Tripwire Cryptographic Module FIPS 140-2 Security Policy

Document Revision: 2013.10.05_V1.0
Software Version: 2.0

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

Revision History		
Revision	Author	Description
2013.10.05_V1.0	Brian Cox	Initial public release.

Table of Contents

Introduction

The Tripwire Cryptographic Module (Software Version: 2.0) is a software only multi-chip standalone cryptographic module designed to provide FIPS validated cryptographic functionality for Tripwire, Inc. products. The cryptographic module implements the interfaces for encrypting sensitive data and to facilitate secure TLS communication channels.

The cryptographic module was tested on the following operational environment on general purpose computer (GPC) platforms:

- Java SE Runtime Environment (build 1.6.0_33-b05) [JavaHotSpot 64-bit Server VM (build 20.8-b03 mixed mode)] on Windows 2008 Server R2 with SP1 (64-bit) running on a Dell Optiplex 960 (single-user mode)

- Java SE Runtime Environment (build 1.6.0_33-b05) [JavaHotSpot 64-bit Server VM (build 20.8-b03 mixed mode)] on Windows 2008 Server R2 with SP1 (64-bit) running on a Dell Optiplex 9010 (single-user mode)

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
- Mac OS X

For the avoidance of doubt, it is hereby stated that the CMVP makes no statement as to the correct operation of the module or the security strengths of the generated keys when so ported if the specific operational environment is not listed on the validation certificate.

Security Levels

The Tripwire Cryptographic Module is validated according to the following FIPS 140-2 Security Levels.

FIPS 140-2 Security Area	**Security Level**
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	Not applicable
Area 6: Operational Environment	Security Level 1
Area 7: Cryptographic Key Management	Security Level 1
Area 8: EMI/EMC	Security Level 1
Area 9: Self-Tests	Security Level 1
Area 10: Design Assurance	Security Level 3
Area 11: Mitigation of Other Attacks	Not applicable

Cryptographic Boundary

The following diagram defines the cryptographic boundary:

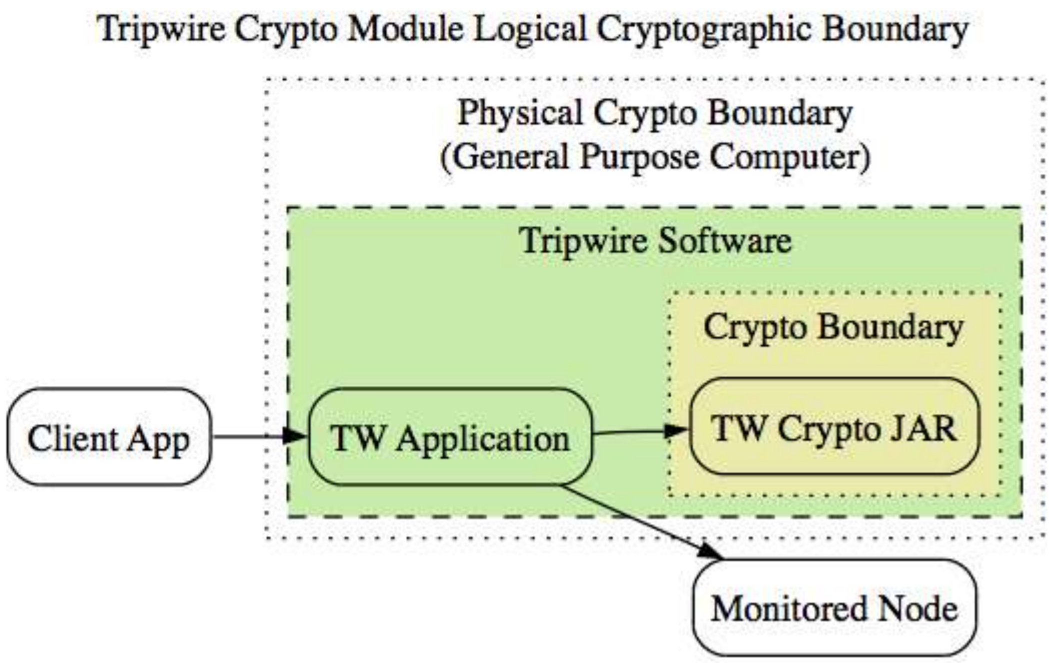

Exhibit 1 – *Specification of Cryptographic Boundary*

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-bit/192-bit/256-bit key sizes), certificate #2719
- RSA, certificate #1414 (NOTICE: This module is impacted by SP800-131A. See Historical RSA List Val #1414. RSA 1024-bit and 1536-bit keys are not allowed in FIPS mode. Any use of key sizes of this length is explicitly disallowed in FIPS mode, constitutes an explicit violation of this Security Policy and deems the module as non-compliant. SHA-1 shall not be used for digital signature generation with the exception as specified in SP 800-52 REV1; any other use of SHA-1 for digital signature generation is explicitly disallowed in FIPS mode, constitutes an explicit violation of this Security Policy and deems the module as non-compliant.)
  - GenKey 9.31
    - Supported public exponent value: 17
    - Supported modulus sizes: 2048, 3072, 4096
  - SigGen PKCS 1.5 and SigVer PKCS 1.5
    - Supported modulus sizes: 2048, 3072, 4096
    - Supported algorithms: SHA-1 (only with the exception as specified in SP 800-52 REV1), SHA-224, SHA-256, SHA-384, SHA-512
- RNG (ANSI X9.31 with AES-256), certificate #1260
- HMAC with SHA-1 (32-byte key size), certificate #1698
- SHA-1, SHA-224, SHA-256, SHA-384, SHA-512, certificate #2284
- DSA, certificate #835
- SP800-135 TLS KDF, CVL certificate #176

Users should reference the transition tables that will be available at the CMVP Web site (http://csrc.nist.gov/groups/STM/cmvp/). The data in the tables will inform users of the risks associated with using a particular algorithm and a given key length.

Non-Approved Algorithms

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

- MD5, HMAC-MD5 and RSA encrypt/decrypt*: Used for TLS key establishment in FIPS mode per FIPS 140-2 IG D.2.
- RSA (key wrapping; key establishment methodology provides 112-bits of encryption strength).

  *NOTICE: RSA encrypt/decrypt is used in FIPS mode for key wrap/unwrap only as a non-Approved but allowed commercially available key establishment method; usage of RSA encrypt/decrypt as a public key cipher for other data is explicitly disallowed in FIPS mode, constitutes an explicit violation of this Security Policy and deems the module as non-compliant.

  *NOTICE: The TLS PRF uses HMAC-MD5 internally (CVL certificate #176); MD5 and HMAC-MD5 “are not” otherwise available.

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 cryptographic module, the logical interface is defined by the cryptographic 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 KAT
- RSA signature verification KAT
- DSA signature generation KAT
- DSA signature verification KAT
- AES 256 CBC mode encrypt KAT
- AES 256 CBC mode decrypt KAT
- RNG (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 – RSA 2048 with SHA-256 signature verification
- SP800-135 KDF KAT
- Critical functions tests:
  - MD5 KAT (within TLS PRF)
  - HMAC-MD5 KAT (within TLS PRF)

  - RSA encrypt/decrypt KAT (within TLS)

Conditional tests

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

Other Rules

- By default, the cryptographic module is in a FIPS Approved mode of operation.
- Upon successful completion of the power-up self-tests the module provides status code "1" (READY) from the status output interface.
- The cryptographic module outputs error status: “-5” (GENERALERROR) from the status output interface in the event of a power-up self-test or conditional self-test failure
- The cryptographic module only supports the following cipher suites for TLS:
  - TLS_RSA_WITH_AES_256_CBC_SHA
  - TLS_RSA_WITH_AES_128_CBC_SHA
- NOTICE: The cryptographic module is in a non-FIPS mode of operation whenever any of the following conditions have been satisfied, each of which is explicitly disallowed in FIPS mode, constitutes an explicit violation of this Security Policy and deems the module as non-compliant:
  - The user uses an instance of the DSA algorithm with a non-2048-bit key or hash other than SHA-256 via any of the following:
    - Signature.initSign
    - Signature.initVerify
    - SSLContext.init
  - The user uses an instance of the RSA algorithm with a key whose size is less than 2048-bits (i.e. 1024-bit or 1536-bit) via any of the following:
    - Signature.initSign
    - Signature.initVerify
    - SSLContext.init
  - The user requests generation of an RSA key using a public exponent other than 17 via any of the following:
    - KeyPairGenerator.initialize
    - KeyPairGenerator.generateKeyPair
  - The user does not supply at least 128-bits of entropy to seed the approved RNG via any of the following:
    - FipsProvider.addEntropy
    - KeyPairGenerator.initialize
    - SecureRandom.setSeed
    - SSLContext.init

  - The user provides a cryptographic key that has an effective strength that is less than 112-bits of equivalent computational resistance to attack via any of the following:
    - Cipher.init
    - Mac.init
    - Signature.initSign
    - Signature.initVerify
    - SSLContext.init
  - The user uses RSA encrypt/decrypt for a purpose other than key wrap/unwrap, via any of the following:
    - SSLSocket.startHandshake *(client)*
    - SSLSocket.startHandshake *(server)*
  - The user uses RSA SHA-1 for digital signature generation, other than the exception as specified in SP800-52 REV1 via the following:
    - Signature.initSign
- If the user ever puts the cryptographic module into non-FIPS mode he is in violation of this 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 cryptographic 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 cryptographic module does not perform any persistent storage of keys or CSPs.
- The cryptographic 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 cryptographic module provides status code "1" from the status output interface.
- The cryptographic module does not support bypass modes.
- The cryptographic 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 cryptographic module is not a radio and does not support OTAR.
- The user must provide a minimum of 128-bits of entropy for each invocation of the following:
  - FipsProvider.addEntropy
  - KeyPairGenerator.initialize
  - SecureRandom.setSeed
  - SSLContext.init
- Only the TLS protocol is supported (i.e., SSL "IS NOT" supported).

- The module generates cryptographic keys whose strengths are modified by available entropy. No assurance of the minimum strength of generated keys.
- RSA 1024-bit and 1536-bit keys are not allowed in FIPS mode. Any use of key sizes of this length is explicitly disallowed in FIPS mode, constitutes an explicit violation of this Security Policy and deems the module as non-compliant.

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.runSelfTest	Runs the FIPS mandated power-up self-tests.
FipsProvider.getStatusCode	Returns a status code representing the state of the cryptographic module (starting, selftested, shutdown, error, etc.).

Service Name	**Service Description**
FipsProvider.addEntropy	Reseeds the cryptographic module RNG. The given seed supplements the existing seed; thus, repeated use is guaranteed never to reduce randomness.
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.
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, inputting data and returning output.
Cipher.doFinal	Finish a multi-part cryptographic operation, inputting 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.

Service Name	**Service Description**
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 RNG.
SSLContext.getInstance	Creates a cryptographic SSLContext object for the specified algorithm.
SSLContext.init	Initializes the SSLContext object for use.
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.createServerSocket	Create a SSLServerSocket to be used for TLS communication.
SSLSocket.startHandshake *(client)* *NOTICE: The TLS PRF uses HMAC-MD5 internally (CVL certificate #176); MD5 and HMAC-MD5 "are not" otherwise available.*	Perform TLS client handshake.
SSLSocket.startHandshake *(server)* *NOTICE: The TLS PRF uses HMAC-MD5 internally (CVL certificate #176); MD5 and HMAC-MD5 "are not" otherwise available.*	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.

Access Name	Description
output	The item is output from the cryptographic boundary.
generate	The item is generated using the Approved RNG.
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 RNG.

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. “CO” and “U” refer to Cryptographic Officer and User, respectively.

Role		**Service**	**Cryptographic Keys, CSPs, & Type(s) of Access**
CO	**U**		
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

Role		Service	Cryptographic Keys, CSPs, & Type(s) of Access
CO	U		
	X	KeyPairGenerator.getInstance	N/A
	X	KeyPairGenerator.initialize	RNG state: enter
	X	KeyPairGenerator.generateKeyPair	• DSA Keypair: generate, output • RSA Keypair: generate, output • RNG state: random
	X	Signature.getInstance	N/A
	X	Signature.initSign	• DSA private key: enter • RSA private key: enter • RNG state: enter
	X	Signature.sign	• DSA private key: sign • RSA private key: sign • RNG 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
	X	SecureRandom.getInstance	N/A
	X	SecureRandom.setSeed	• Seed: enter
	X	SecureRandom.nextBytes	• Random bytes: generate, output
	X	SSLContext.getInstance	N/A

Role		**Service**	**Cryptographic Keys, CSPs, & Type(s) of Access**
CO	**U**		
	X	SSLContext.init	• CA certificate: enter • Local certificate: enter • Local private key: enter • RNG state: enter
	X	SSLContext.getSocketFactory	N/A
	X	SSLContext.getServerSocketFactory	N/A
	X	SSLSocketFactory.createSocket	RNG state: random
	X	SSLServerSocketFactory.createServerSocket	RNG state: random
	X	SSLSocket.startHandshake *(client)* *NOTICE: The TLS PRF uses HMAC-MD5 internally (CVL certificate #176); MD5 and HMAC-MD5 "are not" otherwise available.*	• TLS Pre-master secret: generate, output • TLS Master secret: establish • TLS PRF state: establish • TLS AES session keys: establish • TLS HMAC session keys: establish • RNG State: random • CA certificate: verify • Local certificate: output • Local private key: sign • Remote certificate: enter, encrypt

Role		Service	Cryptographic Keys, CSPs, & Type(s) of Access
CO	U		
	X	SSLSocket.startHandshake *(server)* *NOTICE: The TLS PRF uses HMAC-MD5 internally (CVL certificate #176); MD5 and HMAC-MD5 "are not" otherwise available.*	• TLS Pre-master secret: enter • TLS Master secret: establish • TLS PRF state: establish • TLS AES session keys: establish • TLS HMAC session keys: establish • RNG State: random • CA certificate: verify signature • Local certificate: output • Local private key: decrypt • 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
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
IG	FIPS 140-2 Implementation Guidance
KAT	Known Answer Test
KDF	Key Derivation Function
MAC	Message Authentication Code
MD5	Message-Digest algorithm 5 *NOTICE: The TLS PRF uses HMAC-MD5 internally (CVL certificate #176); MD5 and HMAC-MD5 "are not" otherwise available.*
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 (also referred to as SP800-135 TLS KDF within this document) *NOTICE: The TLS PRF uses HMAC-MD5 internally (CVL certificate #176); MD5 and HMAC-*

	MD5 “are not” otherwise available.
TW	Tripwire
TW Application	A Tripwire application which uses the Module for cryptographic operations