographic Library Load Failure. The O/S will automatically terminate an application linked with the cryptographic library in case the cryptographic library fails to load.

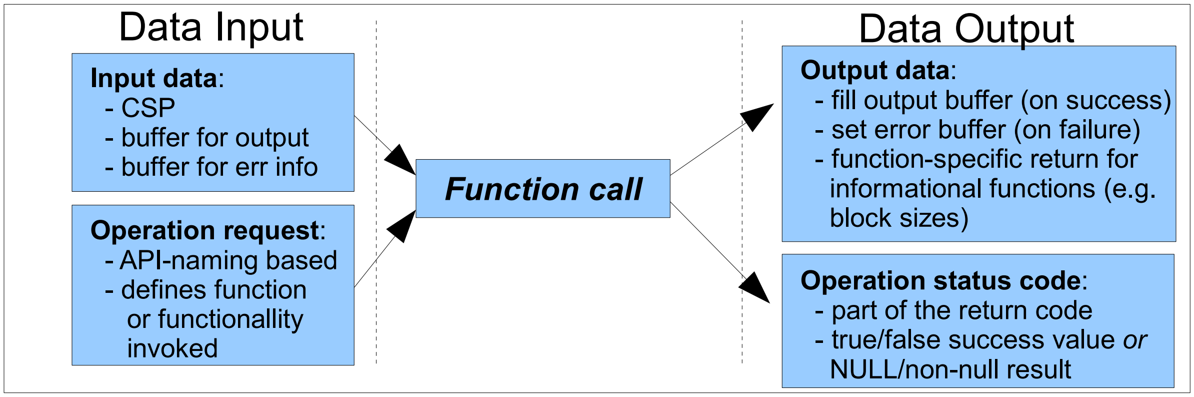
Transition #	***Description***
3	Transition into the "Loaded" State as a result of the O/S successfully loading the cryptographic module in memory.
4	Application termination after the Cryptographic Library has been loaded,without the application using any cryptographic functions offered by the module. This may occur if the O/S starts the application (module is successfully loaded) however the application needs to terminate before needing to perform cryptographic functions (e.g. Detecting application-specific configuration errors)
5	Initiation of a self-test of the library after the cryptographic library has been loaded into memory. If an application needs to use the cryptographic functionality offered by the library, this step has to be performed. Attempting to utilize cryptographic functionality offered by the library before initiating a self-test will result in an error being returned to the application.
6	Transition to the Known-Answer-Test phase. The KAT is the first phase of the application self-test functionality
7	Transition into the Self-Test error state as a result of any one of the Known Answer Tests failing.
8	Transition to the Application Stop phase after the module entered a self-test error state. This is the only possible transition once the module entered a self-test error phase. Any attempts to use any cryptographic functions in the module while the module is in a "self-test error" state will result in an error being returned to the application
9	Transition to the "self-test start" state as a result of the user application requesting an "*on-demand self-test*" of the cryptographic library while the library is in an initialized state.
10	Transition to the "Library HMAC Validation" as a result of the successful validation of the Known answer tests in the library.
11	Transition to the "Initialized" state as a result of the successful validation of the library data HMAC.
12	Transition to the "self-test error" state as a result of a failure to match the library data HMAC of the application code and data with the expected compiled-in value.
13	Transition to the "Initialized" state by the destruction of a Crypto object. The Destruction of a crypto object will perform a in-memory overwrite of the CSP with 0.
14	Creation of a cryptographic object by providing CSP data to the cryptographic module.
15	Stopping the application when the module is in an "Initialized" State.
16	Requesting the cryptographic module to perform a cryptographic function using a specified cryptographic object

Transition #	*Description*
17	The application completing the cryptographic request and returning the result (either an error-code in case of invalid parameters or state or the result of the cryptographic function).

4.3 Cryptographic function input, output and control

The library provides access to cryptographic functionality by using a C++ object-based interface. Each function exported by the library takes as parameters input data (e.g. CSP) and destination buffers for output results like encryption/hash output buffers and error texts. Functions returning object pointers will return NULL on error or a valid object pointer on success. Functions returning boolean values return true on success and false on failure.

In case a self-test operation fails, any subsequent function calls will return a “failure” result code (i.e. “false”) and the error text will be the same as the error text returned by the self-test function.


Data Input
Input data:
- CSP
- buffer for output
- buffer for err info
Operation request:
- API-naming based
- defines function or functionallity invoked
Function call
Data Output
Output data:
- fill output buffer (on success)
- set error buffer (on failure)
- function-specific return for informational functions (e.g. block sizes)
Operation status code:
- part of the return code
- true/false success value *or* NULL/non-null result

Below is a table describing data inputs & outputs for the cryptographic functionality offered by the library:

Type of function	*Example Functions*	*Input*	*Output*	*Status information*
Crypto-object creation	Encryptor_AES128 Decryptor_AES192 Digest_SHA1 ...	CSP (e.g. Key, IV) buffer for error text pointer	Pointer to crypto-object pointer to error text	Return code: NULL on failure valid object pointer on success
Crypt-function call	EncryptECB DecryptCBC AppendDigestData GetDigest ...	Input data buffer pointer to output buffer buffer for error text pointer	Data in output buffer pointer to error text	Return code: true on success false on failure
Self-Test	PerformSelfTest	buffer for error text pointer	Pointer to error text	Return code: true on self-test ok false on failure
Status information	CheckVersion SelfTestStatusOK	Described in the Api Reference	Described in the Api Reference	Described in the Api Reference

	Release GetBlockSize			

Note: “status information” functions refers to functions which do not perform FIPS-approved cryptographic functions and are used for library status information or memory management. The following list describes the purpose of each of these functions:

- ***CheckVersion*** returns the current version of the library as a string (no input required)
- ***SelfTestStatusOK*** returns the current library self-test status and in case of failure, the associated failure description (as a pointer to an error string).
- ***Release*** will release the memory allocated by the library for the storage of some cryptographic object
- ***GetBlockSize*** is an informational function about the block size of encryption objects created by the library. This function cannot fail as the block size is fixed for any algorithm and simply returns this fixed size.

5. Physical Security

The AsigraEncModule is a purely software module and thus physical security requirements do not apply.

6. Cryptographic Key Management

6.1 Key Generation

The Cryptographic Module offers a FIPS-approved RNG for the purpose of generating symmetric keys and random initialization vectors for encryption algorithms such as AES.

6.2 Key Entry and Output

The Cryptographic Module does not import or export keys. It is the responsibility of the application to import/export keys from the physical cryptographic boundary.

6.3 Key Storage

As a cryptographic library, the Cryptographic Module does not provide long-term cryptographic key storage.

The module will perform short-term volatile (in memory) storage of the plain-text version of the cryptographic keys for the duration of the cryptographic operation (encryption / decryption / RNG / HMAC). The encryption keys are automatically zeroized when the “Release” function is called on the cryptographic API objects.

7. Electromagnetic interference / electromagnetic compatibility

The AsigraEncModule is a purely software module and thus electromagnetic interference / electromagnetic compatibility do not apply.

The platforms that host the module are expected to meet the EMI/EMC for FIPS 140-2 security levels 1 and 2.

8. Self-Tests

The Cryptographic Module performs a number of power-up self-test to ensure proper operation of the Module. The tests include in-memory integrity tests, known answer tests and conditional tests. No cryptographic service is available until the power-up self-test has been completed successfully.

On-Demand self-tests can be invoked by the Crypto Officer or user by invoking the “PerformSelfTest” function. The function is described in the *Api Reference*.

In case the integrity test fails or one of the known answer tests fails, the Cryptographic Module enters in the “Self-Test Error” state and no cryptographic functionality is made available.

8.1 Known answer tests

The Cryptographic Module performs the following known answer tests:

Algorithm	***Known Answer Test***
AES	Encryption and Decryption using 128, 192 and 256 keys
SHA	SHA-1, SHA-224, SHA-256, SHA-384, SHA-256 checksum calculation
HMAC	HMAC-SHA-256 checksum calculation
RNG	Random number generation from known IV and date/time stamps

8.2 Conditional tests

The Cryptographic module will perform a continuous RNG test for all data generated by the “ANSI X9.31” RNG. Each newly generated 16 random bytes is checked against the 16 bytes generated by a previous RNG iteration to ensure that the RNG is not stalled (as per FIPS 140-2 section 4.9.2).

8.3 Integrity test

The software will perform a library integrity test using a HMAC-SHA-256 test of the entire library module as it is stored on disk. This computed HMAC value is compared with a compiled-in HMAC in order to verify the module integrity.

The HMAC is computed over the entire library as it is stored on disk, excluding compile-specific parts and the compiled-in HMAC value itself. The HMAC computation is done over the entire library (i.e. “asigraencmodule.so”, “asigraencmodule.dylib” or “asigraencmodule.dll”) and will cover the following items:

- All executable code implemented by the module (covering all cryptographic functions and self-test functionality of the module, including the HMAC computation itself)
- All static data of the module, including initialization vectors and cryptographic functions initialization data (covering all cryptographic functions and self-test functionality of the module, with the only item excluded being the 256-bit HMAC value itself)
- Other operating-specific data like dynamic library headers and linking information (with the exception of Win32 PE header timestamps & checksums and Mach-O timestamps & checksums, which are compile-time specific)

Note that the HMAC key used for the integrity self test is never zeroized. The rationale for not zerozing the HMAC key is that this would cause the module to not pass the self test.

9. Mitigation of other attacks

The Cryptographic Module does not provide security mechanisms to defend against attacks beyond those required by FIPS 140-2 Level 1 for monitoring the integrity of the Module

10. References

The following documents were used to support the validation of the AsigraEncModule library:

[1] FIPS PUB 140-2, *Security Requirements for Cryptographic Modules,* 2001 May 25

[2] National Institute of Standards and Technology, *NIST-Recommended Random Number Generator Based on ANSI X9.31 Appendix A.2.4 Using the 3-Key Triple DES and AES Algorithms*, 2005 January 31

[3] FIPS PUB 197, *Advanced Encryption Standard (AES)*, 2001 November 26

[4] FIPS PUB 180-2 (+ Change Notice to include SHA-224), *Secure Hash Standard (SHS)* , 2002 August 1

[5] FIPS PUB 198, *The Keyed-Hash Message Authentication Code (HMAC)*,2002 March 6

[7] NIST Special Publication 800-38A, *Recommendation for Block Cipher Modes of Operation Methods and Techniques*, 2001 December