FIPS 140-2 Level 1 Security Policy for

ZixCorp Crypto Module

version 1.0

Version 1.6
May 26, 2009

Table of Contents

Table of Figures

Table of Tables

Overview

The ZixCorp Crypto Module version 1.0 is a software cryptographic library that provides cryptographic services to the overall ZixCorp Email Encryption Service. The software token contains implementations of the following Approved cryptographic algorithms:

- AES
- Triple-DES
- RSA
- SHA-1
- HMAC SHA-1
- FIPS 186-2 Appendix 3.1 RNG

The cryptographic module consists of a software token implemented as a single shared object library with the filename “libsoftokn3.so”. It runs in the operational environment of a standard Intel-based computer running the Linux operating system. The cryptographic module boundary is the case of the computer, containing the integrated circuits of the motherboard, the CPU, random access memory, keyboard, mouse, video interfaces, hard drive, and other hardware components. The term “module” is used to refer to the software token working in the operational environment described above.

.

Security Requirements

Cryptographic Module Specification

The cryptographic module is the “libsoftokn3.so” shared object library, version 1.0, running in the operational environment of a standard Intel-based computer running the Linux operating system. Per section 4.5 of FIPS PUB 140-2, the module is a multiple-chip standalone module. No custom integrated circuits are used by the module.

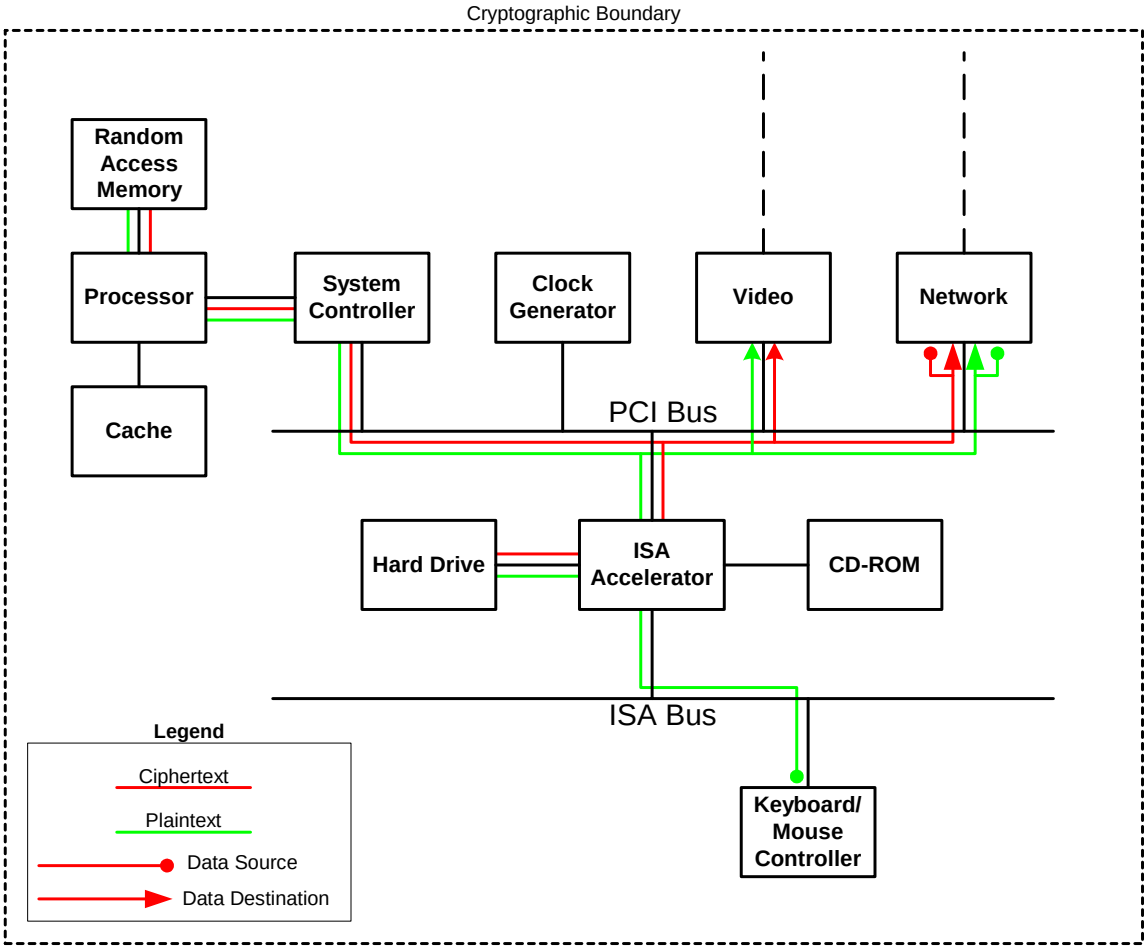

Figure 1 – Generic Computer Hardware Functional Block Diagram

Figure 1 shows the functional block diagram for the computer on which the software token runs. All components shown in the diagram are within the physical cryptographic boundary of the module, and the diagram shows interconnections among the major components of the module. Dashed lines represent connections to equipment or components outside the cryptographic boundary.

Software is stored on the hard drive of the system, and loaded into random access memory for execution. The processor component shown in Figure 1 executes all software.

Figure 2 – Software Architecture Functional Block Diagram

Figure 2 is a functional block diagram that demonstrates how the software token fits into the overall operational environment. The software token is the shaded gray box which represents the libsoftokn3.so token. The functionality of the token is exercised by the application, The libsofttokn3.so file contains all of the module's cryptographic functions.

The logical boundary between the FIPS token and the application is the PKCS#11 API that exposed by the library.

Security Functions

Table 1 lists the cryptographic algorithms implemented by the software token.

Table 1 – Security Functions

Algorithm	Approved	Algorithm Certificate
AES	Y	#321
Triple-DES	Y	#385
RSA	Y[1]	#108

[1] RSA is used for key wrapping. 1024-bit modulus keys provide an effective symmetric key strength of 80 bits, and 2048-bit keys provide an effective symmetric key strength of 112 bits.

SHA-1	Y	#394
HMAC SHA-1	Y	#127
FIPS 186-2 RNG	Y	#145
Diffie Hellman	Y[2]	
DSA	N[3]	
Elliptic Curve	N[4]	
MD2	N	
MD5	N	
HMAC MD5	N	

The interface to the software token is a PKCS#11 based API.

The module supports AES, Triple-DES, and RSA cryptographic keys, X.509 certificates, and HMAC for message authentication and module integrity.

Diffie Hellman, DSA, Elliptic Curve, MD2, MD5, and HMAC MD5 algorithms shall not be used in FIPS mode of operations.

Cryptographic Module Ports and Interfaces

This section will first detail the physical interfaces to the module, and then the logical interfaces.

Physical Interfaces

The physical interfaces are those of a standard Intel-based computer system, including the computer keyboard and mouse, network ports, CD-ROM drive, video monitor port, and power plug. All port connectors used in the module are standard. The system has a serial port which is not used.

Logical Interfaces

The logical interface to the module is the Application Programming Interface (API) of the software token. The overall NSS library treats the software token as a PKCS#11 token implementation, and as such the API of the software token is based on PKCS#11 version 2.2. Physical data and control input is translated into logical data and control inputs and passed to the software token via the API. Data output is a result of the API function calls, and status output is provided as return values from API function calls.

[2] DH is approved for key establishment, but is not included in the FIPS mode of operations..
[3] DSA is an approved algorithm, but it is not compliant. It is not included in the FIPS mode of operations.
[4] Elliptic curve is an approved algorithm, but it is not compliant. It is not included in the FIPS mode of operations.

The operating system controls separation of these logical interfaces when the module communicates over the same physical interfaces. Logical interfaces are separated by the structure of the API and the definition of the interfaces. Each input is directed to a particular API call and each output returned from a particular API call.

The operating environment obtains data from various sources, including network and keyboard interfaces, and prepares that data to become input to the software token. The data might be stored on the hard disk before being used as input data.

Table 2 – Logical Interfaces

FIPS 140-2 Interface	Physical Interface	Logical Interface
Data Input	Network, Keyboard	Input parameters of API function calls
Data Output	Network, Video	Output parameters of API function calls
Control Input	Network, Keyboard	API function calls
Status Output	Network, Video	Function calls that return status information and return codes provided by each API function call
Power Interface	GPC Power Connector	None

When the module enters an error state, it no longer will send or receive data. If an error state is encountered, the module will reset and will not send or receive any data until the reset is completed.

The module performs its self tests during the token initialization process. Until token initialization is complete, no data can be processed by the module, thus data output and input are inhibited during self testing. If self testing fails, the module will enter an error state and the initialization routine will fail. When this occurs, any function calls to the token will result in the `CKR_CRYPTOKI_NOT_INITIALIZED` error result, and thus data output will not be possible.

The output data path is physically and logically disconnected from the processes that perform key generation and zeroization. No key information is output through data output interfaces during key generation or zeroization.

Roles, Services and Authentication

The token supports two roles, a *User* and a *Security Officer Role.* The User Role and Security Officer Roles are implicitly assumed by the calling application. The Security Officer Role is primarily responsible for initializing the token for first time use. Table 3 lists the roles and the services that they can access.

Table 3 – Roles and Services

Role	Services and Access	Keys and CSPs
Security Officer	Token initialization (r, w, x)	
	Reading and writing public token objects (e.g. certificates) (r, w)	RSA public keys X.509 certificates
	Self-test (x)	HMAC key
User	Key wrap/unwrap (r,x)	RSA private keys used to wrap Triple-DES or AES keys RSA public keys used to unwrap Triple DES or AES keys
	Encryption/decryption (r,x)	Triple-DES keys AES keys
	Message authentication (x)	HMAC secrets
	Signing (w, x)	RSA private keys
	Verification (r, x)	RSA public keys X.509 certificates
	Reading and writing public token objects (e.g. certificates) (r, w)	RSA public keys X.509 certificates
	Reading and writing private token objects (e.g. private and secret keys) (r, w)	Triple-DES keys AES keys RSA private keys
	Show Status (r)	

Physical Security

FIPS 140-2 level 1 physical security requirements are met by the commercially available general-purpose hardware computing platform on which the module runs. It shall include production-grade components with standard passivation, and a production–grade enclosure with a removable cover.

Operational Environment

The module's operational environment is described above, and consists of a commercially available general-purpose hardware computing platform and Linux Red Hat Enterprise 5 configured for use in single-user mode.

While cryptographic processing is in use, keys and CSPs are protected by process separation.

When the token starts up, it performs an integrity self-check using the HMAC-SHA1 algorithm.

Cryptographic Key Management

Table 4 shows the cryptographic keys and CSPs used by the module in Approved mode of operations. The third column of Table 3 shows the relationship between the keys and CSPs and the security services provided by the module.

The module does not retain keys internally, and keys inside the software boundary are not accessible from outside. The operating system protects memory and process space from unauthorized access. All API functions are executed by the invoking a process in a non-overlapping sequence such that no two API functions will execute concurrently. In addition, libsoftokn3.so does not perform persistent storage of keys.

Table 4 – Cryptographic Keys and CSPs

Algorithm	Description
AES	AES keys, context for AES key wrapping
RSA	X.509 certificate and the associated RSA private key, X.509 certificate's fields, including RSA public key Array of raw (unparsed) X.509 certificates, with RSA public keys
Triple-DES	Triple-DES keys
HMAC secret	The secret value for HMAC, used in HMAC_Create()
FIPS 186-2 RNG	RNG context information, including seed
RSA	RSA private key
RSA	RSA public key

Self-Tests

The following self tests occur during module operations. Messages are logged according to the module logging settings – either to the internal event log or to the console. All of these tests are run without inputs or action from an operator.

Power On Self Tests

The power on self tests consist of a software integrity test, and known answer tests for the cryptographic algorithm implementations.

Software Integrity Test

A HMAC SHA-1 value is calculated on the module and compared to a stored value calculated when the library was built.

Cryptographic Algorithm Self-Tests

The module performs the following Self-Tests at Power-on:

Table 5 – Known Answer Tests

Algorithm	Test Procedure
AES	KAT
Triple-DES	KAT
RSA	Sign/Verify KAT
RNG	KAT,
SHA-1	KAT

Table 6 – Conditional Self-Tests

Algorithm	Test Procedure
RSA	Pairwise Consistency Test
RNG	Continuous Test

Security Requirements

Secure Installation

The cryptographic module is the “libsoftokn3.so” shared object library, version 1.0, that is installed at manufacturing on a Linux operational environment of the ZIX Gateway. The object library can be installed in the standard library path on the destination platform operation system by copying it to the appropriate location. The object library file shall be accompanied by the “libsoftokn3.chk” file which is used to verify file integrity during module startup.
The following steps must be performed to install and initialize the cryptographic module for operating in a FIPS 140-2 compliant manner:

- The OS must be configured to a single user mode of operation
- Upon initialization, the module must run its power-up self tests. Successful completion of the power-up self tests ensures that the module is operating in the FIPS mode of operation.

Invoking the Approved Mode of Operation

The following policy must always be followed in order to achieve a FIPS 140-2 mode of operation:

- As the module has no way of managing keys, any keys that are input or output from applications utilizing the module must be input or output in encrypted form using FIPS approved algorithms.
- Only FIPS “Approved” algorithms may be used in FIPS mode of operation.
- Calling the compound function “SECMOD_DeleteInternalModule(PR_smprintf("%s", SECMOD_GetInternalModule()->commonName))” will place the module in FIPS mode of operations. This will return SECSuccess if the operation succeeded.
- The function “PK11_IsFIPS()” returns true if the module is in FIPS mode of operation.

Design Assurance

All source code and documentation is stored in a version control system called Subversion. Information about Subversion is available at

http://subversion.tigris.org/

The structure of the module’s components corresponds directly to the security policy’s rules of operation. The security mechanisms provided by the software including access control and cryptographic functionality, are addressed in the Security Policy. The Security Policy contains explicit instructions about how the module is accessed.

The libsoftokn3.so file is coded in C.

Mitigation of Other Attacks

The module is not designed to mitigate any other attacks.