Tripwire Cryptographic Module FIPS 140-2 Security Policy
Document Revision: 2010.06.20_V1.4
S.W. Versions: 1.1 and 1.2

This non-proprietary document may only be reproduced in its entirety without modification.

Revision History		
Revision	Author	Description
2010.06.20_V1.4	Benjamin Jansen	Initial public release

Table of Contents

Introduction

The Tripwire Cryptographic Module (S/W Versions: 1.1 and 1.2) is a software only multi-chip standalone cryptographic module designed to provide FIPS validated cryptographic functionality for Tripwire, Inc. products. It implements the interfaces for encrypting sensitive data and to facilitate secure TLS communication channels.

The cryptographic module was tested on the following operational environment (in single user mode):

- Windows Server 2003 (32-bit)
- Sun Microsystems Java Runtime Environment Version 1.5

As per FIPS 140-2 Implementation Guidance G.5, the cryptographic module will remain compliant with the FIPS 140-2 validation when operating on any general purpose computer (GPC) provided that the GPC uses the specified single user operating system, or another compatible single user operating system such as any of the following:

- Microsoft Windows
- RedHat Enterprise Linux
- SUSE Linux
- Solaris
- IBM AIX
- HP-UX
- IBM i5/OS
- IBM z/Linux

Security Levels

The Tripwire Cryptographic Module is validated according to the following FIPS 140-2 defined levels.

Overall	**Security Level 1**
Area 1 - Cryptographic Module Specification	Security Level 1
Area 2 - Cryptographic Module Ports and Interfaces	Security Level 1
Area 3 - Roles, Services, and Authentication	Security Level 1
Area 4 - Finite State Model	Security Level 1
Area 5 - Physical Security	Security Level 1
Area 6 - Operational Environment	Security Level 1
Area 7 - Cryptographic Key Management	Security Level 1
Area 8 - EMI/EMC	Security Level 3
Area 9 - Self-Tests	Security Level 1
Area 10 - Design Assurance	Security Level 3
Area 11 - Mitigation of Other Attacks	Security Level 1

Cryptographic Boundary

The following diagram defines the cryptographic boundary:

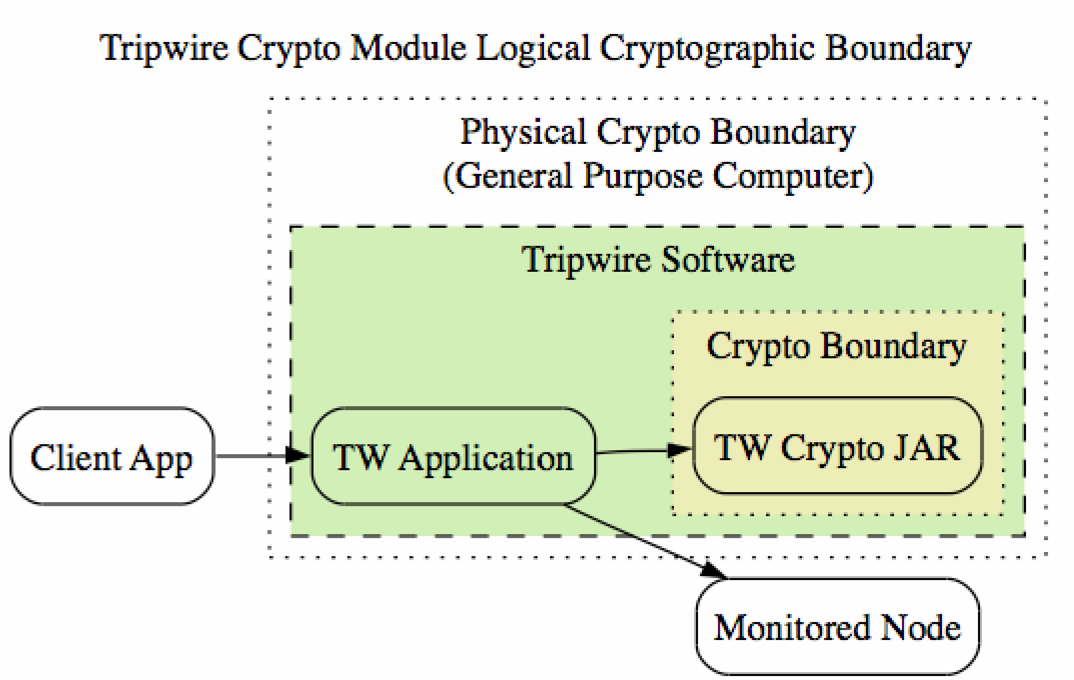

Exhibit 1 – *Specification of Cryptographic Boundary*

The logical cryptographic boundary is defined to include the following software components:

- tw-crypto-fips.jar

Provided Algorithms

FIPS Approved Algorithms

The Tripwire Cryptographic Module provides validated implementations of the following FIPS-approved algorithms:

- AES (CBC and ECB modes, 128/192/256 bit key sizes), certificate #1159
- RSA, certificate #548
    - GenKey 9.31
        - Supported public exponent value: 17
        - Supported modulus sizes: 1024, 1536, 2048, 3072, 4096
    - SigGen PKCS 1.5 and SigVer PKCS 1.5
        - Supported modulus sizes: 1024, 1536, 2048, 3072, 4096

    - Supported algorithms: SHA-1, SHA-224, SHA-256, SHA-384, SHA-512
- RNG (ANSI X9.31 with AES-256), certificate #641
- HMAC with SHA-1 (32-byte key size), certificate #660
- SHA-1, SHA-224, SHA-256, SHA-384, SHA-512, certificate #1072
- DSA (with 1024 bit key and SHA-1), certificate #376

Non-Approved Algorithms

The Tripwire Cryptographic Module also provides implementations of the following non-approved algorithms:

- MD5 (within TLS PRF)
- HMAC-MD5 (within TLS PRF)
- RSA (key wrapping; key establishment methodology provides between 80 and 128 bits of encryption strength; non-compliant less than 80-bits of encryption strength)

Physical Ports and Logical Interfaces

The logical interface for the Tripwire Cryptographic Module is defined by its API. While the physical ports of a GPC (keyboard, mouse, etc.) provide a means to interact with the module, the logical interface is defined by the module's API.

Logical Port	Logical Interface
Method parameters	Data Input
API Method calls	Control Input
Data returned by API method calls	Data Output
Status codes returned by, and exceptions thrown by, API method calls	Status Output

Exhibit 2 – *Specification of Cryptographic Module Physical Ports and Logical Interfaces*

Security rules

The following specifies the security rules under which the cryptographic module shall operate.

Self Tests

The cryptographic module shall support the following self-tests.

Power-up tests

- RSA signature generation/verification KAT

- DSA signature generation/verification KAT
- AES 256 CBC mode KAT
- DRNG (ANSI X9.31 with AES-256) KAT
- HMAC-SHA-1 KAT
- SHA-1 KAT
- SHA-224 KAT
- SHA-256 KAT
- SHA-384 KAT
- SHA-512 KAT
- Software integrity test – DSA signature verification
- Critical functions tests:
    - MD5 KAT (within TLS PRF)
    - HMAC-MD5 KAT (within TLS PRF)
    - RSA key wrap/unwrap KAT (within TLS)

Conditional tests

- Manual key entry = N/A
- Software load test = N/A
- Bypass test = N/A
- Pairwise consistency tests for RSA (key wrap/unwrap; sign/verify)
- Pairwise consistency tests for DSA (sign/verify)
- DRNG (ANSI X9.31 with AES-256) continuous test

Other Rules

- By policy, the cryptographic module is in a FIPS Approved mode of operation.
- The module only supports the following cipher suites for TLS:
    - TLS_RSA_WITH_AES_256_CBC_SHA
    - TLS_RSA_WITH_AES_128_CBC_SHA
- The cryptographic module provides all of the services listed in Exhibit 5 in FIPS mode (the Approved mode of operation) and also in non-FIPS mode. However, in non-FIPS mode the cryptographic module may not be relied upon to cryptographically protect sensitive unclassified data. The cryptographic module is put into a non-FIPS mode of operation whenever any of the following conditions have been performed:
    - The user uses an instance of the DSA algorithm with a 512-bit key, 768-bit key or hash with SHA-224, SHA-256, SHA-384, SHA-512 via any of the following services:
        - KeyPairGenerator.generateKeyPair
        - Signature.initSign
        - Signature.sign
        - Signature.initVerify
        - Signature.verify
    - The user uses an instance of the RSA algorithm with a 512-bit key or 768-bit key via any of the following services:

    - KeyPairGenerator.generateKeyPair
    - Signature.initSign
    - Signature.sign
    - Signature.initVerify
    - Signature.verify
    - SSLContext.init
    - SSLSocket.startHandshake *(client)*
    - SSLSocket.startHandshake *(server)*
  - The user requests generation of an RSA key using a public exponent of 3 or 65,537 via the KeyPairGenerator.generateKeyPair service.
  - The user does not supply at least 128-bits of entropy to seed the approved deterministic random number generator via the FipsProvider.addEntropy service.
- If the user ever puts the module into non-FIPS mode he is in violation of the security policy and shall zeroize all CSPs.
- The cryptographic module enforces logical separation for data input, control input, data output, status output interfaces via the API.
- All data output is inhibited during self tests, error states, and zeroization.
- During error states the module provides no cryptographic services, inhibits all data outputs, and provides error status via return codes and exceptions from the API.
- The cryptographic module protects CSPs from unauthorized disclosure, unauthorized modification, and unauthorized substitution. The cryptographic module protects public keys from unauthorized modification, and unauthorized substitution. The module does not perform any persistent storage of keys or CSPs.
- The module does not support manual key entry.
- The cryptographic module automatically performs self-tests without requiring any inputs or actions by the operator.
- Upon successful completion of the power-up self-tests the module provides status code “1” from the status output interface.
- When the cryptographic module enters the error state, the module provides status code “-5” from the status output interface.
- The module does not support bypass modes.
- The module does not support split-knowledge processes.
- The maintenance role and maintenance interface are not applicable.
- Specific components within the cryptographic boundary have not been excluded from the requirements of FIPS 140-2.
- The user must provide a minimum of 128-bits of entropy for each invocation of the FipsProvider.addEntropy service.
- Only the TLS protocol is supported (i.e., SSL is not supported).

Identification and Authentication Policy

- **Cryptographic Officer**: the role fulfilled by the person who performs on-demand self-tests and status querying.
- **User**: the role fulfilled by the external application that performs general security services.

The role is implicitly assumed based upon the service method being invoked.

Role	Type of Authentication	Authentication Data
Cryptographic Officer	N/A	N/A
User	N/A	N/A

Exhibit 3 - *Roles and Required Identification and Authentication (FIPS 140-2 Table C1)*

Authentication Mechanism	Strength of Mechanism
N/A	N/A

Exhibit 4 - *Strengths of Authentication Mechanisms (FIPS 140-2 Table C2)*

Access Control Policy

Available Services

Following is a list of services supported by the cryptographic module.
Note: The use of the term "SSL" in service names is solely to provide compatibility with the existing API. Only TLS is supported (i.e. SSL is not supported).

Service Name	Service Description
FipsProvider.getStatusCode	Returns a status code representing the state of the module (starting, selftested, shutdown, error, etc.)
FipsProvider.addEntropy	Reseeds the cryptographic module's DRNG. The given seed supplements the existing seed; thus, repeated use is guaranteed never to reduce randomness.
FipsProvider.runSelfTest	Runs the FIPS mandated power-up self-tests
Zeroizable.zeroize	When called on any object representing CSP material, will wipe the sensitive data from memory.
Cipher.getInstance	Creates a cryptographic Cipher object for the specified algorithm.

Service Name	Service Description
Cipher.init	Initialize the Cipher object for use by specifying the mode (encryption or decryption) and key.
Cipher.update	Continue a multi-part cryptographic operation, inputing data and returning output.
Cipher.doFinal	Finish a multi-part cryptographic operation, inputing data and returning output.
MessageDigest.getInstance	Creates a cryptographic MessageDigest object for the specified algorithm.
MessageDigest.update	Update the state of the digesting object with more input data.
MessageDigest.digest	Calculate and return the final hash value of the input data.
Mac.getInstance	Creates a cryptographic MAC object for the specified algorithm.
Mac.init	Initialize the Mac object for use by specifying key.
Mac.update	Update the state of the MAC object with more input data.
Mac.doFinal	Calculate and return the final MAC value of the input data.
KeyPairGenerator.getInstance	Creates a cryptographic KeyPairGenerator object for the specified algorithm.
KeyPairGenerator.initialize	Initializes the key pair generator using the specified parameters (e.g. key size).
KeyPairGenerator.generateKeyPair	Generate and return a new asymmetric keypair.
Signature.getInstance	Creates a cryptographic Signature object for the specified algorithm.
Signature.initSign	Initialize this object for signing by providing the private key to use for signing.
Signature.sign	Calculate the signature bytes of all the data updated.
Signature.update	Updates the data to be signed or verified.
Signature.initVerify	Initializes this object for verification, using the provided public key.
Signature.verify	Verifies the passed-in signature.
SecureRandom.getInstance	Creates a cryptographic SecureRandom object for the specified algorithm.
SecureRandom.setSeed	To enter additional entropy into the object state.
SecureRandom.nextBytes	To request a series of random bytes from the DRNG.
SSLContext.getInstance	Creates a cryptographic SSLContext object for the specified algorithm.
SSLContext.init	Initializes the SSLContext object for use.

Service Name	Service Description
SSLContext.getSocketFactory	Returns a SSLSocketFactory object for this context.
SSLContext.getServerSocketFactory	Returns a SSLServerSocketFactory object for this context.
SSLSocketFactory.createSocket	Create a SSLSocket to be used for TLS communication.
SSLServerSocketFactory.createSocket	Create a SSLServerSocket to be used for TLS communication.
SSLSocket.startHandshake *(client)*	Perform TLS client handshake.
SSLSocket.startHandshake *(server)*	Perform TLS server handshake.

Exhibit 5 – *Cryptographic Module Services*

Service Roles and Key/CSP Access

Each service is defined as accessing certain CSPs in certain ways. The types of access are:

Access Name	Description
Destroy	Actively overwrite.
Enter	The item is input into the cryptographic boundary.
Output	The item is output from the cryptographic boundary.
Generate	The item is generated using the Approved DRNG.
Encrypt	The item is used to encrypt data.
MAC	The item is used to generate a MAC.
Sign	The item is used to sign data.
Verify	The item is used to verify the signature of signed data.
Establish	The item is established as part of the TLS protocol.
Random	The item is used to generate pseudo-random bits using the Approved DRNG.
Wrap	The item is used for RSA key wrap within TLS.
Unwrap	The item is used for RSA key unwrap within TLS.

Exhibit 6 – *Types of CSP Access*

Following is a listing of roles, services, cryptographic keys and CSPs, and types of access to the cryptographic keys and CSPs that are available to each of the authorized roles via the corresponding services.

Role CO	Role U	Service	Cryptographic Keys, CSPs, & Type(s) of Access
X		FipsProvider.runSelfTest	n/a
X		FipsProvider.getStatusCode	n/a
	X	FipsProvider.addEntropy	Seed: enter
	X	Zeroizable.zeroize	All: destroy
	X	Cipher.getInstance	n/a
	X	Cipher.init	AES Key: enter
	X	Cipher.update	AES Key: encrypt, decrypt
	X	Cipher.doFinal	AES Key: encrypt, decrypt
	X	MessageDigest.getInstance	n/a
	X	MessageDigest.update	n/a
	X	MessageDigest.digest	n/a
	X	Mac.getInstance	n/a
	X	Mac.init	HMAC Key: enter
	X	Mac.update	HMAC Key: MAC
	X	Mac.doFinal	HMAC Key: MAC
	X	KeyPairGenerator.getInstance	n/a
	X	KeyPairGenerator.initialize	DRNG state: enter
	X	KeyPairGenerator.generateKeyPair	• DSA Keypair: generate, output • RSA Keypair: generate, output • DRNG state: random
	X	Signature.getInstance	n/a
	X	Signature.initSign	• DSA private key: enter • RSA private key: enter • DRNG state: enter
	X	Signature.sign	• DSA private key: sign • RSA private key: sign • DRNG state: random
	X	Signature.update	n/a
	X	Signature.initVerify	• DSA public key: enter • RSA public key: enter
	X	Signature.verify	• DSA public key: verify • RSA public key: verify

Role		Service	Cryptographic Keys, CSPs, & Type(s) of Access
CO	U		
	X	SecureRandom.getInstance	n/a
	X	SecureRandom.setSeed	• Seed: enter
	X	SecureRandom.nextBytes	• Random bytes: generate, output
	X	SSLContext.getInstance	n/a
	X	SSLContext.init	• CA certificate: enter • Local certificate: enter • Local private key: enter • DRNG state: enter
	X	SSLContext.getSocketFactory	n/a
	X	SSLContext.getServerSocketFactory	n/a
	X	SSLSocketFactory.createSocket	DRNG state: random
	X	SSLServerSocketFactory.createSocket	DRNG state: random
	X	SSLSocket.startHandshake *(client)*	• TLS Pre-master secret: generate, output • TLS Master secret: establish • TLS PRF state: establish • TLS AES session keys: establish • TLS HMAC session keys: establish • DRNG State: random • CA certificate: verify • Local certificate: output • Local private key: sign • Remote certificate: enter, wrap
	X	SSLSocket.startHandshake *(server)*	• TLS Pre-master secret: enter • TLS Master secret: establish • TLS PRF state: establish • TLS AES session keys: establish • TLS HMAC session keys: establish • DRNG State: random • CA certificate: verify signature • Local certificate: output • Local private key: unwrap • Remote certificate: enter, verify signature

Exhibit 7 – *Services Authorized for Roles, Access Rights within Services (FIPS 140-2 Table C3, Table C4)*

Physical Security Policy

The physical security requirements are not applicable to the software only cryptographic module.

Physical Security Mechanisms	Recommended Frequency of Inspection/Test	Inspection/Test Guidance Details
N/A	N/A	N/A

Exhibit 8 - *Inspection/Testing of Physical Security Mechanisms (FIPS 140-2 Table C5)*

Mitigation of Other Attacks Policy

The Tripwire Cryptographic Module does not provide for mitigation of other attacks.

Other Attacks	Mitigation Mechanism	Specific Limitations
N/A	N/A	N/A

Exhibit 9 - *Mitigation of Other Attacks (FIPS 140-2 Table C6)*

Glossary

Term	Definition
AES	Advanced Encryption Standard
ANSI	American National Standards Institute
API	Application Programming Interface
CBC	Cipher-Block Chaining, a block cipher mode of operation
Cipher	Cryptographic algorithm used for encryption/decryption
CSP	Critical Security Parameter
DRNG	Deterministic Random Number Generator
DSA	Digital Signature Algorithm
ECB	Electronic Codebook, a block cipher mode of operation
FIPS	Federal Information Processing Standards
FIPS 140-2	FIPS requirements for cryptographic modules
GPC	General Purpose Computer

HMAC	keyed-Hash Message Authentication Code
KAT	Known Answer Test
MAC	Message Authentication Code
MD5	Message-Digest algorithm 5
N/A	Not Applicable
RNG	Random Number Generator
RSA	An algorithm for public-key cryptography
SHA	Secure Hash Algorithm
TLS	Transport Layer Security
TLS PRF	Transport Layer Security Pseudo-Random Function
TW	Tripwire
TW Application	A Tripwire application which uses the Module for cryptographic operations