Oracle Solaris Userland Cryptographic Framework with SPARC T4 and SPARC T5

Software Version: 1.0 and 1.1; Hardware Version: SPARC T4 (527-1437-01) and SPARC T5 (7043165)

FIPS 140-2 Non-Proprietary Security Policy

Level 1 Validation
Version 1.4

2014-01-10

Table of Contents

Introduction

Purpose

This is a non-proprietary Cryptographic Module Security Policy for the Oracle Solaris Userland Cryptographic Framework with SPARC T4 and SPARC T5 Software Version: 1.0 and 1.1; Hardware Version: SPARC T4 (527-1437-01) and SPARC T5 (7043165) from Oracle Corporation (Oracle). This Security Policy describes how the Solaris Userland Cryptographic Framework with SPARC T4 and SPARC T5 Software Version: 1.0 and 1.1; Hardware Version: SPARC T4 (527-1437-01) and SPARC T5 (7043165) meets the security requirements of FIPS 140-2 (Federal Information Processing Standards Publication 140-2 — *Security Requirements for Cryptographic Modules*) and how to run the module in a secure FIPS 140-2 mode. This policy was prepared as part of the Level 1 FIPS 140-2 validation of the module. FIPS 140-2 details the U.S. Government requirements for cryptographic modules. More information about the FIPS 140-2 standard and validation program is available on the National Institute of Standards and Technology (NIST) Cryptographic Module Validation Program (CMVP) website at http://csrc.nist.gov/cryptval/.

References

This document deals only with operations and capabilities of the module in the technical terms of a FIPS 140-2 cryptographic module security policy. More information is available on the module from the following sources:

- The Oracle Corporation website (http://www.oracle.com) contains information on the full line of products from Oracle.
- The CMVP website (http://csrc.nist.gov/cryptval/) contains contact information for answers to technical or sales-related questions for the module.

Document Organization

The Security Policy document is one document in a FIPS 140-2 Submission Package. In addition to this document, the Submission Package contains:

- Vendor Evidence document
- Finite State Machine
- Other supporting documentation as additional references

With the exception of this Non-Proprietary Security Policy, the FIPS 140-2 Validation Documentation is proprietary to Oracle Corporation and is releasable only under appropriate non-disclosure agreements. For access to these documents, please contact Oracle Corporation.

ORACLE SOLARIS USERLAND CRYPTOGRAPHIC FRAMEWORK WITH SPARC T4 AND SPARC T5 SOFTWARE VERSION: 1.0 AND 1.1; HARDWARE VERSION: SPARC T4 (527-1437-01) AND SPARC T5 (7043165)

Overview

The Oracle Solaris 11 operating system (OS) is a highly configurable UNIX-based operating system that is optimized to quickly and securely deploy services in traditional enterprise data centers, large scale cloud environments and small personal desktop use. Oracle preserves the long-standing guarantee of binary compatibility – applications that run on previous Oracle Solaris releases can still run unchanged on Oracle Solaris 11 within the same processor architecture: x86 or SPARC[1].

The Oracle Solaris 11 OS can be installed on either x86 or SPARC hardware architectures or run in a virtualized environment. The operating system allows one or more processors and multiple hardware peripheral and storage devices to be accessed by multiple users in order to meet user requirements.

Oracle Solaris 11 provides a suite of technologies and applications that create an operating system with optimal performance. Oracle Solaris 11 includes key technologies such as zones, ZFS file system, Image Packaging System (IPS), multiple boot environments, trusted extensions, and cryptographic framework.

The Oracle Solaris OS utilizes two cryptographic modules; one in the Userland space and the second in the Kernel space. The OS uses the Solaris Userland Cryptographic Framework with SPARC T4 and SPARC T5 module for cryptographic functionality for any applications running in user space. It exposes PKCS#11[2] API[3]s, uCrypto APIs, and libmd public interfaces to provide cryptography to any application designed to utilize it.

The OS also utilizes the FIPS-validated Oracle Solaris Kernel Cryptographic Framework module to provide cryptographic functionality for any kernel-level processes that require it. It does this via Oracle-proprietary APIs.

This document will focus on the Oracle Solaris Userland Cryptographic Framework with SPARC T4 and SPARC T5 module. The Oracle Solaris Kernel Cryptographic Framework is discussed in another FIPS 140-2 Non-proprietary Security Policy.

The module meets overall level 1 requirements for FIPS 140-2, and Table 1 describes the level achieved by the module in each of the eleven sections of FIPS 140-2 requirements.

---

[1] SPARC – Scalable Processor Architecture
[2] PKCS #11 – Public Key Cryptography Standards #11
[3] API – Application Programming Interface

Section	Section 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	1
6	Operational Environment	1
7	Cryptographic Key Management	1
8	EMI/EMC	1
9	Self-tests	1
10	Design Assurance	1
11	Mitigation of Other Attacks	N/A

Table 1 – Security Level per FIPS 140-2 Section

Module Specification

The Oracle Solaris Userland Cryptographic Framework with SPARC T4 and SPARC T5 is a software-hybrid module with a multi-chip standalone embodiment. The overall security level of the module is level 1. The following sections will define the physical and logical boundaries of the module.

The cryptographic module is a group of libraries that, collectively, are known as the Oracle Solaris Userland Cryptographic Framework with SPARC T4 and SPARC T5 as well as the SPARC[4] T4 and SPARC T5 processors, which provide cryptographic hardware acceleration. The module provides cryptographic functionality for any application that calls into it. The module provides encryption, decryption, hashing, signature generation and verification, certificate generation and verification, asymmetric key generation, and message authentication functions.
Oracle produces multiple versions of the T4 processor. Oracle affirms that the FIPS module will function the same way and provide the same security services on any of the Oracle-produced T4 processors. This includes the processor in the T4-1 server, T4-1B server, and T4-2 server. This also includes the processor in the T4-4 server which is the same design but runs with a higher clock speed than the version used in T4-1 server.

Oracle also produces multiple versions of the T5 processor. Oracle affirms that the FIPS module will function the same way and provide the same security services on any of the Oracle-produced T5 processors. This includes the processors in the T5-2 server, T5-1B server, T5-4 server and the T5-8 server. This also includes the processors in the M5-32 server which has a six-core architecture and a larger memory cache.

The module makes use of the SPARC T4 and SPARC T5 processor instruction set for acceleration of cryptographic algorithms. The instruction set can only be utilized

[4] SPARC – Scalable Processor Architecture

by the libraries that make up the module. The following algorithms are supported by the SPARC T4 and SPARC T5 processors:

- AES[5]
- Triple-DES[6]
- Diffie-Hellman
- DSA[7]
- ECC[8]
- MD5[9]
- RSA[10]
- SHA-1[11]
- SHA-2

Figure 1 and Figure 2, below, are the logical block diagrams for the module. They highlight the libraries that make up the module in orange, while illustrating the module boundary.

[5] AES – Advanced Encryption Standard
[6] DES – Data Encryption Standard
[7] DSA – Digital Signature Algorithm
[8] ECC – Elliptic-Curve Cryptography
[9] MD5 – Message Digest Algorithm 5
[10] RSA – Rivest, Shamir, and Adleman
[11] SHA – Secure Hash Algorithm

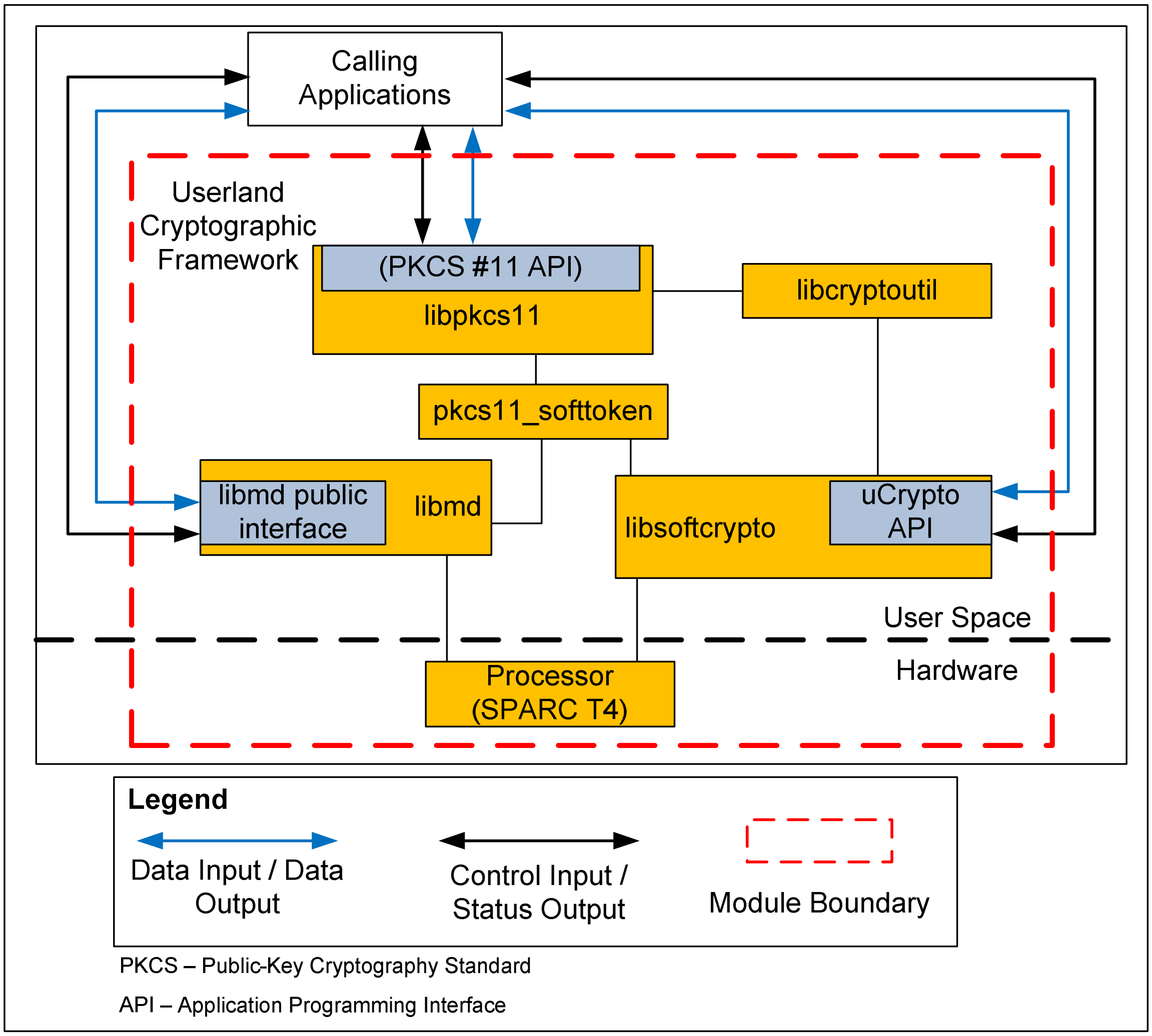

Figure 1 - Oracle Solaris Userland Cryptographic Framework with SPARC T4 and SPARC T5 Logical Block Diagram – T4

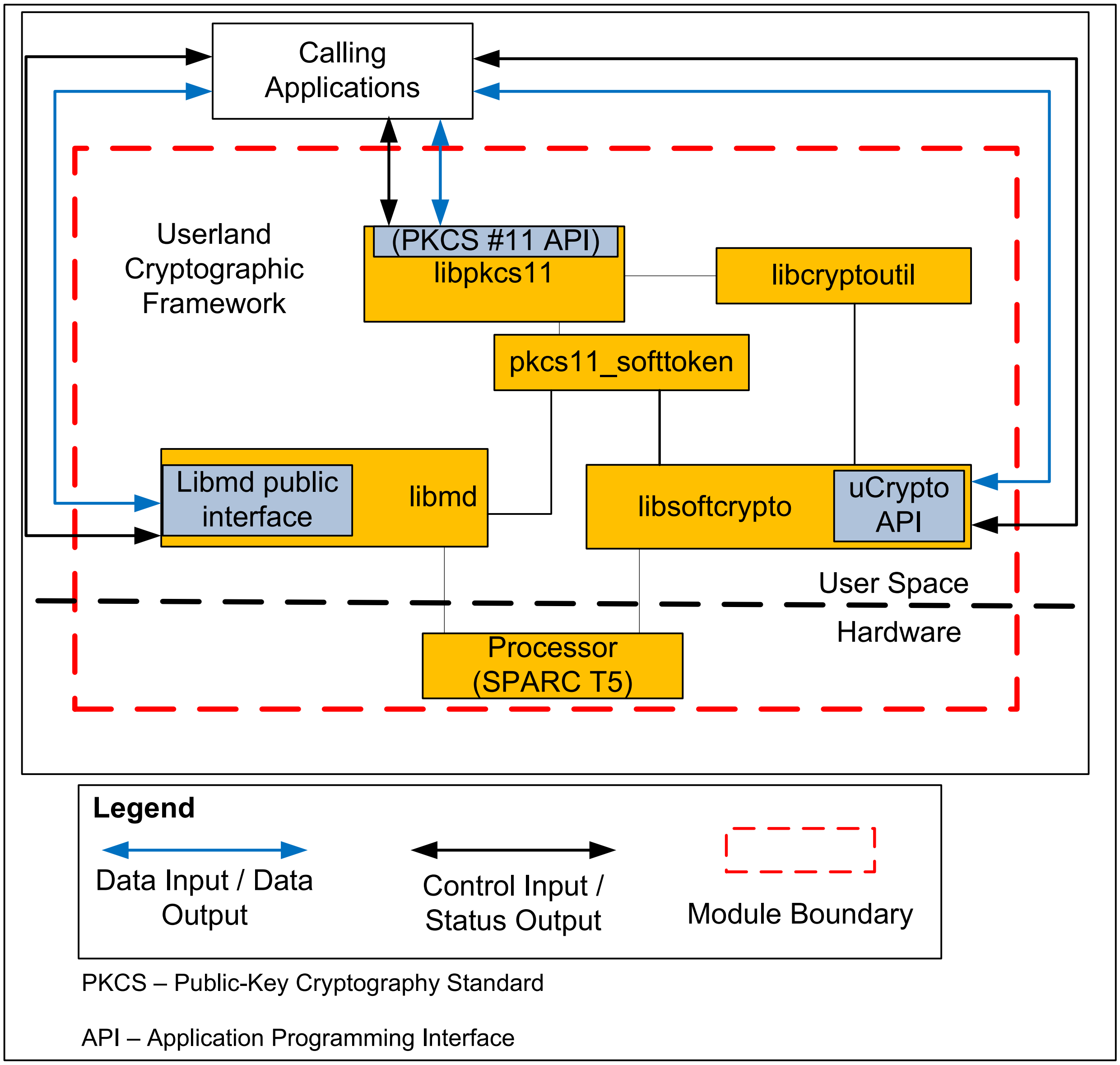

Figure 2 – Oracle Solaris Userland Cryptographic Framework with SPARC T4 and SPARC T5 Logical Block Diagram

Figure 3, below, shows the hardware block diagram for the T4-1 Server appliance, utilizing the SPARC T4 processor that will execute the module. The processor is shown in Figure 5, below.

Figure 4, below, shows the hardware block diagram for the T5-2 Server appliance, utilizing the SPARC T5 processor that will execute the module. The processor is shown in Figure 6, below.

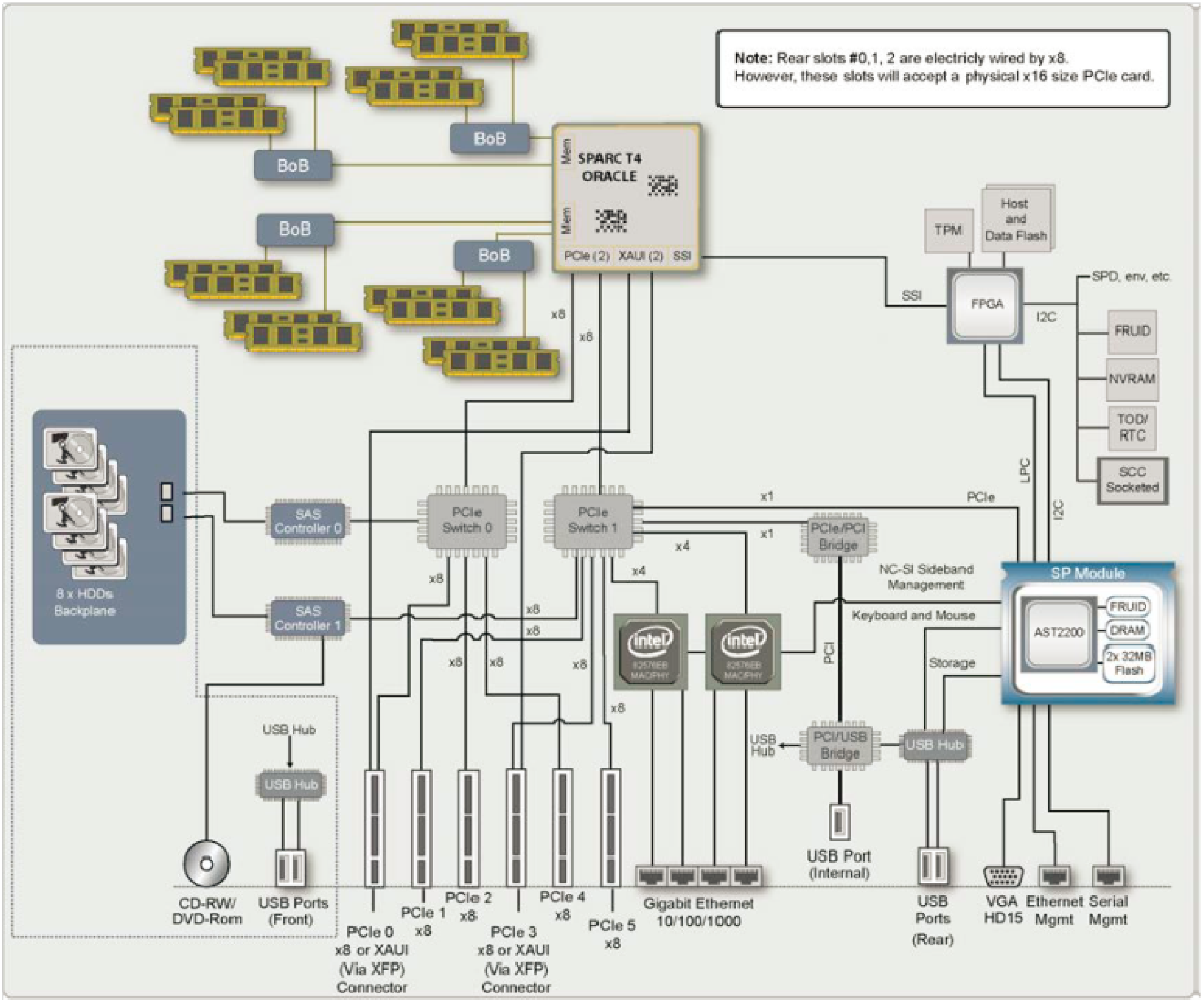

Figure 3 – T4-1 Server Hardware Block Diagram

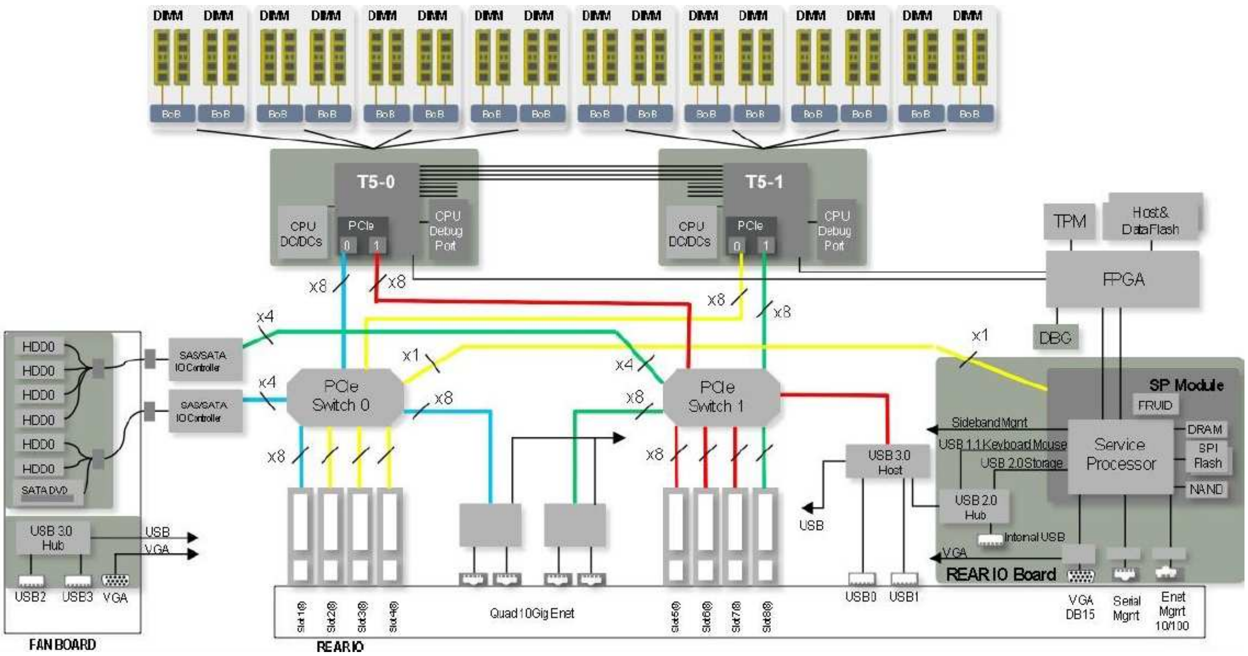

Figure 4 –T5-2 Server Hardware Block Diagram

Figure 5 – T4 Processor

Figure 6 – T5 Processor

Module Interfaces

The module can be accessed in several different ways, depending on which library the calling application is accessing. The module provides three primary interfaces for requesting cryptographic functionality.

- libpkcs11 provides the PKCS #11 interface for accessing the majority of the cryptographic functions of the module, through pkcs11_softtoken, which provides key storage and cryptographic algorithm access.

- libsoftcrypto provides an interface known as uCrypto, which allows access to the cryptographic functions contained within libsoftcrypto, for pkcs11_softtoken and direct calls.

- While the hashing functions can be called from the PKCS#11 interface, libmd provides its own public interface for accessing hashing functions using SHA-1[12], SHA-224, SHA-256, SHA-384, and SHA-512. This interface can be utilized via direct calls to the library.

Table 2, below, shows the interfaces provided by the module.

Figure 7 and Figure 8 below, show the host appliances for the module. These diagrams show the physical ports that are available on the hardware.

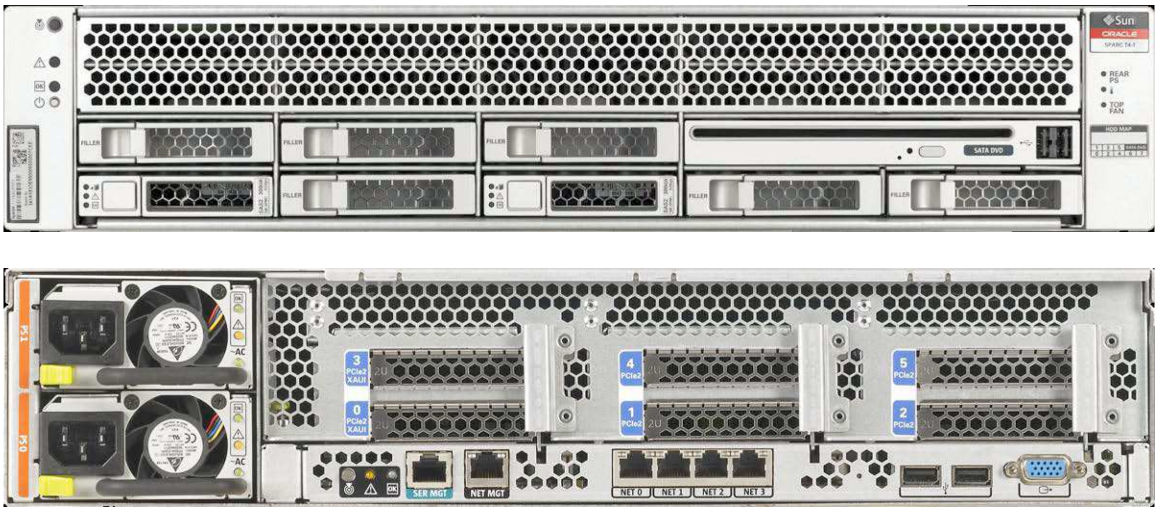

Figure 7 – T4-1 Server Front and Rear Panel Ports (SPARC T4-based)

[12] SHA – Secure Hashing Algorithm

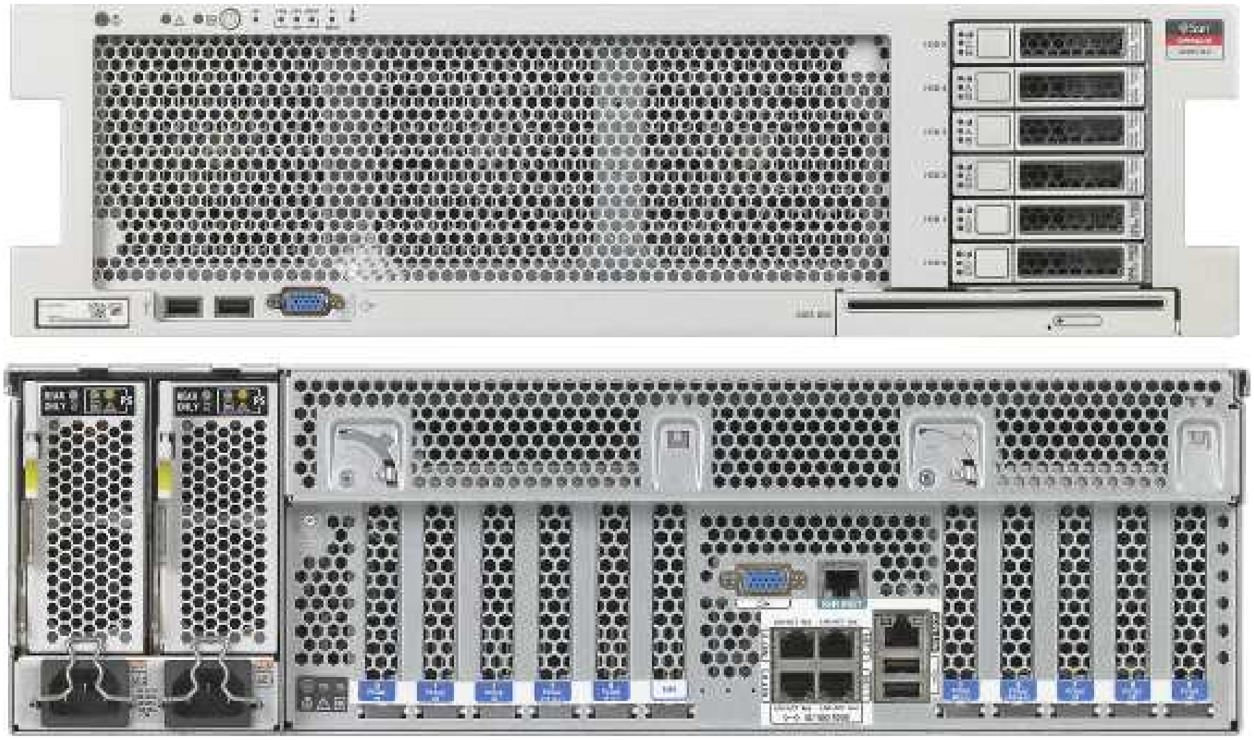

Figure 8 – T5-2 Server Front and Rear Panel Ports (SPARC T5-based)

FIPS 140-2 Logical Interface	**Module Logical Interface**	**Host Appliance Physical Interface**
Data Input	PKCS#11 API, uCrypto API, libmd public interface function calls	Ethernet, USB, SAS[13] port, Serial port (RJ-45), DVD optical drive
Data Output	PKCS#11 API, uCrypto API, libmd public interface function returns	Ethernet, USB, SAS port, Serial port (RJ-45)
Control Input	PKCS#11 API, uCrypto API, libmd public interface function calls	Ethernet, USB
Status Output	PKCS#11 API, uCrypto API, libmd public interface function returns	Ethernet, USB, VGA port, LEDs
Power Input	Not Applicable	Power Port

Table 2 – FIPS 140-2 Logical Interfaces

Roles and Services

The module relies on the host OS for authentication. There are two roles in the module (as required by FIPS 140-2) that operators may assume: a Crypto Officer role and a User role.

[13] SAS – SCSI (Small Computer System Interface) Assisted Storage

Crypto-Officer Role

The Crypto-Officer is any operator on the host appliance with the permissions to utilize the external cryptoadm utility, or a program with the ability to access the module APIs. The Crypto-Officer role has the ability to enable and disable FIPS mode, check the status of the FIPS module, and configure cryptographic operations of the module, including which providers will be available. The Crypto-Officer is able to utilize these services via the cryptoadm commands. Descriptions of the services available to the Crypto-Officer role are provided in Table 3, below.

The Crypto-Officer is also able to utilize all User services, described in Table 4.

Please note that the keys and CSPs listed in the table indicate the type of access required using the following notation:

- Read: The CSP is read.
- Write: The CSP is established, generated, modified, or zeroized.
- Execute: The CSP is used within an Approved or Allowed security function or authentication mechanism

Service	Description	CSP[14]	Type of Access to CSP
Run POST KATs on-demand	Restarting the appliance will force the FIPS self-tests to run when the module is loaded. Calling the fips140_post() function will call the Power-On Self-Tests.	Crypto-Officer credentials	Execute
Module Initialization	Use external cryptoadm utility to initialize the FIPS state.	Crypto-Officer credentials	Execute
Module Configuration	Use external cryptoadm utility to configure the module.	Crypto-Officer credentials	Execute
Zeroize keys	Format operation on the host appliance's hard drive	Crypto-Officer credentials	Execute

Table 3 – Crypto-Officer Services

The credentials for the Crypto-Officer are not considered CSPs, as requirements for module authentication are not enforced for Level 1 validation. The credentials are provided to the host OS, and are not part of the module.

User Role

The User role is able to utilize the cryptographic operations of the module, through its APIs. Descriptions of the services available to the User role are provided in the Table 4 below.

[14] CSP – Critical Security Parameter

Service	Description	CSP	Type of Access to CSP
Symmetric encryption	Encrypt a block of data using a symmetric algorithm	AES[15] key Triple DES[16] key	Execute
Symmetric decryption	Decrypt a block of data using a symmetric algorithm	AES key Triple DES key	Execute
Asymmetric key wrapping	Encrypt a block of data using an asymmetric algorithm	DSA[17] public key RSA[18] public key	Execute
Asymmetric key unwrapping	Decrypt a block of data using an asymmetric algorithm	DSA private key RSA private key	Execute
Signature Generation	Generate a signature	DSA public key RSA public key	Execute
Signature Verification	Verify a signature	DSA private key RSA private key	Execute
Asymmetric keypair generation	Generate a keypair for use in an asymmetric algorithm	DSA public key DSA private key	Write
Generate Elliptic-Curve keypair	Generate an asymmetric keypair for use in Elliptic-Curve DSA cryptographic operations	ECDSA[19] public key ECDSA private key	Write
Hashing	Perform a hashing operation on a block of data, using SHA-1, SHA-224, SHA-256, SHA-384, or SHA-512	N/A	N/A
HMAC[20] signing	Perform a hashing operation on a block of data, using a keyed Hashed Message Authentication Code with any of the hashing operations listed above	HMAC key	Execute
Key derivation	Derive a session key using Elliptic-Curve Diffie-Hellman	Elliptic-Curve Diffie-Hellman keypair	Write
Random Number Generation	Generate random numbers	Userland FIPS 186-2 Seed Userland FIPS 186-2 Seed Key	Execute

Table 4 - User Services

Note that non-FIPS-Approved algorithms can also be used as part of these services, when the module is not operating in a FIPS-Approved mode.

Physical Security

Oracle Solaris Userland Cryptographic Framework with SPARC T4 and SPARC T5 is a software-hybrid module, which FIPS defines as a multi-chip standalone

[15] AES – Advanced Encryption Standard
[16] DES – Data Encryption Standard
[17] DSA – Digital Signature Algorithm
[18] RSA – Rivest, Shamir, Adleman
[19] ECDSA – Elliptic-Curve Digital Signature Algorithm
[20] HMAC – (Keyed-) Hash-based Message Authentication Code

cryptographic module. The module hardware is made up of production-grade components that include standard passivation techniques.

Operational Environment

The module operates as part of the Oracle Solaris 11.1 SRU3 and 11.1 SRU5.5 Operating System. The module is programmed to utilize SPARC special instructions sets for hardware-accelerated cryptography. The module has been tested on Oracle Solaris 11.1 SRU3 OS and Oracle Solaris 11.1 SRU5.5 OS, running on a T4-1 Server, using a SPARC T4 processor. Additionally the module has been tested on Oracle Solaris 11.1 SRU3 OS and Oracle Solaris 11.1 SRU5.5 OS, running on a T5-2 Server, using a SPARC T5 processor. The processor shall be validated as part of the module.

The Crypto-Officer shall ensure that the OS is configured to a Single-User mode of operation. Each calling application calls its own instance of the module. During execution of this instance, the calling application will be the only operator of the module. This instance cannot be called by other applications. Therefore the calling application is the single-user of the module.

Cryptographic Key Management

The module implements the following FIPS-approved algorithms:

Key or CSP	**Certificate Number**	
	11.1 SRU3	**11.1 SRU5.5**
Symmetric Key		
AES: ECB[21], CBC[22], CFB[23]-128, CCM[24], GCM[25], and CTR[26] modes for 128, 192, and 256-bit key sizes	#2310	#2572
Triple DES: CBC and ECB mode for keying option 1	#1457	*#1558*
Asymmetric Key		
RSA PKCS#1.5 signature generation: 2048-bit (SHA-256, SHA-384, SHA-512) RSA PKCS#1.5 signature verification: 1024- , 2048-bit (w/ SHA-1, SHA-256, SHA-384, SHA-512)	#1193	*#1319*
DSA 2048-, 3072-bit Key generation; Signature generation: 2048-, 3072-bit Signature verification: 1024-, 2048-, 3072-bit	#727	#787

21 ECB – Electronic Codebook
22 CBC – Cipher-Block Chaining
23 CFB – Cipher Feedback
24 CCM – Counter with CBC-MAC
25 GCM – Galois/Counter Mode
26 CTR – Counter

Key or CSP	Certificate Number	
	11.1 SRU3	11.1 SRU5.5
ECDSA key generation, signature generation/verification: P-192, -224, -256, -384, -521; K-163, -233, -283, -409, -571; B-163, -233, -283, -409, -571	#375	*#444*
Secure Hashing Standard (SHS)		
SHA-1, SHA-224, SHA-256, SHA-384, SHA-512	#1994	#1994
(Keyed-) Hash-based Message Authentication		
HMAC SHA-1, HMAC SHA-224, HMAC SHA-256, HMAC SHA-384, HMAC SHA-512	#1424	#1594
Random Number Generation		
Userland FIPS 186-2 Random Number Generator	#1153	#1224
Key Agreement Scheme (KAS)		
Diffie-Hellman (2048 – 8192-bit)	Allowed	Allowed
Elliptic-Curve Diffie-Hellman (224 – 571-bit)	Allowed	Allowed

Table 5 – FIPS-Approved Algorithm Implementations

NOTE: The following security functions have been deemed "deprecated" or "restricted" by NIST. Please refer to NIST Special Publication 800-131A for further details.

- After December 31, 2013, key lengths providing less than 112 bits of security strength shall not be used in the Approved mode of operation to generate digital signatures or keys.
- RSA (encrypt/decrypt, sign/verify operations) provides 112 bits of encryption strength, for 2048-bit keys. RSA provides higher bits of encryption strength with higher key sizes; non-compliant with less than 112 bits of encryption strength.
- RSA (key wrapping; key establishment methodology) provides 112 bits of encryption strength, for 2048-bit keys. RSA provides higher bits of encryption strength with higher key sizes; non-compliant with less than 112 bits of encryption strength.
- Diffie-Hellman (key agreement; key establishment methodology) provides 112 bits of encryption strength, for 2048-bit public keys. Diffie-Hellman provides higher bits of encryption strength with higher key sizes; non-compliant with less than 112 bits of encryption strength. After December 31, 2013, $|n| \leq 223$ bits shall not be used in a key agreement scheme. Please see NIST Special Publication 800-131A for further details.
- Elliptic-Curve Diffie-Hellman (key agreement; key establishment methodology) provides between 112 and 256 bits of encryption strength; non-compliant with less than 112 bits of encryption strength. After December 31, 2013, the use of $|n| \geq 224$ is deprecated. Values of $|n| < 224$ shall not be used. Please see NIST Special Publication 800-131A for further details.
- As of January 1, 2014, the use of the RNGs specified in FIPS 186-2, [X9.31] and ANS [X9.62] are deprecated from 2011 through December 31, 2015, and disallowed after 2015.

The module implements the following FIPS-Approved algorithm, however the implementation has not been validated and, as such, shall not be used in the FIPS-Approved mode of operation:

- Two-key Triple-DES
- SHA-512/224
- SHA-512/256

Additionally, the module implements the following non-FIPS-approved algorithms:

- MD5[27]
- MD4
- RC4[28]
- DES
- Blowfish
- AES XCBC-MAC[29] – 128-, 192-, 256-bit
- DSA key generation – 512-, 1024-bit
- DSA signature generation – 512-, 1024-bit
- DSA signature verification – 512-bit
- RSA signature generation – 256-, 512-, 1024-bit
- RSA signature verification – 256-, 512-bit
- Diffie-Hellman – 64-, 128-, 256-, 512-, 1024-bit public keys
- Elliptic-Curve Diffie-Hellman – 112 to 223-bit public keys

The module generates cryptographic keys whose strengths are modified by available entropy. The CSPs that the module supports are listed in Table 6, below.

[27] MD5 – Message Digest Algorithm 5
[28] RC4 – Rivest Cipher 4
[29] XCBC-MAC – Extended Cipher-Block Chaining Message Authentication Code

Key or CSP	Key type	Generation	Output	Storage	Zeroization	Use
AES key	AES 128-, 192-, 256-bit key	Imported (passed as an attribute in an argument)	Output from module through Data Output interface	Reference Pointer Stored in volatile memory during execution Exported to hard drive of GPC	Zeroized upon completion of operation or reboot Zeroized via format hard-drive service	Symmetric encryption
AES GCM IV	Random data	Imported (passed as an attribute in an argument)	Never output from module	Reference Pointer Stored in volatile memory during execution	Zeroized upon completion of operation or reboot	IV input to AES GCM function
Triple DES key	Triple DES 168-bit key	Imported (passed as an attribute in an argument)	Output from module through Data Output interface	Reference Pointer Stored in volatile memory during execution Exported to hard drive of GPC	Zeroized upon completion of operation or reboot Zeroized via format hard-drive service	Symmetric encryption
RSA public key	RSA 1024[30]-, 2048-, 4096-, 8192-bit key	Imported (passed as an attribute in an argument)	Output from module through Data Output interface	Reference Pointer Stored in volatile memory during execution Exported to hard drive of GPC	Zeroized upon completion of operation or reboot Zeroized via format hard-drive service	Key wrapping, certificate generation, certificate verification, signature verification

[30] Please note that RSA 1024-bit key used for signature verification is considered legacy-use after 2010, and is disallowed for signature generation and key transport after 2013.

Key or CSP	Key type	Generation	Output	Storage	Zeroization	Use
RSA private key	RSA 2048-, 4096-, 8192-bit key	Imported (passed as an attribute in an argument)	Output from module through Data Output interface	Reference Pointer Stored in volatile memory during execution Exported to hard drive of GPC	Zeroized upon completion of operation or reboot Zeroized via format hard-drive service	Key wrapping, certificate generation, certificate verification, signature generation
RSA signature	RSA 2048-, 4096-, 8192-bit signature	Imported (passed as an attribute in an argument) Generated internally	Output from module through Data Output interface	Reference Pointer Stored in volatile memory during execution Exported to hard drive of GPC	Zeroized upon completion of operation or reboot Zeroized via format hard-drive service	Signing data, signature verification
DSA public key	DSA 2048-, 3072-bit key	Imported (passed as an attribute in an argument) Generated internally	Output from module through Data Output interface	Reference Pointer Stored in volatile memory during execution Exported to hard drive of GPC	Zeroized upon completion of operation or reboot Zeroized via format hard-drive service	Signature verification
DSA private key	DSA 224-, 256-bit private key	Imported (passed as an attribute in an argument) Generated internally	Output from module through Data Output interface	Reference Pointer Stored in volatile memory during execution Exported to hard drive of GPC	Zeroized upon completion of operation or reboot Zeroized via format hard-drive service	Signature generation

Key or CSP	Key type	Generation	Output	Storage	Zeroization	Use
ECDSA public key	ECDSA 163-, 192-, 224-, 233-, 256-, 283-, 384-, 409-, 512-, 571-bit public key	Imported (passed as an attribute in an argument) Generated internally	Output from module through Data Output interface	Reference Pointer Stored in volatile memory during execution Exported to hard drive of GPC	Zeroized upon completion of operation or reboot Zeroized via format hard-drive service	Encrypting data, verifying signature
ECDSA private key	ECDSA 163-, 192-, 224-, 233-, 256-, 283-, 384-, 409-, 512-, 571-bit private key	Imported (passed as an attribute in an argument) Generated internally	Output from module through Data Output interface	Reference Pointer Stored in volatile memory during execution Exported to hard drive of GPC	Zeroized upon completion of operation or reboot Zeroized via format hard-drive service	Decrypting data, digitally signing data
HMAC key	Secret key for HMAC	Imported	Never output from module	Reference Pointer Stored in volatile memory during execution Exported to hard drive of GPC	Zeroized upon completion of operation or reboot	Message Integrity/Authentication
ECDH[31] private key	P-192, -224, -256, -384, -521; K-163, -233, -283, -409, -571; B-163, -233, -283, -409, -571 private key	Imported (passed as an attribute in an argument) Generated internally	Output from module through Data Output interface	Reference Pointer Stored in volatile memory during execution Exported to hard drive of GPC	Zeroized upon completion of operation or reboot Zeroized via format hard-drive service	Decryption

[31] ECDH – Elliptic-Curve Diffie-Hellman

Key or CSP	Key type	Generation	Output	Storage	Zeroization	Use
ECDH public key	P-192, -224, -256, -384, -521; K-163, -233, -283, -409, -571; B-163, -233, -283, -409, -571 public key	Imported (passed as an attribute in an argument) Generated internally	Output from module through Data Output interface	Reference Pointer Stored in volatile memory during execution Exported to hard drive of GPC	Zeroized upon completion of operation or reboot Zeroized via format hard-drive service	Encryption
Diffie-Hellman private key	224-, 256-, 384-bit private key	Generated internally	Output from module through Data Output interface	Plaintext in volatile memory	Cleared on session close	Decryption
Diffie-Hellman public key	2048-, 3072-, 4096-, 8192-bit public key	Generated internally	Output from module through Data Output interface	Plaintext in volatile memory	Cleared on session close	Encryption
Userland RNG[32] Seed	20-byte Hexadecimal string	Generated internally	Never output from module	Plaintext in volatile memory	Zeroized upon completion of operation or reboot	To calculate SHA-1 string in FIPS 186-2 RNG
Userland RNG Seed Key	20-byte SHA-1 Digest	Generated internally	Never output from module	Plaintext in volatile memory	Zeroized upon completion of operation or reboot	To calculate SHA-1 string in FIPS 186-2 RNG

Table 6 – Listing of Key and Critical Security Parameters

[32] RNG – Random Number Generator

Self-Tests

In order to prevent any secure data from being released, it is important to test the cryptographic components of a security module to ensure all components are functioning correctly.

Power-Up Self-tests

To confirm correct functionality, the software library performs the following self-tests:

- Software Integrity Test (HMAC SHA-1)
- Known Answer Tests (KATs)
    - AES KAT
    - Triple-DES KAT
    - RSA sign/verify KAT
    - SHA-1, SHA-224, SHA-256, SHA-384, SHA-512 HMAC KAT
    - SHA-1, SHA-224, SHA-256, SHA-384, SHA-512 KAT
    - FIPS 186-2 RNG KAT
- Pairwise Consistency Tests
    - DSA signature generation/verification
    - ECDSA signature generation/verification

Conditional Self-tests

The Solaris Userland Cryptographic Framework with SPARC T4 and SPARC T5 performs the following conditional self-tests:

- DSA key generation pairwise consistency test
- ECDSA key generation pairwise consistency test
- FIPS 186-2 continuous random number generator test

Data output from the module is inhibited, while performing self-tests. Should any of the power-up self-tests or conditional self-tests fail, the modules will cease operation, inhibiting any further data output from the modules. The modules will need to reboot and perform power-up self-tests. Successful completion of the power-up self-tests will return the module to normal operation.

Mitigation of Other Attacks

This section is not applicable. The module does not claim to mitigate any attacks beyond the FIPS 140-2 Level 1 requirements for this validation.

SECURE OPERATION

The Oracle Solaris Userland Cryptographic Framework with SPARC T4 and SPARC T5 meets Level 1 requirements for FIPS 140-2. The sections below describe how to place and keep the module in a FIPS-approved mode of operation.

Initial Setup

The Oracle Solaris Userland Cryptographic Framework with SPARC T4 and SPARC T5 module is part of the Oracle Solaris operating system. Immediately after initial installation of the Oracle Solaris operating system, the Crypto-Officer must install a specific service to ensure sufficient entropy for random number generation functions within the module. The "fips-random service" is available from Oracle's standard support repository.

The fips-random service must be loaded onto the operating system's host appliance. The Crypto-Officer must ensure the package repository is up to date using:

http://pkg.oracle.com/solaris/support

The Crypto-Officer will then install the package by running the following command:

```
pkg install pkg:service/security/fips-random
```

Once the package is added, the Oracle Solaris operating system must be rebooted. It should be noted that this package can alternatively be listed in the Auto Install manifest so that it will be added and run at the initial operating system install time.

The next time that the Solaris operating system is booted, the Crypto-Officer must use cryptoadm to make the changes necessary to enable FIPS mode. The Crypto-Officer must verify that pkcs11_softtoken is present in the list of providers, and is not disabled, using the "cryptoadm list" command. If it is not present, the Crypto-Officer must use the "cryptoadm enable <*provider name*> <*mechanism list*>" command to enable /usr/lib/security/$ISA/pkcs11_softtoken.so. The host OS will not boot properly without pkcs11_softtoken being enabled. The pkcs11_softtoken provider is the only provider installed in the validated configuration. MD5 is included as part of this provider, for TLS purposes only. The use of MD5 is allowed in the TLS or DTLS[33] protocol only. MD5 shall not be used as a general hash function in an Approved mode of operation.

[33] DTLS – Datagram Transport Layer Security

The Crypto-Officer must input the command “cryptoadm disable provider=’/user/lib/security$ISA/pkcs11_kernel.so’ all”. This command disables the module’s direct access to functions operating in the kernel.

Next, the Crypto-Officer must create a new boot environment, using the “beadm create <name>” command, where the name is one provided by the Crypto-Officer. This creates a boot environment which is a clone of the currently running environment, to be used if there is a panic or other error with initialization. The Crypto-Officer must then input the command “cryptoadm enable fips-140”, in order to enable FIPS mode. The module must then be restarted by a full system reboot. Once the module loads, it will perform power-on cryptographic self-tests. Once all tests are successful, the module will begin to operate in a FIPS-Approved mode.

Crypto-Officer Guidance

The Crypto-Officer is responsible for making sure the module is running in FIPS-Approved mode of operation and to ensure that only FIPS-Approved algorithms are utilized. The following algorithms and key sizes, provided by the module, cannot be used in FIPS-Approved mode of operation:

- MD5 – for non-TLS uses
- MD4
- RC4
- DES
- Blowfish
- 2-key Triple DES
- AES XCBC-MAC (128-, 192-, 256-bit)
- DSA key generation – 512-, 1024-bit
- DSA signature generation – 512-, 1024-bit
- DSA signature verification – 512-bit
- RSA signature generation – 256-, 512-, 1024-bit
- RSA signature verification – 256-, 512-bit

Initialization

It is the Crypto-Officer’s responsibility to configure the module into the FIPS-Approved mode.

Management

Using the commands available to the Crypto-Officer, outlined in Table 3, the cryptoadm utility can be used to configure and manage the module.

Zeroization

As shown in Table 6, certain keys are stored on the host appliance’s hard drive. A format of the host appliance’s hard-drive will zeroize all keys.

User Guidance

It is the responsibility of the User calling applications to ensure that only FIPS-Approved algorithms and providers are being utilized by their commands. The User is required to operate the module in a FIPS-Approved mode of operation. In order to maintain FIPS-mode, the User must do the following:

- Only utilize the module interfaces to call FIPS-Approved algorithms

ACRONYMS

AES	Advanced Encryption Standard
API	Application Programming Interface
CBC	Cipher-Block Chaining
CCM	Counter with CBC-MAC
CFB	Cipher Feedback
CMVP	Cryptographic Module Validation Program
CSP	Critical Security Parameter
CTR	Counter
DES	Data Encryption Standard
DSA	Digital Signature Algorithm
DTLS	Datagram Transport Layer Security
DVD	Digital Versatile Disc
ECB	Electronic Codebook
ECC	Elliptic-Curve Cryptography
ECDH	Elliptic-Curve Diffie-Hellman
ECDSA	Elliptic-Curve Digital Signature Algorithm
EMC	Electromagnetic Compatibility
EMI	Electromagnetic Interference
ESP	Encapsulating Security Payload
FCC	Federal Communication Commission
FIPS	Federal Information Processing Standard
GCM	Galois/Counter Mode
GPC	General Purpose Computer
HD	High Density
HMAC	(Keyed-) Hash-based Message Authentication Code
KAT	Known Answer Test
LED	Light Emitting Diode
MAC	Message Authentication Code
MD5	Message Digest Algorithm 5
NIST	National Institute of Standards and Technology
OS	Operating System
PKCS	Public Key Cryptography Standards
RC4	Rivest Cipher 4
RCI	Remote Cabinet Interface
RJ	Registered Jack
RNG	Random Number Generator
RSA	Rivest Shamir and Adleman
SAS	Small Computer System Interface-Assisted Storage
SHA	Secure Hash Algorithm
SPARC	Scalable Processor Architecture
UPC	Usage Parameter Control
USB	Universal Serial Bus
VGA	Video Graphics Array
XCBC-MAC	Extended Cipher-Block Chaining Message Authentication Code