Red Hat Enterprise Linux GnuTLS Cryptographic Module

version 5.0

FIPS 140-2 Non-proprietary Security Policy

Document version 1.1

Last Update: 2017-09-01

Prepared by:
atsec information security corporation
9130 Jollyville Road, Suite 260
Austin, TX 78759
www.atsec.com

Table of Contents

1. Cryptographic Module Specification

This document is the non-proprietary security policy for the Red Hat Enterprise Linux GnuTLS Cryptographic Module v5.0, and was prepared as part of the requirements for conformance to Federal Information Processing Standard (FIPS) 140-2, Level 1 Software Module.

1.1. Description of the Module

The Red Hat Enterprise Linux GnuTLS Cryptographic Module (hereafter referred to as the "module") is a set of libraries implementing general purpose cryptographic algorithms and network protocols. The module supports the Transport Layer Security (TLS) Protocol defined in [RFC5246] and the Datagram Transport Layer Security (DTLS) Protocol defined in [RFC4347]. The module provides a C language Application Program Interface (API) for use by other calling applications that require cryptographic functionality or TLS/DTLS network protocols.

The components of the cryptographic module are specified in the following table:

Component	Description
libgnutls	This library provides the main interface which allows the calling applications to request cryptographic services. The Approved cryptographic algorithm implementations provided by this library include the TLS protocol, DRBG, RSA Key Generation, Diffie-Hellman and EC Diffie-Hellman.
libnettle	This library provides the cryptographic algorithm implementations, including AES, Triple-DES, SHA, HMAC, RSA Digital Signature, DSA and ECDSA.
libhogweed	This library includes the primitives used by libgnutls and libnettle to support the asymmetric cryptographic operations.
libgmp	This library provides the big number arithmetic operations to support the asymmetric cryptographic operations.
*.hmac	The .hmac files contain the HMAC-SHA-256 values of its associated library for integrity check during the power-up.

Table 1: Cryptographic Module Components

The module has been tested on the following multi-chip standalone platforms:

Manufacturer	Model	Test Configurations	Processor
Dell	PowerEdge R630	Red Hat Enterprise Linux 7.4 with/without AES-NI	Intel(R) Xeon(R) CPU E5-2640 v3

Table 2: Tested Platform

Note: Per FIPS 140-2 IG G.5, the CMVP makes no statement as to the correct operation of the module or the security strengths of the generated keys when this module is ported and executed in an operational environment not listed on the validation certificate.

The following table shows the security level for each of the eleven sections of the validation:

Security Component	FIPS 140-2 Security Level
Cryptographic Module Specification	1

Cryptographic Module Ports and Interfaces	1
Roles, Services and Authentication	1
Finite State Model	1
Physical Security	N/A
Operational Environment	1
Cryptographic Key Management	1
EMI/EMC	1
Self-Tests	1
Design Assurance	1
Mitigation of Other Attacks	1

Table 3: Security Level of the Module

1.2. Description of the Approved Modes

The module supports two modes of operation:

- In "FIPS mode" (the FIPS Approved mode of operation) only approved or allowed security functions with sufficient security strength can be used.
- In "non-FIPS mode" (the non-Approved mode of operation) only non-approved security functions can be used.

When the module is powered up, the module executes the power-up tests and obtains the HMAC value of the module for integrity check from the .hmac file for each software libraries within the module's logical boundary. The module enters FIPS mode automatically after power-up tests succeed. If the module fails any power-up tests, the module will return an error code and enter the error state to prohibit any further cryptographic operations. The operator should follow the guidance in section 9.3 for descriptions of possible self-test errors and recovery procedures.

Once the module completes power-up tests successfully and enters FIPS mode by default, the module is available to provide cryptographic services. The mode of operation is implicitly assumed depending on the security function invoked and the security strength of the cryptographic keys.

Critical security parameters used or stored in FIPS mode are not used in non-FIPS mode, and vice versa.

The module supports the following FIPS 140-2 Approved algorithms in FIPS mode:

Algorithm	CAVS Certificates	Standards	Keys/CSPs
AES with the following mode: • CBC • GCM	Certs. #4658, #4659, #4660, #4661, #4662 and #4663	FIPS 197 AES SP 800-38A SP 800-38D GCM	AES keys 128 bits, 192 bits and 256 bits (CBC) AES keys 128 bits and 256 bits (GCM)

Algorithm	CAVS Certificates	Standards	Keys/CSPs
3-key Triple-DES with the following mode: • CBC	Certs. #2479 and #2480	SP 800-67 SP 800-38A	Triple-DES keys 192 bits
DRBG using AES-256 CTR_DRBG where AES encryption is provided by the C implementation from the nettle library **Note:** CTR_DRBG without Derivation Function, without Prediction Resistance and no Reseeding implementation	Certs. #1574 and #1575 Dependent AES Certs. implemented by the nettle library: Certs. #4584, #4585	SP 800-90A	Entropy input string, seed, V and Key
SHA: • SHA-1 • SHA-224 • SHA-256 • SHA-384 • SHA-512	Certs. #3817, #3818, #3819 and #3820		
HMAC: • SHA-1 • SHA-224 • SHA-256 • SHA-384 • SHA-512	Certs. #3086 and #3087	FIPS 198-1	At least 112 bits HMAC Key
DSA Domain Parameters Generation and Verification • SHA-384	Certs. #1233 and #1234	FIPS 186-4	DSA keys: • L=2048, N=224 • L=2048, N=256 • L=3072, N=256
DSA Key Generation			
DSA Signature Generation			L=2048, N=224 (SHA-224, SHA-256, SHA-384, SHA-512) L=2048, N=256 (SHA-256, SHA-384, SHA-512) L=3072, N=256 (SHA-256, SHA-384, SHA-512)

Algorithm	CAVS Certificates	Standards	Keys/CSPs
DSA Signature Verification • SHA-1 • SHA-224 • SHA-256 • SHA-384 • SHA-512			L=1024, N=160 (SHA-1, SHA-224, SHA-256, SHA-384, SHA_512) L=2048, N=224 (SHA-224, SHA-256, SHA-384, SHA-512) L=2048, N=256 (SHA-256, SHA-384, SHA-512) L=3072, N=256 (SHA-256, SHA-384, SHA-512)
RSA Key Generation • SHA-384	Certs. #2542 and #2543	FIPS 186-4	RSA keys 2048 and 3072 bits
RSA (PKCS#1 v1.5) Signature Generation			RSA keys 2048 bits: • SHA-224 • SHA-512 RSA keys 3072 bits: • SHA-1 • SHA-224 • SHA-256 • SHA-384 • SHA-512
RSA (PKCS#1 v1.5) Signature Verification			RSA keys 1024 bits: • SHA-1 • SHA-224 • SHA-256 • SHA-384 • SHA-512 RSA keys 2048 bits: • SHA-1 • SHA-224 • SHA-256 • SHA-384 • SHA-512 RSA keys 3072 bits: • SHA-1 • SHA-224 • SHA-256 • SHA-384 • SHA-512

Algorithm	CAVS Certificates	Standards	Keys/CSPs
ECDSA Key Pair Generation and Public Key Verification	Certs. #1146 and #1147	FIPS 186-4	ECDSA keys based on P-256, P-384, or P-521 curve
ECDSA Signature Generation • SHA-224 • SHA-256 • SHA-384 • SHA-512			
ECDSA Signature Verification • SHA-1 • SHA-224 • SHA-256 • SHA-384 • SHA-512			
CVL (KAS FCC/ECC)	Certs. #1307 and #1309	SP 8000-56A	
Key Derivation Function in TLS v1.0, v1.1 and v1.2 with: • SHA-256 • SHA-384 • SHA-512	CVL Certs. #1308 and #1310	SP800-135 rev1 Section 4.2	None

Table 4: Validated Cryptographic Algorithms

Note: The TLS and DTLS network protocols have not been reviewed or tested by the CAVP and CMVP.

Note2: 1024 bit RSA and DSA signature verification is legacy-use.

The module supports different AES and SHA implementations based on the underlying platform's capability. The module supports the use of AES-NI and SSSE3 when it is operated in an Intel® x86-64 architecture environment. When the AES-NI is enabled in the operating environment, the module performs the AES operations using the support from the AES-NI instructions; when the AES-NI is disabled in the operating environment, the module performs the AES operations using the supports from the Supplemental Streaming SIMD Extensions 3 (SSSE3). The module also performs SHA operations using the supports from the SSSE3. The SSSE3 cannot be disabled on the test platform that runs in the Intel® x86 architecture environment. The AES and SHA implementations that uses the AES-NI and SSSE3 supports and their related algorithms have been CAVS tested and functionally tested. Although the module implements different implementations for AES and SHA, only one implementation for one algorithm will ever be available for AES SHA and HMAC cryptographic services at run-time.

The module implements the following non-Approved algorithms which are allowed in FIPS mode:

- RSA Key Wrapping with encryption and decryption primitives and at least 2048 bits modulus size
- FIPS 186-4 RSA Key Generation and Signature Generation with at least 2048 bits modulus

size except 2048 and 3072 bits[1]

- FIPS 186-4 RSA Signature Verification with at least 1024 bits modulus size except 1024, 2048 and 3072 bits[1]
- FIPS 186-4 RSA Digital Signature with MD5 or SHA-1[2]
- Diffie-Hellman KAS with SHA-224 and SHA-256 and 2048 bits domain parameters size (CVL Certs. #1307 and #1309)
- Diffie-Hellman KAS with greater than 2048 bits domain parameters size (CVL Certs. #1307 and #1309)
- EC Diffie-Hellman KAS with SHA-256, SHA-384 and SHA-512, and keys associated with the NIST curves P-256, P-384, P-521 (CVL Certs. #1307 and #1309)

The module implements the following non-Approved algorithms only available in non-FIPS mode:

- AES with counter mode (CTR)
- Blowfish
- Camellia
- CAST 128
- DES
- Diffie-Hellman KAS with smaller than 2048 bits domain parameters size
- FIPS 186-2 RSA Key Generation
- FIPS 186-4 RSA Key Generation, Signature Generation and Key Wrapping with modulus size smaller than 2048 bits and/or non-Approved hashing
- FIPS 186-4 RSA Signature Verification with smaller than 1024 bits modulus size
- FIPS 186-4 DSA Signature Generation with smaller than 2048 bits public key size
- GOST Hash R 34.11-94 (RFC4357)
- Lagged Fibonacci Pseudo-randomness Generator
- MD2
- MD4
- MD5
- NDRNG
- PBKDFv2 (RFC2898)
- RC2
- RC4
- RIPEMD-160
- Salsa20
- Serpent
- SHA-3
- Twofish
- UMAC
- Yarrow RNG

Regarding the available services in FIPS mode of operation and non-FIPS mode of operation, please

1 These algorithms are CAVS tested and listed as Approved algorithms in this module.
2 Please note that MD5 and SHA-1 can be used in the digital signature when it is used in TLS protocol according to the [SP800-52].

refer to Table 6: Services Available in FIPS mode and Table 7. Services Available in non-FIPS mode in section 3.2 Services.

1.3. Cryptographic Boundary

The module's physical boundary is the physical boundary of the test platform. The embodiment type of the module is defined as multi-chip standalone.

The module's logical boundary is the shared library files and their integrity check HMAC files, which are delivered through Red Hat Package Manager (RPM) listed in section 9.1. The binary files and the HMAC files within the module's logical boundary are listed below:

- libgnutls library:
    - /usr/lib{64,}/libgnutls.so.28.43.0
    - /usr/lib{64,}/.libgnutls.so.28.43.0.hmac
- libnettle library:
    - /usr/lib{64,}/libnettle.so.4.7
    - /usr/lib{64,}/.libnettle.so.4.7.hmac
- libhogweed library:
    - /usr/lib{64,}/libhogweed.so.2.5
    - /usr/lib{64,}/.libhogweed.so.2.5.hmac
- libgmp library:
    - /usr/lib{64,}/libgmp.so.10.2.0
    - /usr/lib{64,}/fipscheck/libgmp.so.10.2.0.hmac

1.3.1. Hardware Block Diagram

The physical boundary of the module is the physical boundary of the test platform which is a General Purpose Computer (GPC). The following block diagram shows the hardware components of a GPC:

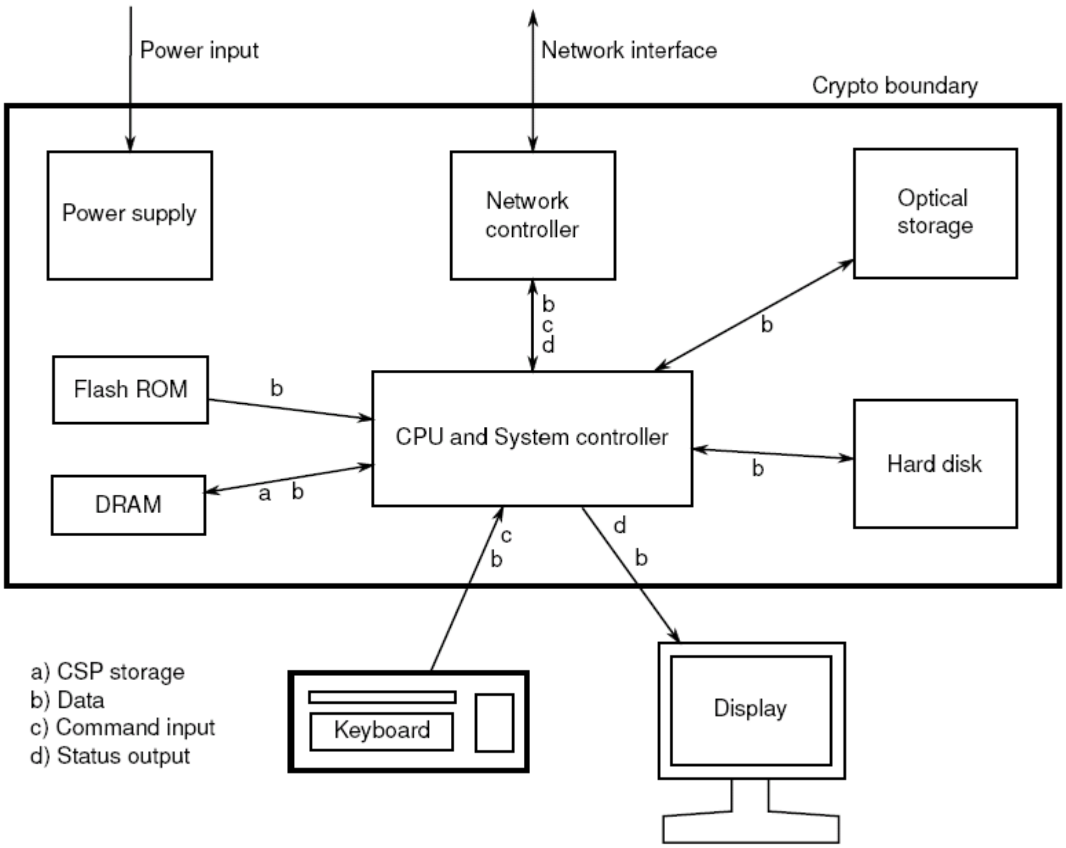

Figure 1. Hardware Block Diagram

1.3.2. Software Block Diagram

The block diagrams below shows the module's logical boundary, its interface with the operational environment and the delimitation of its logical boundary which are included in **BLUE** box:

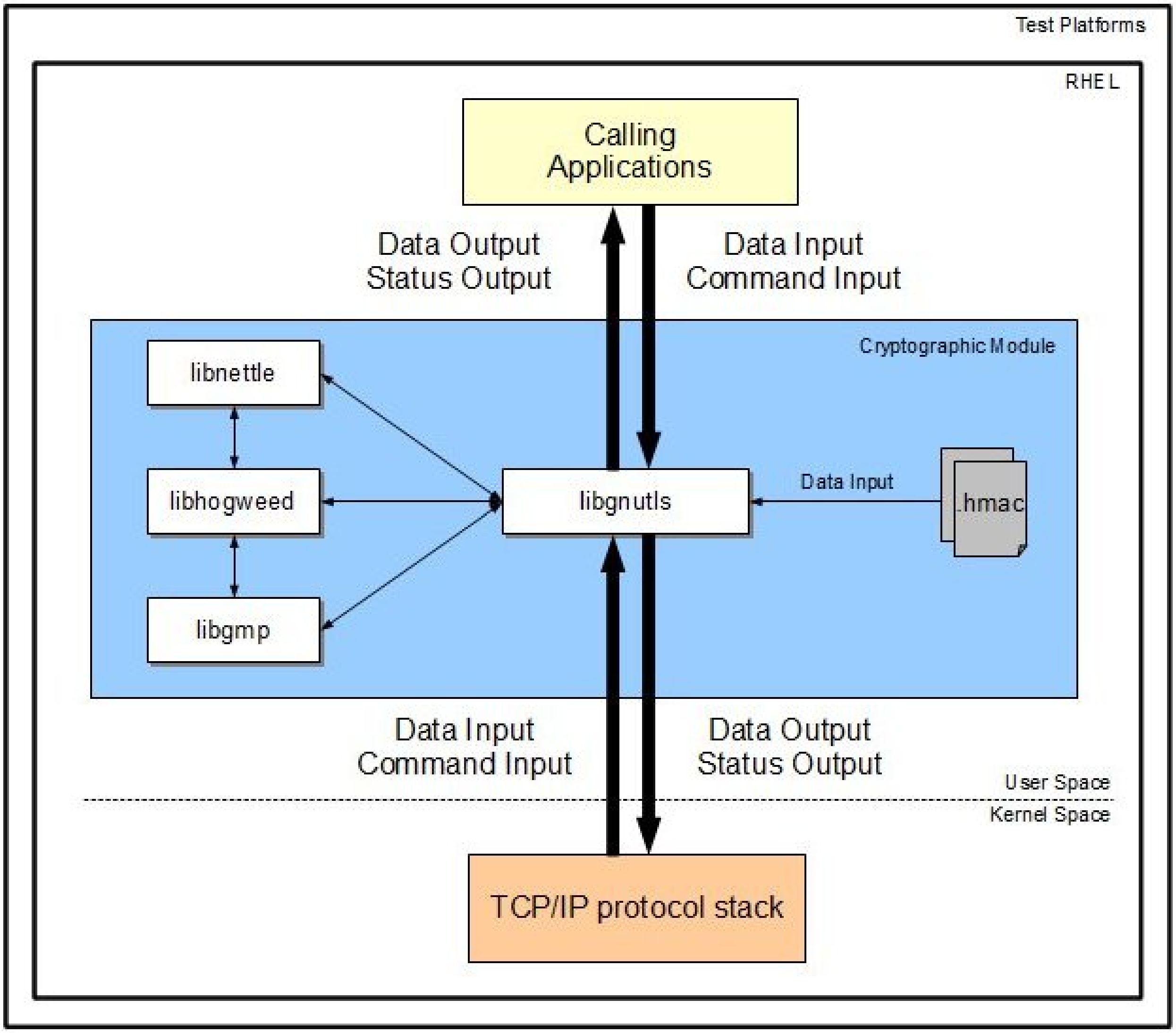

Figure 2. Software Block Diagram

2. Cryptographic Module Ports and Interfaces

The physical ports of the module are the same as the computer system on which it executes. The logical interface is a C-language Application Program Interface (API) through libgnutls library.

The Data Input interface consists of the input parameters of the API functions. The Data Output interface consists of the output parameters of the API functions. The Control Input interface consists of the actual API functions. The Status Output interface includes the return values of the API functions. The ports and interfaces are shown in the following table.

FIPS Interface	Physical Port	Module Interface
Data Input	Ethernet ports	API input parameters, kernel I/O – network or files on file system, TLS protocol
Data Output	Ethernet ports	API output parameters, kernel I/O – network or files on file system, TLS protocol
Control Input	Keyboard, Serial port, Ethernet port, Network	API function calls, TLS protocol
Status Output	Serial port, Ethernet port, Network	API return codes, TLS protocol
Power Input	PC Power Supply Port	N/A

Table 5: Ports and Interfaces

Note: The module is an implementation to support the TLS protocol defined in [RFC5246] and TLS is a port networking interface to provide secure channel between entities. When the calling application sends the data to the module, the module packages the data according to the TLS standard and sends it to other entity confidentially and integrity. The module is considered a user interface to use the TLS protocol to communicate with other remote entities securely through the network.

3. Roles, Services and Authentication

This section defines the roles, services, and authentication mechanisms and methods with respect to the applicable FIPS 140-2 requirements.

3.1. Roles

The module supports the following roles:

- **User role**: performs all services (in both FIPS mode and non-FIPS mode of operation), except module installation, configuration and initialization.
- **Crypto Officer role**: performs module installation, configuration and initialization.

The User and Crypto Officer roles are implicitly assumed by the entity accessing services implemented by the module.

3.2. Services

The module provides services to users that assume one of the available roles. All services are described in detail in the user documentation.

The following table lists the Approved services and the non-Approved but allowed services in FIPS mode of operation, the roles that can request the service, the Critical Security Parameters (CSP) involved and how they are accessed:

Service	Role	Keys/CSPs	Access
Cryptographic Library Services			
Symmetric Encryption and Decryption	User	AES 128, 192 or 256 bit key	Read
		3-key Triple-DES 192 bit key	
Asymmetric Key Generation in X509 Certificate	User	RSA public-private keys with at least 2048 bits of modulus size	Create
		DSA public-private keys with at least 2048 bits of public key size	
		ECDSA public-private keys with P-256, P-384 or P-521 curve	
Digital Signature Generation in X509 Certificate	User	RSA public-private keys with at least 2048 bits of modulus size	Read
		DSA public-private keys with at least 2048 bits of public key size	
		ECDSA public-private keys with P-256, P-384 or P-521 curve	
Digital Signature Verification	User	RSA public-private keys with at least 1024 bits of modulus size	Read
		DSA public-private keys with at least 1024 bits public key size	
		ECDSA public-private keys with P-256, P-384 or P-521 curve	

Service	Role	Keys/CSPs	Access
Public Key Verification	User	RSA public-private keys with at least 2048 bits of modulus size	Read
		DSA public-private keys with at least 2048 bits of public key size	
		ECDSA public-private keys with P-256, P-384 or P-521 curve	
Diffie-Hellman Parameters Generation, Import and Export	User	Diffie-Hellman domain parameters	Create, Read, Write
Import and Export Public Key	User	RSA, DSA or ECDSA public key	Read, Write
Import and Export Private Key	User	RSA, DSA or ECDSA private key	Read, Write
Keyed Hash (HMAC)	User	At least 112 bits HMAC Key	Read
Message Digest (SHA)	User	None	None
Random Number Generation (SP800-90A DRBG)	User	Entropy input string and seed	Read
Network Protocols Services (Note: The underlying algorithms are the same as the algorithm implementations provided in the Cryptographic Library Services.)			
TLS or DTLS Handshaking Initialization	User	None	None
TLS Alert Protocol	User	None	None
TLS Record Protocol	User	AES or Triple-DES key, HMAC key	Read
TLS Handshaking using X509 Certificates Authentication method with: • Diffie-Hellman KAS • EC Diffie-Hellman KAS • RSA-based Key Wrapping using RSA Encryption and Decryption Primitives (SP 800-56B Section 7.1)	User	AES or Triple-DES key, RSA, DSA or ECDSA public-private key, HMAC Key, Shared Secret, Diffie-Hellman domain parameters and EC Diffie-Hellman EC public-private keys	Create, Read
TLS Handshaking using Anonymous Authentication method with: • Diffie-Hellman KAS • EC Diffie-Hellman KAS	User	AES or Triple-DES key, DSA or ECDSA public-private key, HMAC Key, Shared Secret, Diffie-Hellman domain parameters and EC Diffie-Hellman EC public-private keys	Create, Read

Service	Role	Keys/CSPs	Access
TLS Handshaking using Pre-Shared Key (PSK) Authentication method with: • Diffie-Hellman KAS • EC Diffie-Hellman KAS • RSA-based Key Wrapping using RSA Encryption and Decryption Primitives (SP 800-56B Section 7.1)	User	AES or Triple-DES key, RSA, DSA or ECDSA public-private key, HMAC Key, Shared Secret, Diffie-Hellman domain parameters and EC Diffie-Hellman EC public-private keys	Create, Read
TLS X.509 Certificate Handling, including digital signature, key/certificate import and export, and support the following format: • PKCS#7 • PKCS#12 • Binary (DER) encoding • ASCII (PEM) encoding	User	RSA, DSA or ECDSA public-private key	Read, Write
TLS Extensions	User	RSA, DSA or ECDSA public-private key	Read
Other FIPS-related Services			
Show status	User	None	None
Self-test	User	None	None
Zeroize	User	All aforementioned CSPs	Zeroize
Module Installation	Crypto Officer	None	None
Module Initialization	Crypto Officer	None	None

Table 6: Services Available in FIPS mode

The following table lists the services only available in non-FIPS mode of operation.

Service	Role	Keys/CSPs	Access
FIPS 186-4 RSA Key Generation, Signature Generation, Public Key Verification or Key Wrapping with modulus size smaller than 2048 bits	User	RSA public-private keys with modulus size smaller than 2048 bits	Create, Read
FIPS 186-4 RSA Signature Verification with modulus size smaller than 1024 bits	User	RSA public-private keys with modulus size smaller than 1024 bits	Read
FIPS 186-2 RSA Key Generation	User	RSA public-private keys	Create
FIPS 186-4 RSA Signature Generation with non-Approved Message Digest algorithms	User	RSA public-private keys	Read

Service	Role	Keys/CSPs	Access
DSA Key Generation, Signature Generation or Public Key Verification with public key size smaller than 2048 bits	User	DSA public-private keys with public key size smaller than 2048 bits	Create, Read
DSA Signature Verification with public key size smaller than 1024 bits	User	DSA public-private keys with public key size smaller than 1024 bits	Read
Asymmetric Key for Data Encryption and Decryption	User	RSA public-private keys	Read
Diffie-Hellman KAS with key sizes smaller than 2048 bits	User	Diffie-Hellman domain parameters	Read
Random Number Generation or Symmetric Key Generation using Yarrow RNG or Lagged Fibonacci Pseudorandomness Generator	User	PRNG seed	Read
Encryption and Decryption using AES CTR, Blowfish, Camellia, CAST 128, DES, RC2, RC4, Salsa20, Serpent, or Twofish	User	8 to 2048 bits key	Read
Message Digest using GOST Hash R 34.11-94 (RFC4357), MD2, MD4, MD5, RIPEMD-160, or SHA-3	User	None	None
SP 800-132 Password-based Key Derivation Function (PBKDF2)	User	Password, derived keying material	Read, Create
MAC generation using UMAC	User	MAC key	Read
Support to use DANE Certificate	User	RSA, DSA and ECDSA public-private keys	Read
Support to use OpenPGP Certificate	User	RSA, DSA and ECDSA public-private keys	Read
Support to use PKCS#11 Certificate	User	RSA, DSA and ECDSA public-private keys	Read
Support to use the Secure RTP (SRTP) defined in RFC5764	User	AES and HMAC keys	Read
Support to use Trusted Platform Module (TPM)	User	RSA, DSA and ECDSA public-private keys	Create, Read

Table 7. Services Available in non-FIPS mode

Note: The module does not share CSPs between FIPS mode of operation and a non-FIPS mode of operation. All cryptographic keys used in the FIPS mode of operation must be generated in the FIPS mode or imported while running in the FIPS mode. The DRBG shall not be used for key generation for non-Approved services in non-FIPS mode.

More information about the services and their associated APIs listed in Table 6: Services Available in FIPS mode can be found in the GnuTLS documentation gnutls.pdf provided with the module's code or manpages from the module.

3.3. Operator Authentication

The module does not implement authentication. The role is implicitly assumed based on the service requested.

4. Physical Security

The module comprises of software only and thus does not claim any physical security.

5. Operational Environment

This module operates in a modifiable operational environment per the FIPS 140-2 definition.

5.1. Applicability

The module operates in a modifiable operational environment per FIPS 140-2 level 1 specifications. The module runs on a commercially available general-purpose operating system executing on the hardware specified in section 2.2.

The Red Hat Enterprise Linux operating system is used as the basis of other products which inlude but are not limited to:

- Red Hat Enterprise Linux Atomic Host
- Red Hat Virtualization (RHV)
- Red Hat OpenStack Platform
- OpenShift Container Platform
- Red Hat Gluster Storage
- Red Hat Ceph Storage
- Red Hat CloudForms
- Red Hat Satellite.

Compliance is maintained for these products whenever the binary is found unchanged.

The module operates in a modifiable operational environment per FIPS 140-2 level 1 specifications. The module runs on a commercially available general-purpose operating system executing on the hardware specified in section 1.1.

5.2. Policy

The operating system is restricted to a single operator (concurrent operators are explicitly excluded). The application that request cryptographic services is the single user of the module, even when the application is serving multiple clients.

In FIPS Approved mode, the ptrace(2) system call, the debugger (gdb(1)), and strace(1) shall be not used.

6. Cryptographic Key Management

The following table summarizes the Keys and Critical Security Parameters (CSPs) that are used by the cryptographic services implemented in the module in FIPS mode:

Keys/CSPs	Generation	Entry and Output	Zeroization
128, 192 or 256 bits AES Key	The key material can be generated during the Diffie-Hellman and EC Diffie-Hellman KAS, or can be generated by the SP 800-90A DRBG.	The key is passed into the module via API input parameters. The key can exit the module via TLS protocol by using RSA-based key wrapping.	Call gnutls_cipher_deinit() to zeroize the key.
192 bits Triple-DES Key	The key material can be generated during the Diffie-Hellman and EC Diffie-Hellman KAS, or can be generated by the SP 800-90A DRBG.	The key is passed into the module via API input parameters. The key can exit the module via TLS protocol by using RSA-based key wrapping.	Call gnutls_cipher_deinit() to zeroize the key.
At least 112 bits HMAC Key	The key material can be generated during the Diffie-Hellman and EC Diffie-Hellman KAS, or can be generated by the SP 800-90A DRBG.	The key is passed into the module via API input parameters. The key can exit the module via TLS protocol by using RSA-based key wrapping.	Call gnutls_hmac_deinit() to zeroize the key.
RSA Public-Private Keys with at least 2048 bits of modulus size	The RSA public-private keys with the modulus size of 2048 and 3072 bits are generated using FIPS 186-4 RSA Key Generation method and the random value used in key generation is generated using SP 800-90A DRBG.	The key is passed into the module via API input parameters, or imported via service calls. The public-private keys can be exported via service calls, and the public key can exit the module via TLS protocol.	Call gnutls_rsa_params_deinit(), gnutls_privkey_deinit() or gnutls_x509_privkey_deinit() to zeroize the key.
DSA Public-Private Keys where the public key size is at least 2048 bits	The DSA public-private keys with the public key size of 2048 and 3072 bits are generated using FIPS 186-4 DSA Key Generation method and the random value used in key generation is generated using SP 800-90A DRBG.	The key is passed into the module via API input parameters, or imported via service calls. The public-private keys can be exported via service calls.	Call gnutls_privkey_deinit() or gnutls_x509_privkey_deinit() to zeroize the key.

ECDSA Public-Private Keys where the key associated with P-256, P-384 or P-521 curve	The ECDSA public-private keys are generated using FIPS 186-4 ECDSA Key Generation method and the random value used in key generation is generated using SP 800-90A DRBG.	The key is passed into the module via API input parameters, or imported via service calls. The public-private keys can be exported via service calls.	Call gnutls_privkey_deinit() or gnutls_x509_privkey_deinit() to zeroize the key.
Diffie-Hellman domain parameters where the public key size is at least 2048 bits	The domain parameters used in Diffie-Hellman is generated using SP 800-90A DRBG.	The domain parameters are passed into the module via API input parameters, or imported via service calls. The domain parameters can be exported via service calls, and the generated public key can exit the module via TLS protocol.	Call or gnutls_deinit() or gnutls_dh_params_deinit() to zeroize the Diffie-Hellman domain parameters.
EC Diffie-Hellman EC public-private keys where the key associated with P-256, P-384 or P-521 curve	The components to generate the public-private keys used in EC Diffie-Hellman is generated using SP 800-90A DRBG.	The key is passed into the module via API input parameters. The public key can exist the module via TLS protocol.	Call gnutls_deinit() to zeroize the EC public-private keys.
Shared Secret	The shared secret (i.e., the key material) is generated by the module in the Diffie-Hellman or EC Diffie-Hellman KAS function.	The module does not import or export this CSP.	Call gnutls_deinit() to zeroize the shared secret.
Entropy Input String for DRBG seed	Obtained from NDRNG outside of the module’s logical boundary within the module's physical boundary	The module does not import or export the key or CSP.	Call gnutls_global_deinit() to zeroize the internal state of the DRBG.
DRBG internal V and Key	Generated internally in the DRBG	The module does not import or export the key or CSP.	Call gnutls_global_deinit() to zeroize the internal state of the DRBG.

Table 8: Keys/CSPs

6.1. Random Number Generation

The module employs a Deterministic Random Bit Generator (DRBG) based on [SP800-90A] for the creation of key components of asymmetric keys, symmetric keys, and random number generation.

The module implements the CTR_DRBG with AES-256 without derivation function and without prediction resistance. The CTR_DRBG is implemented in the libgnutls library and provides at least 128 bits of output data per each request.

The module uses the output of an NDRNG (i.e., /dev/urandom) as the entropy source for seeding the CTR_DRBG. The NDRNG is implemented in the O/S which is outside of the module's logical boundary within the module's physical boundary. This pseudo device is initialized by the O/S at system startup.

The module collects 384 bits of data from the NDRNG for generating the initial seed during initialization of the CTR_DRBG, and reseeding internally which occurs less than 2^{48} times of DRBG services request. The module obtains at least 112 bits of entropy from the NDRNG per each call.

The continuous self-tests on the output of NDRNG for seeding the SP800-90A DRBG is done by the O/S to ensure that consecutive random numbers do not repeat.

6.2. Key Generation

The Key Generation methods implemented in the module for Approved services in FIPS mode is compliant with [SP800-133].

For generating RSA, DSA and ECDSA keys the module implements asymmetric key generation services compliant with [FIPS186-4] and [SP800-90A]. A seed (i.e. the random value) used in asymmetric key generation is directly obtained from the [SP800-90A] DRBG.

The module does not offer a dedicated service for generating keys for symmetric algorithms or for HMAC in FIPS mode of operation. However, the module offers a DRBG compliant to [SP800-90A] to allow a caller to obtain random numbers which can be used as key material for symmetric algorithms or HMAC. The shared secret generated during the Diffie-Hellman or EC Diffie-Hellman KAS is also the key material for symmetric algorithms or HMAC.

The public and private key pairs used in the Diffie-Hellman and EC Diffie-Hellman KAS are generated internally by the module using the same DSA and ECDSA key generation compliant with [FIPS186-4] which is compliant with [SP800-56A].

CAVEAT:

- The module generates cryptographic keys whose strengths are modified by available entropy

6.3. Key Establishment/Key Derivation

The module supports the [SP800-56A] Diffie-Hellman with at least 2048 bits key size and EC Diffie-Hellman with P-256, P-384 or P-521 curve in FIPS mode. The Diffie-Hellman with less than 2048 bits key size is only available in non-FIPS mode.

The module also supports RSA key wrapping using encryption and decryption primitives with the modulus size of at least 2048 bits in FIPS mode. The modulus size of 1024 bits is only available in non-FIPS mode.

According to Table 2: Comparable strengths in NIST SP 800-57 Part1 (dated on March 8, 2007), the key sizes of RSA, Diffie-Hellman and EC Diffie-Hellman provides the following security strength for the corresponding key establishment method shown below:

- RSA key wrapping provides between 112 and 256 bits of encryption strength;
- Diffie-Hellman key agreement provides between 112 and 256 bits of encryption strength;
- EC Diffie-Hellman key agreement provides between 128 and 256 bits of encryption strength.

Note: As the module supports the RSA key pair with 15360 bits or more modulus size and the DSA key pair with the public key size of 15360 bits or more, the encryption strength 256 bits is claimed for RSA key wrapping and Diffie-Hellman KAS.

6.4. Key Entry and Output

The module does not support manual key entry or intermediate key generation key output.

For symmetric algorithms or for HMAC, the keys are provided to the module via API input parameters for the cryptographic operations. For asymmetric algorithms, the keys are also provided to the module via API input parameters. The module also provides the services to import and export public and private keys.

6.5. Key/CSP Storage

The module does not support persistent key storage. The keys and CSPs are stored as plaintext in the RAM.

The symmetric keys and HMAC keys are provided to the module via API input parameters, and are destroyed by the module using appropriate API function calls before they are released in the memory.

Asymmetric public and private keys are provided to the module via API input parameters, and are destroyed by the module using appropriate API function calls before they are released in the memory.

The HMAC key used for integrity test is stored in the .hmac file and relies on the operating system for protection.

6.6. Key/CSP Zeroization

The memory occupied by keys is allocated by regular libc malloc/calloc() calls. The application that uses the module is responsible for calling the appropriate destruction functions from the GnuTLS API to zeroize the keys or keying material. The destruction functions then overwrite the memory occupied by keys with pre-defined values and deallocates the memory with the free() call. In case of abnormal termination, or swap in/out of a physical memory page of a process, the keys in physical memory are overwritten by the Linux kernel before the physical memory is allocated to another process.

7. Electromagnetic Interference/Electromagnetic Compatibility (EMI/EMC)

MARKETING NAME.......................... PowerEdge R630
REGULATORY MODEL..................... E26S
REGULATORY TYPE......................... E26S001
EFFECTIVE DATE............................. September 03, 2014
EMC EMISSIONS CLASS.................. Class A

7.1. Statement of compliance

This product has been determined to be compliant with the applicable standards, regulations, and directives for the countries where the product is marketed. The product is affixed with regulatory marking and text as necessary for the country/agency. Generally, Information Technology Equipment (ITE) product compliance is based on IEC and CISPR standards and their national equivalent such as Product Safety, IEC 60950-1 and European Norm EN 60950-1 or EMC, CISPR 22/CISPR 24 and EN 55022/55024. Dell products have been verified to comply with the EU RoHS Directive 2011/65/EU. Dell products do not contain any of the restricted substances in concentrations and applications not permitted by the RoHS Directive.

8. Self-Tests

FIPS 140-2 requires that the module perform power-up tests to ensure the integrity of the module and the correctness of the cryptographic functionality at start up. In addition, some functions require continuous testing of the cryptographic functionality, such as the asymmetric key generation. If any self-test fails, the module returns an error code and enters the error state. No data output or cryptographic operations are allowed in error state.

See section 9.3 for descriptions of possible self-test errors and recovery procedures.

8.1. Power-Up Tests

The module performs power-up self-tests automatically when the module is loaded into memory; power-up tests ensure that the module is not corrupted and that the cryptographic algorithms work as expected. Input, output, and cryptographic functions cannot be performed while the module is in a self-test state because the module is single-threaded and will not return to the calling application until the power-up self-tests are completed. If any power-up self-test fails, the module returns the error code listed in section 9.3 and displays “Error in GnuTLS initialization” with the specific error message associated with the returned error code, and then enters the error state. The subsequent calls to the module will also fail - thus no further cryptographic operations are possible. If the power-up self-tests complete successfully, the module will return 0 and accepts cryptographic operation services request.

8.1.1. Integrity Tests

The integrity of the module is verified by comparing an HMAC-SHA-256 value calculated at run time with the HMAC value stored in the .hmac file that was computed at build time for each component of the module. If the HMAC values do not match, the test fails and the module enters the error state.

8.1.2. Cryptographic Algorithm Test

The module performs self-tests on all FIPS-Approved cryptographic algorithms supported in the approved mode of operation, using the known answer tests (KAT) and pair-wise consistency test (PCT), shown in the following table:

Algorithm	Power-Up Tests
AES	• KAT AES CBC encryption • KAT AES CBC decryption
Triple-DE	• KAT Triple-DES CBC encryption • KAT Triple-DES CBC decryption
HMAC	• KAT HMAC-SHA-1 • KAT HMAC-SHA-224 • KAT HMAC-SHA-256 • KAT HMAC-SHA-384 • KAT HMAC-SHA-512
SHS	• KATs SHA are covered in the KATs for HMAC as allowed with IG 9.1
DSA	• KAT DSA 2048-bit key with SHA-256 signature generation • KAT DSA 2048-bit key with SHA-256 signature verification • PCT, sign and verify

Algorithm	Power-Up Tests
RSA	• KAT RSA 2048-bit key with SHA-256 signature generation • KAT RSA 2048-bit key with SHA-256 signature verification
ECDSA	• KAT ECDSA (NIST P-256, P-384 and P-521) signature generation • KAT ECDSA (NIST P-256, P-384 and P-521) signature verification • PCT, sign and verify
Diffie-Hellman	Primitive "Z" Computation KAT
EC Diffie-Hellman	Primitive "Z" Computation KAT with P-256 curve
DRBG	KAT CTR_DRBG with AES-256 bit

Table 9: Power-Up Self-Tests

For the KAT, the module calculates the result and compares it with the known value. If the answer does not match the known answer, the KAT is failed and the module returns the error code and enters the error state. For the PCT, if the signature generation or verification fails, the module returns the error code and enters the error state.

As described in section 1.2, only one AES or SHA implementation from libnettle library written in C language or using the support from AES-NI or SSSE3 instructions is available at run-time. The KATs cover different implementations dependent on the implementations availability in the operating environment.

8.2. On-Demand Self-Tests

The on-demand self-tests is invoked by powering-off and reloading the module which causes the module to run the power-up tests again. During the execution of the on-demand self-tests, services are not available and no data output or input is possible.

8.3. Conditional Tests

The module performs conditional tests on the cryptographic algorithms, using the pair-wise consistency test (PCT) and Continuous Random Number Generator Test (CRNGT), shown in the following table:

Algorithm	Conditional Tests
DSA	Pairwise consistency test: signature generation and verification
ECDSA	Pairwise consistency test: signature generation and verification
RSA	Pairwise consistency test: encryption and decryption
DRBG	CRNGT is not required per IG 9.8
NDRNG	CRNGT is implemented in the Kernel

Table 10: Module Conditional Tests

9. Guidance

9.1. Crypto Officer Guidance

The binaries of the module are delivered via Red Hat Package Manager (RPM) packages. The Crypto Officer shall follow this Security Policy to configure the operational environment and install the module to be operated as FIPS 140-2 validated module.

The following version of the RPM packages containing the FIPS validated module and the operating environment settings:

Processor Architecture	RPM packages
x86_64	gnutls-3.3.26-9.el7.x86_64.rpm gmp-6.0.0-15.el7.x86_64.rpm nettle-2.7.1-8.el7.x86_64.rpm

Table 11: RPM packages

The RPM packages of the module can be installed by standard tools recommended for the installation of RPM packages on a Red Hat Enterprise Linux system (for example, yum, rpm, and the RHN remote management tool).

For proper operation of the in-module integrity verification, the prelink has to be disabled. This can be done by setting PRELINKING=no in the /etc/sysconfig/prelink configuration file. If module were already prelinked, the prelink should be undone on all the system files using the '`prelink -u -a`' command.

Operating Environment Configurations:

The configuration of the operating environment to support FIPS is provided by the dracut-fips package. The dracut-fips RPM package is only used for configuring the operating environment and does not provide any services to operators interacting with the module. Therefore the dracut-fips RPM package is outside the module's logical boundary. To configure the operating environment to support FIPS, the following shall be performed:

1. Install the dracut-fips package:

   ```
   # yum install dracut-fips
   ```

2. Recreate the INITRAMFS image:

   ```
   # dracut -f
   ```

After regenerating the initramfs, the Crypto Officer has to append the following string to the kernel command line by changing the setting in the boot loader:

```
fips=1
```

If /boot or /boot/efi resides on a separate partition, the kernel parameter boot=<partition of /boot or /boot/efi> must be supplied. The partition can be identified with the following command respectively:

```
"df /boot"
```

or

```
"df /boot/efi"
```

For example:

```
$ df /boot
Filesystem     1K-blocks   Used    Available    Use%  Mounted on
/dev/sda1       233191     30454   190296       14%   /boot
```

The partition of /boot is located on /dev/sda1 in this example. Therefore, the following string needs to be appended to the kernel command line:

```
"boot=/dev/sda1"
```

Reboot to apply these settings.

The Crypto Officer shall check the file /proc/sys/crypto/fips_enabled if the file exists and contains “1” . If the file does not exist or does not contain “1”, the operating environment is not configured to support FIPS and the module will not operate properly.

Once the operating environment has been configured to support FIPS, it is not possible to switch back to standard mode without terminating the module first.

Module Installations:

The Crypto Officer can install the RPM packages contains the module listed in Table 11: RPM packages based on the processor architecture. The integrity of the RPM is automatically verified during the installation of the module and the Crypto Officer shall not install the RPM file if the RPM tool indicates an integrity error.

9.2. User Guidance

The applications must be linked dynamically to run the module. Only the services listed in Table 6: Services Available in FIPS mode are allowed to be used in FIPS mode.

The libraries of GMP and Nettle provides the support of cryptographic operations to the GnuTLS library. The operator shall use the API provided by the GnuTLS library for the services. Invoking the APIs provided by the supporting libraries are forbidden.

9.2.1. TLS and Diffie-Hellman

The TLS protocol implementation provides both, the server and the client sides. As required by SP800-131A, Diffie-Helllman with keys smaller than 2048 bits must not be used.

The TLS protocol lacks the support to negotiate the used Diffie-Hellman key sizes. To ensure full support for all TLS protocol versions, the TLS client implementation of the cryptographic module accepts Diffie-Hellman key sizes smaller than 2048 bits offered by the TLS server.

For complying with the requirement of [FIPS140-2] to not allow Diffie-Hellman key sizes smaller than 2048 bits, the operator must ensure that:

- in case the module is used as TLS server, the Diffie-Hellman domain parameters must be 2048 bits or larger;
- in case the module is used as TLS client, the TLS server must be configured to only offer

Diffie-Hellman domain parameters of 2048 bits or larger.

9.2.2. AES GCM IV

In case the module's power is lost and then restored, the key used for the AES GCM encryption or decryption shall be re-distributed.

The AES GCM IV generation is in compliance with the [RFC5288] and shall only be used for the TLS protocol version 1.2 to be compliant with [FIPS140-2_IG] IG A.5; thus, the module is compliant with [SP800-52].

9.2.3. RSA and DSA Keys

The module allows the use of 1024 bit RSA and DSA keys for legacy purposes, including signature generation.

As per SP800-131A, RSA and DSA must be used at least 2048 bit keys in FIPS mode. To comply with the requirements of [FIPS140-2], the operator must therefore only use keys with at least 2048 bits in FIPS mode.

9.2.4. Symmetric Key Generation

The API function gnutls_key_generate() shall not be used in FIPS mode of operation. The caller shall call gnutls_rnd() which calls the DRBG compliant to [SP800-90A] to generate the key materials for symmetric keys or HMAC keys.

9.3. Handling Self-Test Errors

When the module fails any self-test, it will return an error code to indicate the error and enters error state that any further cryptographic operations is inhibited. Here is the list of error codes when the module fails any self-test or in error state:

Error Events	Error Codes	Error Messages
When the KAT or PCT fails at the power-up	GNUTLS_E_SELF_TEST_ERROR (-400)	"Error while performing self checks."
When the KAT of DRBG fails at the power-up	GNUTLS_E_RANDOM_FAILED (-206)	"Failed to acquire random data."
When the new generated RSA, DSA or ECDSA key pair fails the PCT	GNUTLS_E_PK_GENERATION_ERROR (-403)	"Error in public key generation."
When the module is in error state and caller requests cryptographic operations	GNUTLS_E_LIB_IN_ERROR_STATE (-402)	"An error has been detected in the library and cannot continue operations."

Table 12: Error Events, Error Codes and Error Messages

Self-test errors transition the module into an error state that keeps the module operational but prevents any cryptographic related operations. The module must be restarted and perform power-up self-test to recover from these errors. If failures persist, the module must be re-installed. When downloading the module, the Crypto Officer shall confirm from the RPM tool that the module was downloaded properly.

A completed list of the error codes can be found in Appendix C “Error Codes and Descriptions” in the gnutls.pdf provided with the module's code.

10. Mitigation of Other Attacks

RSA is vulnerable to timing attacks. In a setup where attackers can measure the time of RSA decryption or signature operations, blinding is always used to protect the RSA operation from that attack.

The internal API function of _rsa_blind() and _rsa_unblind() are called by the module for RSA signature generation and RSA decryption operations. The module generates a random blinding factor and include this random value in the RSA operations to prevent RSA timing attacks.

11. Glossary and Abbreviations

AES	Advanced Encryption Specification
AES-NI	Advanced Encryption Standard New Instructions
API	Application Program Interface
CAVP	Cryptographic Algorithm Validation Program
CAVS	Cryptographic Algorithm Validation System
CBC	Cypher Block Chaining
CMVP	Cryptographic Module Validation Program
CRNGT	Continuous Random Number Generator Test
CSP	Critical Security Parameter
CTR	Counter Mode
CVL	Component Validation List
DES	Data Encryption Standard
DRBG	Deterministic Random Bit Generator
DSA	Digital Signature Algorithm
DTLS	Datagram Transport Layer Security
ECC	Elliptic Curve Cryptography
FFC	Finite Field Cryptography
FIPS	Federal Information Processing Standards Publication
GCM	Galois Counter Mode
GPC	General Purpose Computer
HMAC	Hash Message Authentication Code
IG	Implementation Guidance
KAS	Key Agreement Schema
KAT	Known Answer Test
MAC	Message Authentication Code
NDRNG	Non-Deterministic Random Number Generator
NIST	National Institute of Science and Technology
O/S	Operating System
PCT	Pair-wise Consistency Test
RHEL	Red Hat Enterprise Linux
RNG	Random Number Generator
RPM	Red Hat Package Manager

RSA	Rivest, Shamir, Addleman
SHA	Secure Hash Algorithm
SSSE3	Supplemental Streaming SIMD Extensions 3
TLS	Transport Layer Security

12. References

FIPS140-2 **FIPS PUB 140-2 - Security Requirements For Cryptographic Modules**
May 2001
http://csrc.nist.gov/publications/fips/fips140-2/fips1402.pdf

FIPS140-2_IG **Implementation Guidance for FIPS PUB 140-2 and the Cryptographic Module Validation Program**
September 15, 2015
http://csrc.nist.gov/groups/STM/cmvp/documents/fips140-2/FIPS1402IG.pdf

FIPS180-4 **Secure Hash Standard (SHS)**
March 2012
http://csrc.nist.gov/publications/fips/fips180-4/fips 180-4.pdf

FIPS186-4 **Digital Signature Standard (DSS)**
July 2013
http://nvlpubs.nist.gov/nistpubs/FIPS/NIST.FIPS.186-4.pdf

FIPS197 **Advanced Encryption Standard**
November 2001
http://csrc.nist.gov/publications/fips/fips197/fips-197.pdf

FIPS198-1 **The Keyed Hash Message Authentication Code (HMAC)**
July 2008
http://csrc.nist.gov/publications/fips/fips198 1/FIPS-198 1_final.pdf

PKCS#1 **Public Key Cryptography Standards (PKCS) #1: RSA Cryptography Specifications Version 2.1**
February 2003
http://www.ietf.org/rfc/rfc3447.txt

RFC4347 **Datagram Transport Layer Security**
April 2006
https://tools.ietf.org/html/rfc4347.txt

RFC5246 **The Transport Layer Security (TLS) Protocol Version 1.2**
August 2008
https://tools.ietf.org/html/rfc5246.txt

RFC5288 **AES Galois Counter Mode (GCM) Cipher Suites for TLS**
August 2008
https://tools.ietf.org/html/rfc5288.txt

RFC6520 **Transport Layer Security (TLS) and Datagram Transport Layer Security (DTLS) Heartbeat Extension**
February 2012
https://tools.ietf.org/html/rfc6520.txt

SP800-38A **NIST Special Publication 800-38A - Recommendation for Block Cipher Modes of Operation Methods and Techniques**
December 2001
http://csrc.nist.gov/publications/nistpubs/800-38a/sp800-38a.pdf

SP800-52	**NIST Special Publication 800-52 Revision 1 - Guidelines for the Selection, Configuration, and Use of Transport Layer Security (TLS) Implementations** April 2014 http://nvlpubs.nist.gov/nistpubs/SpecialPublications/NIST.SP.800-52r1.pdf
SP800-56A	**NIST Special Publication 800-56A Revision 2 - Recommendation for Pair Wise Key Establishment Schemes Using Discrete Logarithm Cryptography** May 2013 http://nvlpubs.nist.gov/nistpubs/SpecialPublications/NIST.SP.800 56Ar2.pdf
SP800-67	**NIST Special Publication 800-67 Revision 1 - Recommendation for the Triple Data Encryption Algorithm (TDEA) Block Cipher** January 2012 http://csrc.nist.gov/publications/nistpubs/800-67-Rev1/SP-800-67-Rev1.pdf
SP800-90A	**NIST Special Publication 800-90A Revision 1 - Recommendation for Random Number Generation Using Deterministic Random Bit Generators** June 2015 http://nvlpubs.nist.gov/nistpubs/SpecialPublications/NIST.SP.800-90Ar1.pdf
SP800-135	**NIST Special Publication 800-135 Revision 1 - Recommendation for Existing Application-Specific Key Derivation Functions** December 2011 http://nvlpubs.nist.gov/nistpubs/Legacy/SP/nistspecialpublication800-135r1.pdf