Tanium Cryptographic Module v1.0 FIPS 140-2 Non-Proprietary Security Policy

Document Revision 0.6

08/10/2016

Prepared for:

Tanium, Inc.

2200 Powell Street, 6th Floor, Emeryville, CA 94608

Prepared By:

www.gossamersec.com

Revision History

Revision	Date	Authors	Summary
0.1	4/28/2014	Khai Van	Initial draft
0.2	4/15/2016	Khai Van	Updated
0.3	4/20/2016	Khai Van	Updated based on comments
0.4	6/10/2016	Khai Van	Updated based on CMVP comments
0.5	8/1/2016	Khai Van	Updated OE to state vendor affirmed platforms
0.6	8/10/2016	Khai Van	Added ECDH keys back into SP

Table of Contents

1. Introduction 4
2. Tanium Cryptographic Module 5
2.1 Module Specification 5
2.1.1 Security Level 5
2.1.2 FIPS Mode of Operation 6
2.1.3 Approved Cryptographic Algorithms 6
2.1.4 Non-Approved Cryptographic Algorithms Allowed In FIPS Mode 6
2.1.5 Non-Approved Cryptographic Algorithms Not Allowed in FIPS Mode 7
2.2 Module Interfaces 7
2.3 Roles, Services and Authentication 8
2.4 Finite State Model 15
2.5 Physical Security 15
2.6 Operational Environment 15
2.7 Key Management 16
2.8 Electromagnetic Interference and Compatibility 17
2.9 Self-Tests 17
2.9.1 Power-up Self-tests: 17
2.9.2 Conditional Self-tests 17
2.10 Guidance and Secure Operation 18
2.10.1 Crypto-office Guidance 18
2.10.2 User Guidance 18
2.11 Mitigation of Other Attacks 18

1. Introduction

This non-proprietary FIPS 140-2 security policy for the Tanium Cryptographic Module details the secure operation of the Tanium Cryptographic Module as required in Federal Information Processing Standards Publication 140-2 (FIPS 140-2) as published by the National Institute of Standards and Technology (NIST) of the United State Department of Commerce. This document, the Cryptographic Module Security Policy (CMSP), also referred to as the Security Policy, specifies the security rules under which the Tanium Cryptographic Module must operate.

The Tanium Cryptographic Module underpins Tanium's security management platform. Tanium's platform is a security and configuration management solution that provides instant visibility and allows enterprises to collect data and update machines in any-sized network, in seconds. Tanium's platform is able to query information from hundreds of thousands of machines in seconds because of its intelligent peer-to-peer communication model. This speed means that information is current and accurate when assessing a security threat or vulnerability. Its next-generation architecture also allows enterprises to fully deploy a real-time environment in a matter of days and run a single Tanium server to support hundreds of thousands of endpoints.

2. Tanium Cryptographic Module

2.1 Module Specification

The Tanium Cryptographic Module (Version v1.0) (hereinafter referred to as the "Library", "cryptographic module" or the "module") is a software only cryptographic module composed of an interface DLL containing cryptographic functionality executing on a general-purpose computer system running Microsoft Windows. The module is named TaniumCryptoLibrary.dll and the signature file is named TaniumCryptoLibrary.dll.sig.

The physical perimeter of the general-purpose computer (GPC) comprises the module's physical cryptographic boundary, while the Tanium Cryptographic Module DLL constitutes the module's logical cryptographic boundary.

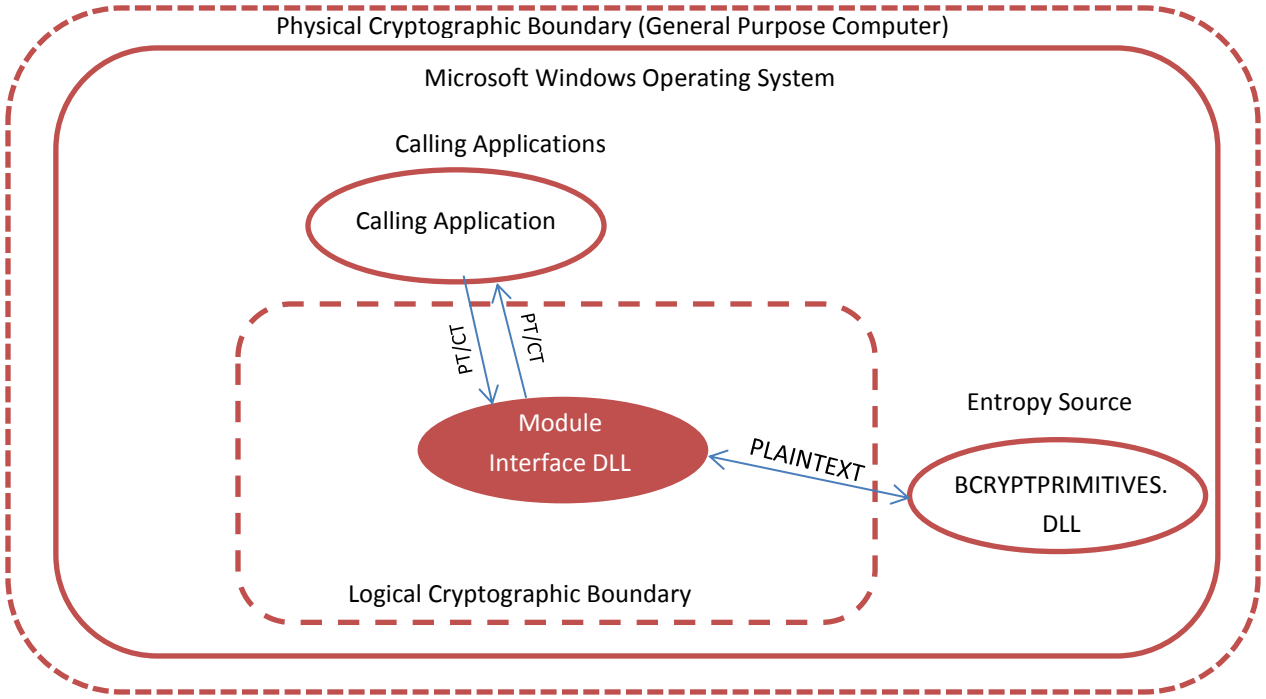

Figure 1 - Logical Diagram

*PT = Plaintext, CT = Ciphertext

2.1.1 Security Level

The Tanium Cryptographic Module meets the overall requirements applicable to Level 1 security overall of FIPS 140-2 and the below specified section security levels.

Table 1 - Module Security Level Specification

#	FIPS 140-2 Section	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	N/A
6	Operational Environment	1
7	Cryptographic Key Management	1
8	EMI/EMC	3
9	Self-tests	1
10	Design Assurance	3
11	Mitigation of Other Attacks	N/A
	Overall Level	1

2.1.2 FIPS Mode of Operation

The Tanium Cryptographic Module utilizes only FIPS-Approved algorithms to ensure FIPS compliant operation. As a result, the operator need not adhere to any rules to utilize the module in a FIPS compliant fashion.

2.1.3 Approved Cryptographic Algorithms

The module uses cryptographic algorithm implementations that have received the following certificate numbers from the Cryptographic Algorithm Validation Program.

Table 2 – FIPS-Approved Algorithm Certificates

Algorithm	Modes and Key/Curve Sizes	CAVP Certificate when operating on Microsoft Windows 32-bit and 64-bit
FIPS 186-4 ECDSA	PKV, SigGen, SigVer; P-521	#836
SP 800-90A DRBG	AES-256 CTR	#1105
Secure Hash Standard (SHS)	SHA-1/224/256/384/512	#3197
HMAC	SHA-1/224/256/384/512	#2519
AES	ECB/CBC/GCM; 128/256 bits	#3876
SP 800-135	TLS 1.0/1.1/1.2 KDF	#745
ECC CDH Primitive	EC-Diffie-Hellman; P-521	#744

NOTE: The TLS protocol has not been reviewed or tested by the CAVP and CMVP.

2.1.4 Non-Approved Cryptographic Algorithms Allowed In FIPS Mode

NDRNG

NOTE: The NDRNG is classified as the module's entropy source for its Approved DRBG. It is a non-Approved algorithm allowed for use in FIPS mode.

Additionally, the module obtains entropy from the 140-2 validated BCRYPTPRIMITIVES provider in its Windows operational environment. As described in the BCRYPTPRIMITIVES Security Policy, the BCRYPTPRIMITIVES.DLL module generates cryptographic keys whose strengths are modified by available entropy, and thus, the Tanium Cryptographic Module also must generates cryptographic random numbers and signatures (as the Tanium Cryptographic Module does not generate any keys) whose strengths are modified by available entropy.

The following maps the BCRYPTPRIMITIVES versions to the respective platforms used to test the Tanium Cryptographic Module

Windows 7 (32-bit) 140-2 Validation certificate #1329:
http://csrc.nist.gov/groups/STM/cmvp/documents/140-1/140val-historical.htm#1329

Windows 7 (64-bit) 140-2 Validation certificate #1329:
http://csrc.nist.gov/groups/STM/cmvp/documents/140-1/140val-historical.htm#1329

Windows Server 2008 R2 140-2 Validation certificate #1336:
http://csrc.nist.gov/groups/STM/cmvp/documents/140-1/140val-historical.htm#1336

Windows Server 2012 140-2 Validation certificate #1892:
http://csrc.nist.gov/groups/STM/cmvp/documents/140-1/1401val2013.htm#1892

2.1.5 Non-Approved Cryptographic Algorithms Not Allowed in FIPS Mode

The module does not use any non-FIPS 140-2 approved cryptographic algorithms. The module only utilizes FIPS-Approved cryptography.

2.2 Module Interfaces

The module is classified as a multiple-chip standalone module for FIPS 140-2 purposes. As such, the module's physical cryptographic boundary encompasses the general-purpose computer running the Microsoft Windows operating system and interfacing with the computer peripherals (USB devices [keyboard and mouse], video devices [monitors, screens, camera], optical drives, audio devices [speakers, headset, and microphone], network devices [Ethernet and Wireless adapters], and power adapter).

However, the module provides only a logical interface via an Application Programming Interface (API) and does not interface or communication with or across any of the physical ports of the GPC. This logical interface exposes service that operators (calling applications) may use directly.

The API interface provided by the module is mapped onto the four FIPS 140-2 logical interfaces: data input, data output, control input, and status output. It is through this logical API that the module logically separates them into distinct and separate interfaces. The mapping of the module's API to the four FIPS 140-2 interfaces is as follows.

- Data input – input arguments to all constructors and functions specifying input parameters
- Data output – modified input arguments (those passed by reference) and return values for all constructors and methods modifying input arguments and returning values
- Control input – invocation of all methods
- Status output – information returned by the GetStatus and GetVersionString methods and any exceptions thrown by constructors and methods.

2.3 Roles, Services and Authentication

The module supports both of the FIPS 140-2 required roles, the Crypto-officer and the User role, and supports no additional roles. An operator implicitly selects the Crypto-officer role when loading (or causing loading of) the library and selects the User role when soliciting services from the module through its API. The module requires no operator authentication, and the below table enumerates the module's services.

Table 3 - Service Descriptions for Crypto-officer and User Roles

Service	Type[1]	Description
Crypto-Officer services		
Library Loading	N/A	The process of loading the shared library
User services		
TaniumCryptoLibraryCryptosystem		
NewTaniumCryptoLibraryCryptosystem	F	Creates a new crypto system instance
CheckCryptosystemCorrectness	F	Runs integrity checks for the module
DefaultHashName	F	Returns the default hash name
GenerateWord32	F	Returns a 32-bit random bitstring
GetVersionString	F	Initializes a new instance of the Class
GetStatus	F	Returns FIPS state of the module
HashCreator		
HashCreator	C	Returns a new object instance
~HashCreator	D	Destructs the object
MaximumTokenSize	F	Maximum Size of SHA Hashing instance
CreateToken	F	Create hash token from message and range
UpdateToken	F	Update hash token of message with range
GetFinalToken	F	Return final hash token of message
HashVerifier		
HashVerifier	C	Returns a new object instance
Verify	F	Take message and hash and verify

[1] (C)onstructor, (D)estructor, or Member (F)unction

Service	Type[1]	Description
PrivateKey		
~PrivateKey	D	Destroys an object instance
WritePKCS8	F	Create PKCS8 object of Private Key
Clone	F	Return clone of private key object
CreatePublicKey	F	Derive public key from Private Key
CreateSigner	F	Instantiate signing object from Private Key
CreateMatchingHashCreator	F	Instantiate HashCreator matching the key type
CreateMatchingHashVerifier	F	Instantiate HashVerifier matching the key type
PublicKey		
~PublicKey	D	Destroys an object instance
WriteX509	F	Create X509 object of Public Key
Clone	F	Return clone of Public Key Object
CreateVerifier	F	Instantiate Verifier object with Public Key
CreateMatchingHashCreator	F	Instantiate HashCreator matching the key type
CreateMatchingHashVerifier	F	Instantiate HashVerifier matching the key type
AuthenticationCreator		
~AuthenticationCreator	D	Destroys an instance of the object
MaximumTokenSize	F	Returns the maximum size of a token
CreateToken	F	Abstract base class
CreateToken	F	Creates a digesttoken based on an input message
UpdateToken	F	Updates digest in the token
GetFinalToken	F	Returns a constructed tokendigest
AuthenticationVerifier		
~AuthenticationVerifier	D	Destroys an instance of the object
Verify	F	Matches message based on input
Verify	F	Matches message based on AuthenticationToken
SignatureVerifier		
SignatureVerifier	C	Creates an object based on public key and hash name
Verify	F	Verify signature matches message
Signer		
Signer	C	Creates new object instance
MaximumTokenSize	F	Return Maximimum Signature length for Signer
CreateToken	F	Create signature from message
UpdateToken	F	Update signature
GetFinalToken	F	Finalize signature
HashCache		
HashCache	C	Creates a new object instance
HashCache	C	Creates a new object with number of entries and size
swap	F	Replaces data in cache with another
clear	F	Dissasociates data and memory location
resize	F	Resizes buffer
size	F	Returns counts or size of cache
EntrySize	F	Returns size of each cache entry
Locate	F	Returns location of given constant
Match	F	Attempts to match location with value
Put	F	Inserts value into location supplied
CachingSignatureVerifier		
CachingSignatureVerifier	C	Creates a new object instance
swap	F	Replaces data in cache with another
SetCacheSize	F	Resizes the cache
Verify	F	Verifies the signature
SSLContext		
SSLContext	C	Returns a new object with input flags

Service	Type[1]	Description
SSLContext	C	Creates a new object based on context input
~SSLContext	D	Destroys an object instance
CreateSSLConnection	F	Returns a new SSL connection
Reset	F	Flushes connection details
SSLConnection		
SSLConnection	C	Returns a new object instance
~SSLConnection	D	Destroys an object instance
Connect	F	Connect as client. Also populates handshake data
SubmitIncomingEncodedData	F	Receives encoded data and handshake sent from peer
ReadIncomingClearData	F	Reads app-level decoded data
SubmitOutgoingClearData	F	Receives data to be encoded and sent to peer
ReadOutgoingEncodedData	F	Reads encoded data to be sent
HasOutgoingEncodedData	F	Returns true if outgoing data buffer is not empty
HasPendingData	F	Returns true if object's buffers are not empty
ClearPendingData	F	Dissasociates buffer data and memory location
Shutdown	F	Flushes object's buffers and shuts down function
GetPeerCertificate	F	Collects Peer's certificate if connected
CryptoSystem		
Cryptosystem	C	Creates a new object instance
~Cryptosystem	D	Destroys an object instance
GenerateWord32	F	Generates a 32 bit integer
DefaultHashName	F	Returns the string "SHA-512"
GetVersionString	F	Returns OpenSSL version
GetStatus	F	Returns self-test status of CryptoSystem object
CreateHashCreator	F	Creates a HashCreator based on hash name
CreateHashCreator	F	Creates a HashCreator based on default hash name
CreateHashVerifier	F	Creates a new HashVerifier based on hash name
ReadPrivateKeyInPKCS8	F	Creates PKCS8 PrivateKey
ReadPublicKeyInX509	F	Creates x509 PublicKey
ReadCertificateX509	F	Creates x509 Certificate
CreateClientSSLContext	F	Creates new SSL context as a client
CreateServerSSLContext	F	Creates new SSL context as a server
CertificateExtension		
CertificateExtension	C	Creates an object instanceAdds extension value to certificate object
CertificateExtension	C	Adds extension value to certificate object
~CertificateExtension	D	Destroys an object instance
GetNID	F	Returns ID in Certificate object
GetName	F	Returns name in Certificate object
GetNameSize	F	Returns size of name in Certificate object
GetValue	F	Returns value in Certificate object
GetValueSize	F	Returns size of value in Certificate object
Certificate		
Certificate	C	Creates a new x509 object
Certificate	C	Creates an x509 object from an x509 pointer
~Certificate	D	Destroys an object instance
GetPEM	F	Returns PEM from Certificate object
GetSubject	F	Returns subject from Certificate object
GetIssuer	F	Returns issuer from Certificate object
GetNotAfterDateStr	F	Returns not after date from Certificate object
GetNotBeforeDateStr	F	Returns not before date from Certificate object
GetSerialNumber	F	Returns serial nmber from Certificate object
GetFingerprint	F	Returns fingerprint from Certificate object

Service	Type[1]	Description
GetVersion	F	Returns version from Certificate object
GetExtensionCount	F	Returns number of extensions from Certificate object
GetExtension	F	Returns extension name from Certificate object
SSLReadWriteError		
SSLReadWriteError	C	Adds error code to a string
what	F	Returns string
~SSLReadWriteError	D	Destroys an object instance

Table 4 - Service Inputs and Outputs

Service	Data Input	Data Output	CSP	Access[2]	Status Out
Crypto-Officer services					
Library Loading	N/A	N/A	N/A	N/A	Flag
User services					
TaniumCryptoLibraryCryptosystem					
DefaultHashName	N/A	Hash name	N/A	N/A	Exception
GenerateWord32	N/A	Random number	N/A	N/A	Exception
GetVersionString	N/A	Version	N/A	N/A	Exception
NewTaniumCryptoLibraryCryptosystem	Public Key, FIPS flag, & crypto object	CryptoSystem	ECDSA public key	X	Exception
CheckCryptosystemCorrectness	N/A	N/A	ECDSA public key	X	Exception
GetStatus	N/A	N/A	N/A	N/A	Exception
HashCreator					
HashCreator	Hash name	Instance ref	N/A	N/A	Exception
~HashCreator	N/A	N/A	N/A	N/A	Exception
MaximumTokenSize	N/A	Size of token	N/A	N/A	Exception
CreateToken	Buffer & msg	Resultant hash (single step)	N/A	N/A	Exception
UpdateToken	Msg	Intermediate hash (multi-stage)	N/A	N/A	Exception
GetFinalToken	Buffer	Resultant hash (final-stage)	N/A	N/A	Exception
HashVerifier					
HashVerifier	Hash name	Instance ref	N/A	N/A	Exception
Verify	Digest & msg	Result	N/A	N/A	Exception
PrivateKey					
~PrivateKey	N/A	N/A	N/A	N/A	Exception
WritePKCS8	Stream	PKCS8 private key	ECDSA private key	W	Exception
Clone	N/A	Instance ref	ECDSA private key	W	Exception
CreatePublicKey	N/A	Instance ref	ECDSA private key	W	Exception
CreateSigner	N/A	Instance ref	ECDSA private key	W	Exception

[2] (G)enerate, (R)ead, (W)rite, e(X)excute, (Z)eroize

Service	Data Input	Data Output	CSP	Access[2]	Status Out
CreateMatchingHashCreator	N/A	Instance ref	N/A	N/A	Exception
CreateMatchingHashVerifier	N/A	Instance ref	N/A	N/A	Exception
PublicKey					
~PublicKey	N/A	N/A	N/A	N/A	Exception
WriteX509	Stream pointer	X.509 public key	ECDSA public key	R	Exception
Clone	N/A	Instance ref	ECDSA public key	W	Exception
CreateVerifier	N/A	Instance ref	ECDSA public key	W	Exception
CreateMatchingHashCreator	N/A	Instance ref	ECDSA public key	W	Exception
CreateMatchingHashVerifier	N/A	Instance ref	ECDSA public key	W	Exception
AuthenticationCreator					
CreateToken	Message	Token	N/A	N/A	Exception
~AuthenticationCreator	N/A	N/A	N/A	N/A	Exception
MaximumTokenSize	N/A	size_t	N/A	N/A	Exception
CreateToken	Message and SplitOctetBuffer	Octet range	N/A	N/A	Exception
UpdateToken	Message	N/A	N/A	N/A	Exception
GetFinalToken	Octet buffer	N/A	N/A	N/A	Exception
AuthenticationVerifier					
~AuthenticationVerifier	N/A	N/A	N/A	N/A	Exception
Verify	Token range & msg	Result	N/A	N/A	Exception
Verify	Token & msg	Result	N/A	N/A	Exception
SignatureVerifier					
SignatureVerifier	Public Key & hash name	Instance ref	ECDSA public key	X	Exception
Verify	Signature & msg	Result	ECDSA public key	X	Exception
Signer					
Signer	PrivateKey instance & hash name	Instance ref	ECDSA private key	W	Exception
MaximumTokenSize	N/A	Max private key length	ECDSA private key	X	Exception
CreateToken	Buffer & Msg	Instance ref	ECDSA private key	X	Exception
UpdateToken	Msg	Exception	N/A	N/A	Exception
GetFinalToken	Buffer	Exception	N/A	N/A	Exception
HashCache					
HashCache	N/A	N/A	N/A	N/A	Exception
HashCache	Number of entries & size of entries	Instance ref	N/A	N/A	Exception
swap	HashCache object	N/A	N/A	N/A	Exception
clear	N/A	N/A	N/A	N/A	Exception
resize	Number of entries & size of entries	Instance ref	N/A	N/A	Exception
size	N/A	Integer	N/A	N/A	Exception

Service	Data Input	Data Output	CSP	Access[2]	Status Out
EntrySize	N/A	Integer	N/A	N/A	Exception
Locate	Octet range	Integer	N/A	N/A	Exception
Match	Integer & Octet range	Result	N/A	N/A	Exception
Put	Integer & Octet range	Instance ref	N/A	N/A	Exception
CachingSignatureVerifier					
CachingSignatureVerifier	N/A	N/A	N/A	N/A	Exception
swap	Object instance	N/A	N/A	N/A	Exception
SetCacheSize	New hash cache size	Instance ref	N/A	N/A	Exception
Verify	Signature & msg	Result	N/A	N/A	Exception
SSLContext					
SSLContext	cert path, key path, cipher order flag, ciphersuite string, peer CA file string, client flag	Instance ref	N/A	N/A	Exception
SSLContext	SSLContext object	Instance ref	N/A	N/A	Exception
~SSLContext	N/A	N/A	N/A	N/A	Exception
CreateSSLConnection	N/A	N/A	N/A	N/A	Exception
Reset	N/A	N/A	N/A	N/A	Exception
SSLConnection					
Connect	N/A	Result	N/A	N/A	Exception
SSLConnection	SSLContext pointer & client flag	Instance ref	N/A	N/A	Exception
~SSLConnection	N/A	N/A	N/A	N/A	Exception
SubmitIncomingEncodedData	Data	N/A	N/A	N/A	Exception
ReadIncomingClearData	Buffer	Buffer contents	N/A	N/A	Exception
SubmitOutgoingClearData	Data	N/A	N/A	N/A	Exception
ReadOutgoingEncodedData	Buffer	Buffer contents	N/A	N/A	Exception
HasOutgoingEncodedData	N/A	Result	N/A	N/A	Exception
HasPendingData	N/A	Result	N/A	N/A	Exception
ClearPendingData	N/A	N/A	N/A	N/A	Exception
Shutdown	N/A	N/A	N/A	N/A	Exception
GetPeerCertificate	N/A	certificate pointer	N/A	N/A	Exception
CryptoSystem					
Cryptosystem	N/A	Instance ref	N/A	N/A	Exception
~Cryptosystem	N/A	N/A	N/A	N/A	Exception
GenerateWord32	N/A	Random number	N/A	N/A	Exception
DefaultHashName	N/A	String	N/A	N/A	Exception
GetVersionString	N/A	String	N/A	N/A	Exception
GetStatus	N/A	Result	N/A	N/A	Exception
CreateHashCreator	N/A	Instance ref	N/A	N/A	Exception
CreateHashCreator	Hash name	Instance ref	N/A	N/A	Exception
CreateHashVerifier	Hash name	Instance ref	N/A	N/A	Exception
ReadPrivateKeyInPKCS8	Stream	ECDSA private key	ECDSA private key	W	Exception

Service	Data Input	Data Output	CSP	Access[2]	Status Out
ReadPublicKeyInX509	Stream	ECDSA public key	ECDSA public key	W	Exception
ReadCertificateX509	Stream	x509 certificate	ECDSA public key	R	Exception
CreateClientSSLContext	cert path, key path, cipher order flag, ciphersuite string, peer CA file string, client flag	Instance ref	N/A	N/A	Exception
CreateClientSSLContext	cert path, key path, cipher order flag, ciphersuite string, peer CA file string	Instance ref	N/A	N/A	Exception
CreateClientSSLContext	cert path, key path, cipher order flag, ciphersuite string	Instance ref	N/A	N/A	Exception
CreateServerSSLContext	cert path, key path, cipher order flag, ciphersuite string, peer CA file string, client flag	Instance ref	N/A	N/A	Exception
CreateServerSSLContext	cert path, key path, cipher order flag, ciphersuite string, peer CA file string	Instance ref	N/A	N/A	Exception
CreateServerSSLContext	cert path, key path, cipher order flag, ciphersuite string	Instance ref	N/A	N/A	Exception
CertificateExtension					
CertificateExtension	N/A	Instance ref	N/A	N/A	Exception
CertificateExtension	name ID, name, name size, value, value size	Instance ref	N/A	N/A	Exception
~CertificateExtension	N/A	N/A	N/A	N/A	Exception
GetNID	N/A	Name ID	N/A	N/A	Exception
GetName	N/A	Name	N/A	N/A	Exception
GetNameSize	N/A	Name size	N/A	N/A	Exception
GetValue	N/A	Value	N/A	N/A	Exception
GetValueSize	N/A	Value size	N/A	N/A	Exception
Certificate					
Certificate	Stream	Instance ref	N/A	N/A	Exception
Certificate	x509 object pointer	Instance ref	N/A	N/A	Exception

Service	Data Input	Data Output	CSP	Access[2]	Status Out
~Certificate	N/A	N/A	N/A	N/A	Exception
GetPEM	N/A	PEM string	N/A	N/A	Exception
GetSubject	N/A	Subject	N/A	N/A	Exception
GetIssuer	N/A	Issuer	N/A	N/A	Exception
GetNotAfterDateStr	N/A	Not after date	N/A	N/A	Exception
GetNotBeforeDateStr	N/A	Not before date	N/A	N/A	Exception
GetSerialNumber	N/A	Serial number	N/A	N/A	Exception
GetFingerprint	N/A	Fingerprint	N/A	N/A	Exception
GetVersion	N/A	Version	N/A	N/A	Exception
GetExtensionCount	N/A	integer	N/A	N/A	Exception
GetExtension	Index integer	Value at index	N/A	N/A	Exception
SSLReadWriteError					
SSLReadWriteError	String & error code	N/A	N/A	N/A	Exception
what	N/A	String	N/A	N/A	Exception
~SSLReadWriteError	N/A	N/A	N/A	N/A	Exception

2.4 Finite State Model

The module has a Finite State Model (FSM) that descrbes the module's behavior and transitions based upon its current state and the command received. The module's FSM was reviewed as part of the overall FIPS 140-2 validation.

2.5 Physical Security

The physical security requirements does not apply to the module. The module is a software-only module that executes upon a general-purpose computer.

2.6 Operational Environment

The module executes on a general purpose operating system running in single user mode that segregates processes into separate process spaces. Thus, the operating system separates each process space from all others. The below table listed the specific Microsoft Windows operating systems upon which validation testing was performed and the associated FIPS 140-2 certificate number for the Microsoft Windows Cryptographic Primitives Library (BCRYPTPRIMITIVES) upon which the module relies on for entropy.

Table 5 - Validated Operational Environments

#	Operating System and Test Platform	140-2 Cert.#
1	Microsoft Windows 7 (32-bit) running on a Dell PowerEdge R430 (single-user mode)	1329
2	Microsoft Windows 7 (64-bit) running on a Dell PowerEdge R430 (single-user mode)	1329
3	Microsoft Windows Server 2008 R2 (64-bit) running on a Dell PowerEdge R430 (single-user mode)	1336

4	Microsoft Windows Server 2012 (64-bit) running on a Dell PowerEdge R430 (single-user mode)	1892

In addition, the module is also able to execute on the above same four operating systems running as virtual guest operating systems atop VMWare Server version 5.5 on a Dell PowerEdge R430. As no validation testing has been executed on these operational environments, they are considered vendor affirmed. The module follows porting rules under FIPS 140-2 Implementation Guidance G.5.

2.7 Key Management

The module possesses only one key, its self-integrity test ECDSA Public key. Beyond that key, the module does not store any other keys persistently, and it is the calling applications responsibility to appropriately manage keys. The module cannot generate keys but can accept keys entered by an operator, and affords an operator the ability to zeroize keys held in RAM. The following table describes the module's Security Relevant Data Items (SRDI's) including asymmetric and symmetric keys.

Table 6 - Module Keys

Key	Type	Size	Description	Origin	Stored	Zeroized
ECDSA public key	ECDSA	521 bits	Asymmetric keys used for signature verification.	Entered by calling application	RAM / plaintext	Zeroize context
ECDSA private key	ECDSA	521 bits	Asymmetric keys used for signature generation	Entered by calling application	RAM / plaintext	Zeroize context
DRBG Seed	Seed	384 bits	Entropy input	Entropy source	RAM / plaintext	Zeroize context
DRBG Key	AES-256	256 bits	Key value used in DRBG	Random data	RAM / plaintext	Zeroize context
DRBG V	Secret	128 bits	Secret internal state value	Random data	RAM / plaintext	Zeroize context
TLS Pre-Master secret	TLS Secret	384 bits	Secret value generated by the client to be sent via key exchange	DRBG output	RAM / plaintext	Zeroize context
TLS Master secret	TLS Secret	384 bits	Master secret derived from Pre-Master secret	Derived from Pre-Master secret	RAM / plaintext	Zeroize context
TLS Session keys	TLS Session key	128, 192.256 bits	Session encryption keys derived from Master secret	Derived from Master secret	RAM / plaintext	Zeroize context
HMAC keys	HMAC	14 to 256 bytes	Keys used for keyed hashing	Derived from TLS key exchange	RAM / plaintext	Zeroize context
AES keys	AES ECB, CBC, GCM	128, 256 bits	Keys used for encryption/decryption	Derived from TLS key exchange	RAM / plaintext	Zeroize context

Self-integrity Pub Key	ECDSA	521 bits	ECDSA Public key used by the module for it's power up integrity test	Compiled into the module	Module image	N/A (see 140-2 IG 7.4)
EC DH public key	EC Diffie-Hellman	521 bits	Public key used for key establishment	Entered by calling application	RAM / plaintext	Zeroize context
EC DH private key	EC Diffie-Hellman	521 bits	Private key used for key establishment	Entered by calling application	RAM / plaintext	Zeroize context

2.8 Electromagnetic Interference and Compatibility

The module meets level 3 security for FIPS 140-2 EMI/EMC requirements as the Tanium Cryptographic Module passed validation executing upon general-purpose computers that confirm to the EMI/EMC requirements specific by 47 Code of Federal Regulations, Part 15, Subpart B, Unintentional Radiators, Digital Devices, Class B (i.e., for home use).

2.9 Self-Tests

The module automatically performs a complete set of power-up self-tests during library load to ensure proper operation, thus an operator has no access to cryptographic functionality unless the power-up self-tests pass and the library load succeeds. The module performs the following self-tests:

2.9.1 Power-up Self-tests:

ECDSA P-521 sign/verify test
ECDH KAT
AES encrypt KAT
AES decrypt KAT
DRBG KAT
HMAC-SHA-1, HMAC-SHA-256, HMAC-512 KAT (covers SHA-1, SHA-256, and SHA-512 KATs)
Integrity self-test (ECDSA P-521 w/ SHA-256 signature verification)

2.9.2 Conditional Self-tests

DRBG Continuous Random Number Generator Test
SP 800-90A Section 11.3 DRBG Health Tests
- Instantiate
- Generate
- Reseed
- Uninstantiate

NDRNG Continuous Random Number Generator Test

Should the module fail a self-test, the module will return an error and inhibit all cryptographic operations. Finally, an operator may invoke the power-up self-tests at any time by power-cycling the GPC and then reloading module.

2.10 Guidance and Secure Operation

The Module meets overall Level 1 requirements for FIPS PUB 140-2. The sections below describe the Crypto-officer and User guidance.

2.10.1 Crypto-office Guidance

The Crypto-officer or operator responsible for configuring the operational environment upon which the module runs must ensure FIPS compliant operation (as described in section 2.1.2, FIPS Mode of Operation, of the Security Policy).

Additionally, the Crypto-officer is defined to be the operator responsible for loading the library, thus when invoked by a calling application (either at library load or dynamically), the operating system loader will load the module, causing it to automatically perform its power-up self-tests. Should the module fail its power-up self-tests, the module set a status indicator and inhibit its cryptographic functions.

2.10.2 User Guidance

Once the operating system has been properly configured by the Crypto-officer (if needed), the Tanium Cryptographic Module requires no special usage to operate in a FIPS-compliant manner. The module utilizes only FIPS-Approved cryptographic algorithms. The User must assume responsibility for managing all keys, as the module does not provide any persistent key storage. For AES-GCM, the operator must reset the IV to the last one used in case the module's power is lost and then restored.

2.11 Mitigation of Other Attacks

The Tanium Cryptographic Module does not claim to mitigate any attacks beyond the FIPS 140-2 Level 1 requirements for validation.