FIPS 140-2 Level 1 Security Policy for Cisco Secure ACS FIPS Module

OL-13030-01

Contents

Overview

The Cisco Secure ACS FIPS Module Version 1.1 is a software cryptographic library that provides cryptographic services to the Cisco Access Control Server (ACS) application. Essentially the Secure ACS FIPS module provides FIPS compliant cryptography supporting AAA for IEEE 802.11i security (WPA2) with EAP protocols like EAP-TLS, EAP-FAST, PEAP with RADIUS Key Wrap functionalities.

Americas Headquarters:
Cisco Systems, Inc., 170 West Tasman Drive, San Jose, CA 95134-1706 USA

The Secure ACS FIPS Module does not implement any of the protocols directly. Instead, the Secure ACS FIPS Module provides the cryptographic primitives and functions to all ow a developer to implement various EAP protocols. The software module contains implementations of the following Approved cryptographic algorithms:

- AES Key Wrap
- AES
- RSA
- SHA-1
- HMAC SHA-1
- ANSI X9.31 PRNG

For FIPS 140-2 purposes the Cisco Secure ACS FIPS module is classified as a multi-chip standalone module. The logical cryptographic boundary of the module is the Cisco Secure ACS FIPS module which comprises of CryptoLib DLL. The physical cryptographic boundary of the module is the enclosure of the computer system on which the Cisco Secure ACS FIPS module is executing.

The Cisco Secure ACS FIPS module runs in the operational environment of a standard Intel-based computer running the Windows 2000 or Windows 2003 operating systems. The term "module" is used to refer to the Cisco Secure ACS FIPS module working in the operational environment described above. The module performs no communications other than with the process that calls it.

Note This document may be copied in its entirety and without modification. All copies must include the copyright notice and statements on the last page.

Security Requirements

Cryptographic Module Specification

The cryptographic module is the "Cisco Secure ACS FIPS Module, version 1.1" running in the operational environment of a standard Intel-based computer running the Windows 2000 or Windows 2003 operating systems. 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***

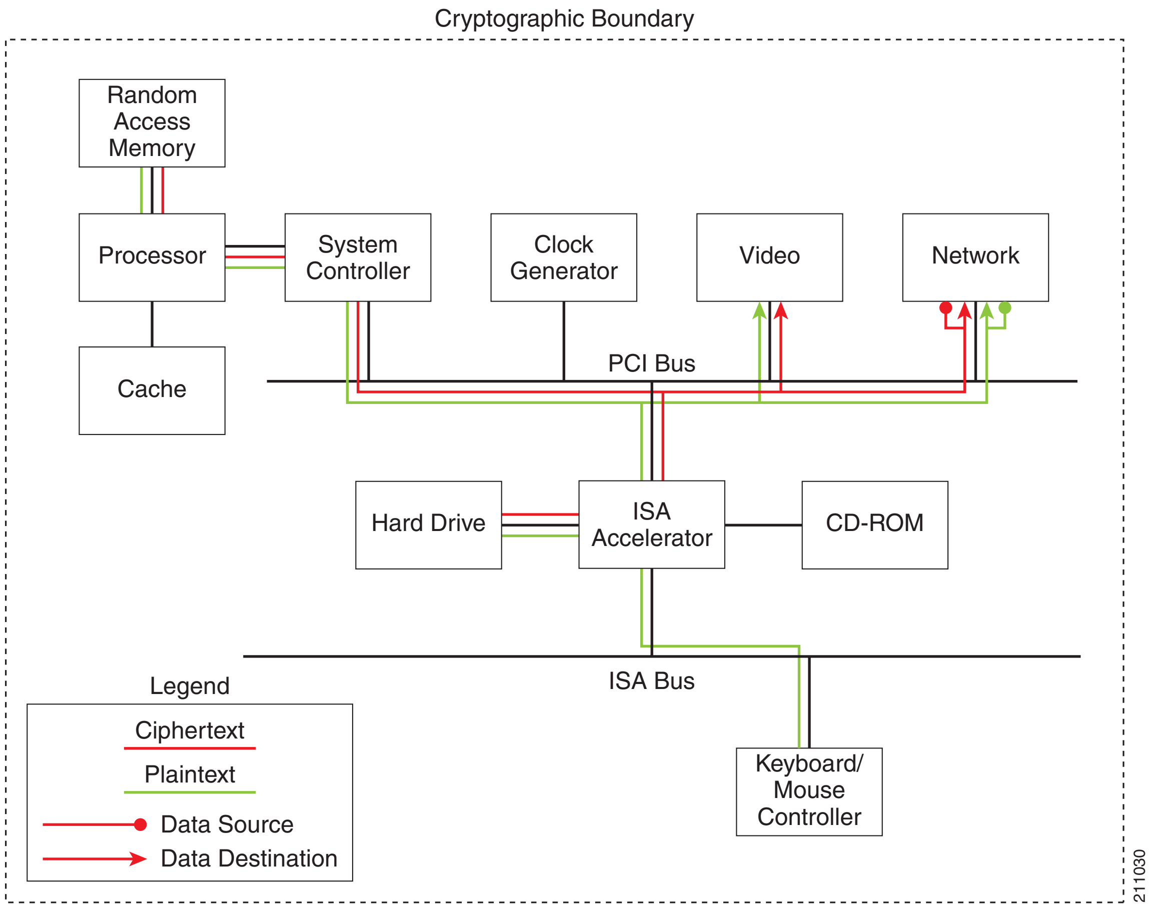

Figure 1 shows the functional block diagram for the computer on which the module runs. All components shown in the diagram are within the physical cryptographic boundary of the module, and the diagram shows interconnections among the major components of the module. Dashed lines represent connections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

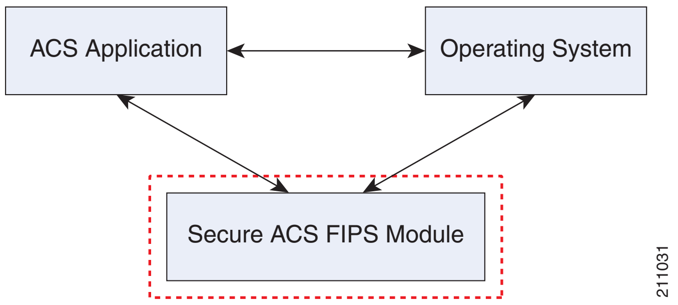

Figure 2 is a functional block diagram that demonstrates how the Cisco Secure ACS FIPS module fits into the overall operational environment. The crypto boundary is represented as a dotted red line which consists of the Cisco Secure ACS FIPS Module. The functionality of the module is exercised by the Cisco ACS application.

The FIPS mode initialization is performed when the application invokes the CryptoLib DLL with CryptoLibrary::Init() function.

When the module is in FIPS mode all security functions and cryptographic algorithms are performed using FIPS Approved functions.

Security Functions

Table 1 lists the cryptographic algorithms implemented by the module.

Table 1 Security Functions

Algorithm	Approved	Algorithm Certificate Number
AES (Encrypt/Decrypt)	Y	566
RSA (Sign/Verify)	Y	263
SHA-1	Y	632
HMAC SHA-1	Y	303
ANSI X9.31 RNG	Y	331
AES Key Wrap	Y	—
RSA (Encrypt/Decrypt)	N (1024-4096 bit key size; permitted for use in FIPS approved mode)	—
Diffie Hellman	N (2048 bit key size; permitted for use in FIPS approved mode)	—
MD5	N	—
MD2	N	—
MD4	N	—
HMAC-MD5	N	—
RC4	N	—
RC2	N	—
DES	N	—

In the FIPS approved mode, the module supports AES key wrap, AES for encryption/decryption, RSA (X.509 certificates) for authentication, key transport and module integrity test, Diffie Hellman for key agreement and HMAC-SHA-1 for message authentication.

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 Cisco Secure ACS FIPS module. Physical data and control input is translated into logical data and control inputs and passed to the module 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.

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 module. Table 2 describes the logical interfaces.

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 Cisco Secure ACS FIPS module performs its self tests during the module initialization process. Until the self tests are 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 will fail. When this occurs, any function calls to the module will result in an error, 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 Cisco Secure ACS FIPS module supports two roles, a User and a Crypto Officer Role. The User Role and Crypto Officer Role are implicitly assumed by the calling application. Calling applications will implicitly choose a role based on the module service requested. Moreover, a calling application will assume only one role in a particular instance in time.

Crypto Officer Role: The Crypto officer can install and initialize the Cisco Secure ACS FIPS module. Other tasks performed by the Crypto Officer include key entry, initiate the power-on self-tests on demand and check the status of the cryptographic module.

User Role: The user role can access the module's API functions to perform cryptographic operations. This role can also initiate self tests on demand and check the status of the module.

Table 3 lists the roles and the services that they can access.

Table 3 Roles and Services

Role	Services and Access	Interface	Keys and CSPs
Crypto Officer	Module initialization (r, w, x)	CryptoLibrary::Init()	—
	Key Entry (r, w, x)	CryptoLibrary::GetKeyWrapInterface() CryptoLibrary::GetCertManagerInterface()	KEK, MACK, RSA private keys, HMAC key
	Show status (r, x)	ICLAuditInterface()	—
	Self-test (x)	CryptoLibrary::GetFipsControlInterface()	—
User	AES Key Wrap (r, w, x)	CryptoLibrary::GetKeyWrapInterface()	KEK, MACK, PMK, Random Nonce
	EAP-TLS (r, w, x)	CryptoLibrary::GetSslInterface()	RSA keys, DH keys, TLS Master secret, TLS encryption key (AES), TLS integrity key (HMAC-SHA-1), TLS session key
	EAP-FAST (r, w, x)	CryptoLibrary::GetSslInterface()	TLS encryption key (AES), TLS integrity key (HMAC-SHA-1), TLS Master secret for EAP-FAST, EAP-FAST Master key, KEK, PAC key, Session Key seed, IMCK, EAP-FAST Master session key
	PEAP (r, w, x)	CryptoLibrary::GetSslInterface()	TLS encryption key (AES), TLS integrity key (HMAC-SHA-1), PEAP tunnel key, PEAP master session key
	Cryptographic Primitives	CryptoLibrary::GetCryptoPrimitivesInterface() CryptoLibrary::GetNonFIPSCompliantInterface()	General symmetric and asymmetric keys
	Self-Test (x)	CryptoLibrary::GetFipsControlInterface()	—
	Show Status (r, x)	ICLAuditInterface()	—

Physical Security

Because the module is validated as a FIPS 140-2 level 1 software module and provides no physical security protection mechanisms, this requirement is not applicable.

Operational Environment

The module's operational environment is described above, and consists of a commercially available general-purpose hardware computing platform and Windows 2000 or Windows 2003 operating system configured for use in single-user mode. Both of these operating systems were used for validation testing.

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

When the module starts up, it performs an integrity self-check by verifying an RSA with SHA-1 digital signature.

Cryptographic Key Management

The module supports AES RADIUS key wrap, AES for encryption/decryption, RSA for authentication and key transport, Diffie Hellman for key agreement and HMAC-SHA-1 for message authentication and module integrity.

Key Generation

All keys that are generated in the module are generated using the ANSI X9.31 FIPS Approved random number generator. The Cisco Secure ACS FIPS Module supports generation of the EAP-FAST PAC key, EAP-FAST Master Key, the random nonce, and the DH key pair.

Key Establishment

The TLS session keys, i.e., the TLS encryption key and the integrity key are established during the TLS handshake using asymmetric key establishment protocols like RSA key transport or Diffie Hellman Key agreement.

The module supports DH key sizes of 2048 bits and RSA key sizes of 1024-4096 bits. These key establishment mechanisms do not provide sufficient protection of the key being transported upon. However, as allowed by NIST SP800-57, the Diffie Hellmann Key agreement, key establishment methodology provides 112 and bits of encryption strength. RSA Key wrapping, key establishment methodology provides between 80-bits and 152-bits of encryption strength per NIST 800-57.

The EAP-FAST and PEAP protocol handshakes occur over a TLS session. All the EAP-FAST and PEAP keys are established and distributed to the supplicant over a TLS tunnel encrypted with the TLS negotiated cipher suites. The Session Keys (aka PMK) is established at the end of EAP-TLS, EAP-FAST, or PEAP handshakes.

Additionally, the module also uses the NIST AES Key Wrap specification (Draft) to protect transmission of the PMK. The KEK is a 128-bit AES key. This key wrapping methodology provides 128-bits of encryption strength.

As noted in the Overview of this Security Policy, the module only provides the cryptographic functions necessary for calling applications to implement the key establishment protocols noted above.

Key Entry/Output

The RSA key pair and the AES Key wrap keys (i.e. the KEK and MACK) are electronically input into the module in plain-text form by the Crypto Officer. The KEK and MACK Keys are used for AES Key Wrapping the PMK. The KEK and MACK are not output from the module. Encrypted key output of PMK is performed when the module sends the PMK to the authenticator, protected by AES key wrap.

All EAP-FAST and PEAP keys are entered and output from the module encrypted with the TLS negotiated cipher suites. In Phase-0 of EAP-FAST, the PAC-Opaque is sent to the supplicant outside the TLS session. Hence The PAC key field in the PAC-Opaque is AES key wrapped with the PAC KEK.

Key Storage

The module does not provide persistent storage for the keys used by the algorithms.

Key Zeroization

The Keys exist only in volatile memory for the duration of their use and the relevant keys are automatically zeroized after use. All buffers containing key material are zeroized after use before deleting or freeing them.

Table 4 shows the cryptographic keys and CSPs used by the module in the approved mode of operation.

Table 4 Cryptographic Keys and CSPs

Name	Description	Algorithm	Storage	Generation
RSA Private Key	Used to authenticate to the supplicant during the TLS handshake. Also used to decrypt the premaster secret during the TLS handshake.	1024-4096 bit RSA Key	RAM	Outside the module
RSA Public Key	Used during TLS handshake.	1024-4096 bit RSA Key	RAM	Outside the module
Diffie-Hellman Key pair	Used for DH based key exchange during TLS. **Note** Only authenticated DH key exchange is supported.	2048 bit DH Key	RAM	Generated inside the Module using ANSI X9.31 RNG
EAP-TLS master secret	Shared secret created using asymmetric Cryptography from which new session keys can be created.	Shared secret	RAM	Negotiated during the TLS handshake. Zeroized when the session is closed.
TLS Master Secret for EAP-FAST	Shared secret created using asymmetric Cryptography from which new session keys can be created. This secret is created from the EAP-FAST PAC key which is used as the pre-master secret.	Shared secret	RAM	Negotiated during the TLS handshake. Zeroized when the session is closed.
TLS Encryption Key	128-bit AES key used to encrypt session data. Zeroized at the end of the TLS session.	AES-128 bit	RAM	Negotiated during the TLS handshake. Zeroized when the session is closed.
TLS Integrity Key	HMAC-SHA-1 key used for integrity protection. Zeroized at the end of the TLS session.	HMAC-SHA-1	RAM	Negotiated during the TLS handshake. Zeroized when the session is closed.

Table 4 Cryptographic Keys and CSPs (continued)

Name	Description	Algorithm	Storage	Generation
EAP-TLS Master Session Key	This session key is independently derived by both the server and the supplicant at the end of the EAP-TLS handshake. This key is later sent to the authenticator (using AES key wrap) and may be used as the 802.11i PMK to establish other 802.11i keys.	Shared secret	RAM	Derived from TLS master secret during EAP-TLS handshake. Zeroized when the session is closed.
EAP-FAST Master Key	This key is used to derive the EAP-FAST Key encryption key.	ANSI X9.31 RNG	RAM	Generated inside the module.
EAP-FAST KEK	This Key encryption key is used to key wrap the PAC-Key within the PAC-Opaque.	AES-128 bit	RAM	Derived from EAP-FAST master key.
EAP-FAST PAC Key	A shared secret used to authenticate the client to the server. This key is used as the TLS premaster secret and is used to derive the TLS master secret for EAP-FAST.	Shared secret	RAM	Generated inside the module using ANSI X9.31 RNG.
EAP-FAST Session Key Seed	SKS is used to bind the TLS tunnel in EAP-FAST to inner EAP methods. SKS is used to generate the EAP-FAST Intermediate Compound Keys and the Master session key.	32-byte shared secret	RAM	This key is derived from the TLS master secret for EAP-FAST during the handshake. Zeroized when the session is closed.
EAP-FAST Intermediate Compound Keys (IMCK)	IMCK is used to verify the integrity of the TLS tunnel after each successful inner authentication and in the generation of EAP-FAST Master Session Key (MSK).	Shared secret	RAM	This key is derived from the EAP-FAST session key seed and inner EAP methods during the EAP-FAST handshake. Zeroized when the session is closed.
EAP-FAST Master Session Key	This session key is independently derived by both the server and the supplicant at the end of the EAP-FAST handshake. This key is later sent to the authenticator (using AES Key Wrap) and may be used as the 802.11i PMK to establish other 802.11i keys.	Shared secret	RAM	Derived from the EAP-FAST session key seed and IMCK during EAP-FAST handshake. Zeroized when the session is closed.
AES key wrap KEK (Key Encryption Key)	Used for encrypting the session key (aka PMK) for secure delivery to the authenticator using AES Key wrap.	AES-128 bit	RAM	Generated outside the module.
MACK (Message Authentication Code Key)	Used for keyed HMAC calculation over the RADIUS message, to ensure authenticity of the RADIUS messages.	HMAC-SHA-1	RAM	Generated outside the module.

Table 4 Cryptographic Keys and CSPs (continued)

Name	Description	Algorithm	Storage	Generation
Seed	This is the seed for the PRNG.	ANSI X9.31	RAM	Generated by passing SHA-1 function over data received from the program stack, the current time and a running number.
Seed Key	This is the seed key for the PRNG.	ANSI X9.31	RAM	Generated by passing SHA-1 function over data received from the program stack, the current time and a running number.
Random Nonce	Used to provide randomness to the MAC and connection between RADIUS request and response messages. Also provides randomness to the session key calculations.	ANSI X9.31 RNG	RAM	Generated outside the module.
PEAP Tunnel Key (TK)	Generated from the TLS master secret. This key is used to derive the intermediate compound session keys.	Shared secret	RAM	Negotiated during the PEAP handshake.
PEAP Master Session Key	This session key is independently derived by both the server and the supplicant at the end of the PEAP handshake. This key is later sent to the authenticator (using AES key wrap) and may be used as the 802.11i PMK to establish other 802.11i keys.	Shared secret	RAM	Derived during the EAP-FAST handshake.

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

The Software Integrity check is performed when the Crypto Module is initialized. The module implements an integrity test for the module software by verifying its 1536-bit RSA signature. The software integrity test passes if and only if the signature verifies successfully using the RSA public key.

Cryptographic Algorithm Self-Tests

The module performs the following Self-Tests at Power-on (see Table 5):

Table 5 Known Answer Tests

Algorithm	Test Procedure
AES Key Wrap	KAT
AES	KAT
RSA	Sign/Verify, Encrypt/Decrypt KAT
RNG	KAT
SHA-1	KAT
HMAC	KAT

Conditional Self Tests

Table 6 lists the conditional self tests performed by the module.

Table 6 Conditional Self-Tests

Algorithm	Test Procedure
Approved RNG	Continuous test.
Non-approved RNG	Continuous test.

Security Requirements

Secure Installation

The cryptographic module is the Cisco Secure ACS FIPS Module, version 1.1, which is installed on the Windows 2000 or Windows 2003 operational environment of the Cisco ACS Server Application.

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

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.
- Calling the function CryptoLibrary::Init() will place the module in FIPS mode of operations.
- In the FIPS mode, the logging level should be set to FULL or LOW.
- When negotiating TLS cipher suites, only FIPS approved algorithms must be specified.

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 module is coded in C.

Mitigation of Other Attacks

The module is not designed to mitigate any other attacks.