Security Policy
for the
Reflection Security Component

04-RSC-0001 Version 2.7

March 2, 2005

Revision Table

Revision #	Date	Author	Description
1.0	4-Mar-04	E. Raisters	Initial Submission
2.0	21-Jun-04	E. Raisters, S. Tinsley	Added OpenSSL TLS information and addressed InfoGuard March 19 concerns.
2.1	30-Jul-04	S. Tinsley, E. Raisters	Addressed InfoGard June 28 concerns
2.2	2-Sep-04	S. Tinsley	Incorporate changes from 8/30/04.
2.3	30-Sep-04	E. Raisters, S. Tinsley, Z, Evans	Addressed InfoGard Sept 27 comments
2.4	1-Oct-04	E. Raisters	Addressed InfoGard Sept 30 comments
2.5	5-Oct-04	S. Tinsley	Remove DH Public Exponent from list of CSPs
2.6	25-Feb-05	S. Tinsley	Address comments from NIST.
2.7	2-Mar-05	E. Raisters	Address comments from NIST.

Table of Contents

1. Module Overview

The Reflection cryptographic module is a software module that encompasses specific security functions utilized by the Reflection products. The module consists of several separate Dynamic Link Libraries (DLLs) that can only be invoked by the Reflection products The physical boundary for the crypto module (as shown in figure 1) is defined as the enclosure (the PC case) of the computer system on which the cryptographic software module is to be executed. The module's software boundary (as shown in figure 2) includes rfips.dll, the openssh module (openssh.dll), the kerberos crypto modules (rscrypto.dll, bdes56.dll, desauth.dll) and the openssl module (openssl.dll) and the Microsoft CryptoAPI. The physical configuration of the module as defined in FIPS PUB 140-2, is Multi-Chip Standalone. The primary purpose for this software module is to provide secure communication over TCP/IP networks between a host computer and a PC.

This security policy and the FIPS 140-2 validation applies to version 12.0.3 of the cryptographic modules in the Reflection products.

Cryptographic boundary

The following two block diagrams illustrate the cryptographic module, its relationships to other components, and the physical and logical cryptographic boundaries. The cryptographic module includes one third-party component, the FIPS 140-1 validated CryptoAPI cryptographic library, which is part of the Microsoft Windows operating system. Table 1 details the acceptable CryptoAPI modules and their FIPS 140-1 certificate number.

Figure 1 – Image of the Physical Boundary of the Cryptographic Module

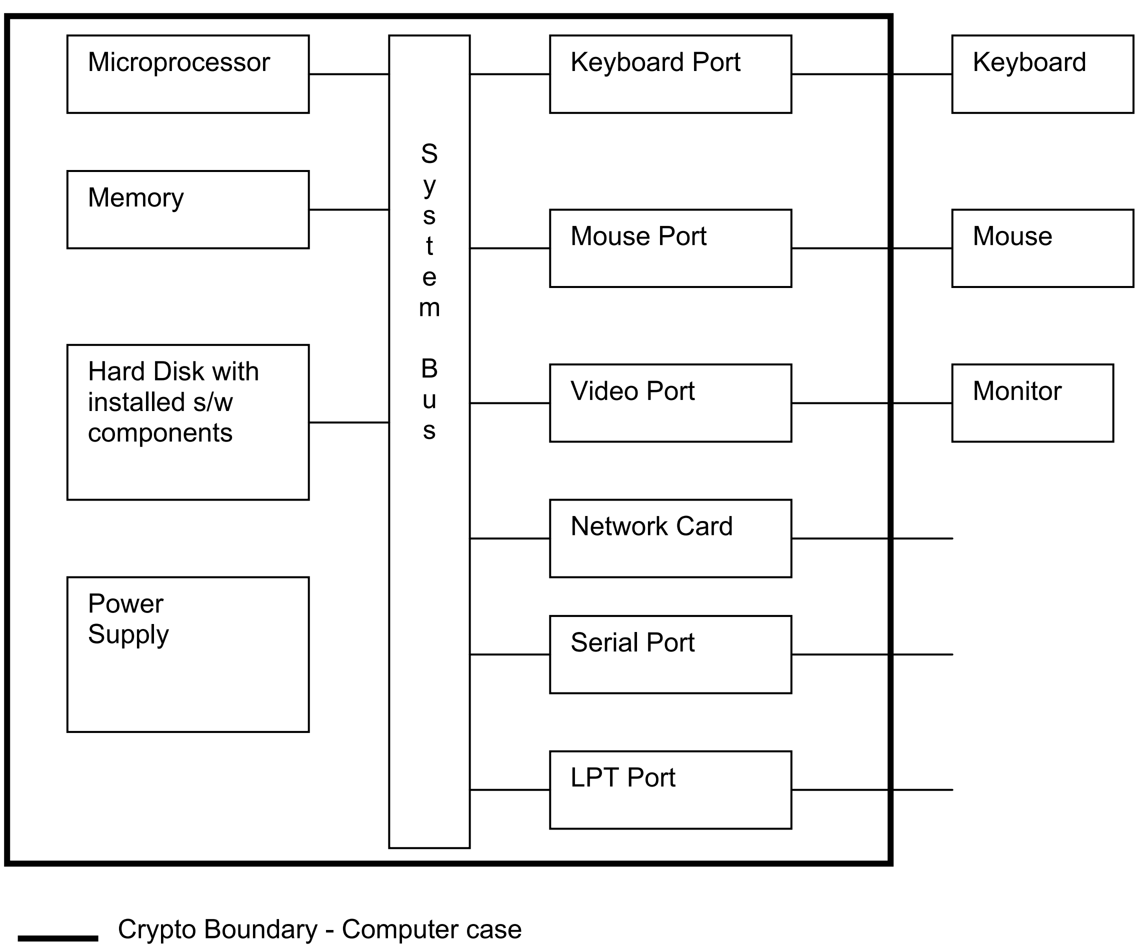

Figure 2 – Diagram of the Logical Boundary of the Cryptographic Module

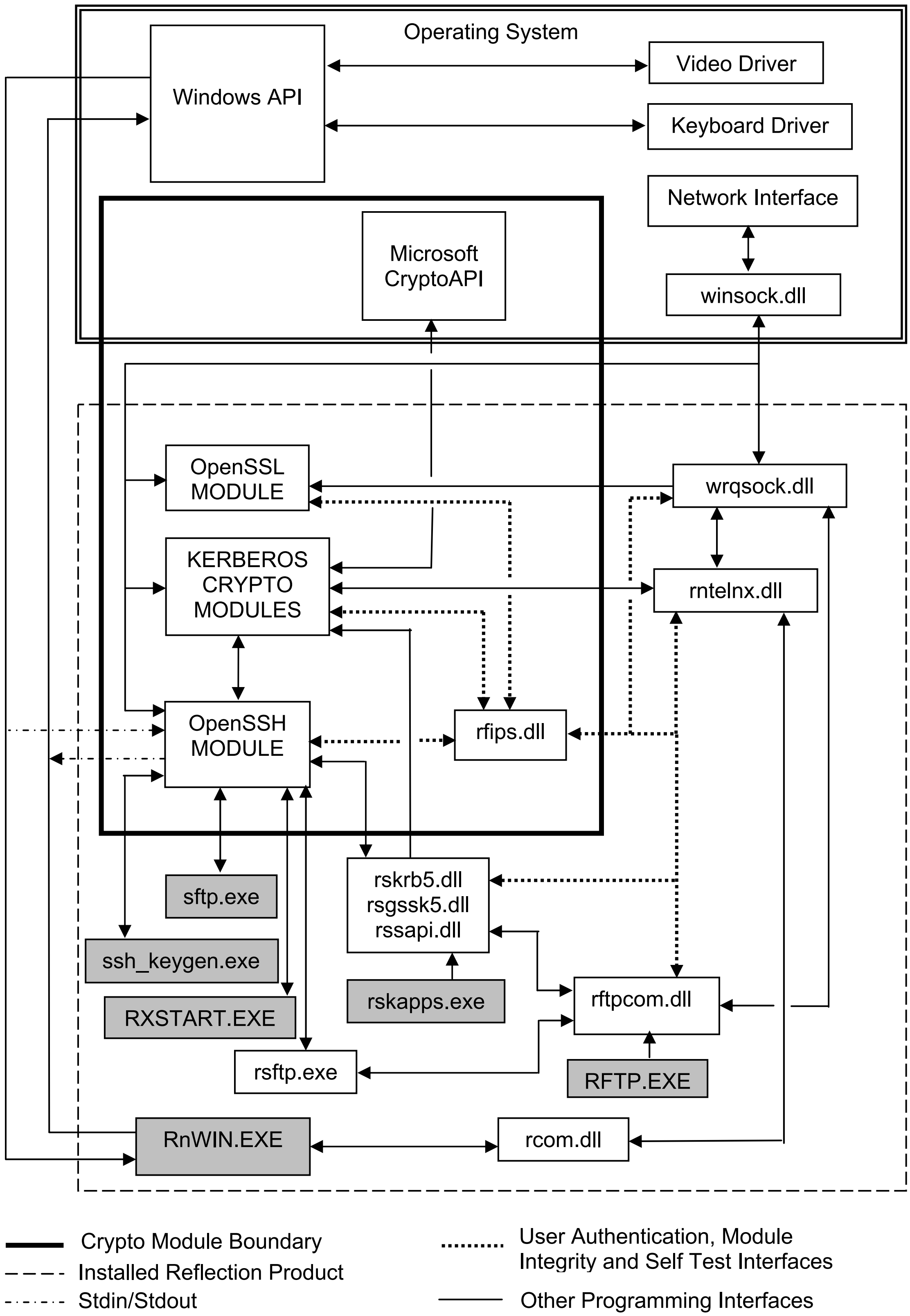

Table 1 – Acceptable CryptoAPI modules

Module	Cert#
Base DSS Cryptographic Provider, Base Cryptographic Provider, DSS/Diffie-Hellman Enchanced Cryptographic Provider, and Enhanced Cryptographic Provider (Version 5.0.2150.1)	76
Base DSS Cryptographic Provider, Base Cryptographic Provider, DSS/Diffie-Hellman Enhanced Cryptographic Provider, and Enhanced Cryptographic Provider ((Base DSS: 5.0.2150.1391 [SP1], 5.0.2195.2228 [SP2] and 5.0.2195.3665 [SP3]), (Base: 5.0.2150.1391 [SP1], 5.0.2195.2228 [SP2] and 5.0.2195.3839 [SP3]), (DSS/DH Enh: 5.0.2150.1391 [SP1], 5.0.2195.2228 [SP2] and 5.0.2195.3665 [SP3]), (Enh: 5.0.2150.1391 [SP1], 5.0.2195.2228 [SP2] and 5.0.2195.3839 [SP3]))	103
Enhanced Cryptographic Provider (RSAENH) Version 5.1.2600.1029 also known as Base Cryptographic Provider (Versions 5.1.2518.0 and 5.1.2600.1029)	238
DSS/Diffie-Hellman Enhanced Cryptographic Provider for Windows XP (Software Version 5.1.2518.0)	240

2. Security Level

The cryptographic module meets the overall requirements applicable to Level 1 security of FIPS 140-2.

Table 2 - Module Security Level Specification

Security Requirements Section	Level
Cryptographic Module Specification	3
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	3
Self-Tests	1
Design Assurance	1
Mitigation of Other Attacks	N/A

3. Modes of Operation

The module supports both an Approved and a non-Approved mode of operation. The mode of operation is selected prior to power-up module (before the module instantiation). The module indicates whether it is in an Approved mode of operation when the operator invokes the *Show Status* service.

Approved mode of operation

In FIPS mode, the cryptographic module only supports FIPS Approved algorithms as follows:

- In the OpenSSH client (SSHv2 only):
    - RSA or DSA keys (minimum of 1024 bits) for SHA-1 based digital signature generation and verification
    - AES (128-bit key) for encryption
    - Triple-DES (168-bit, three key) for encryption
    - HMAC-SHA-1 for MACing
    - SHA-1 for hashing
    - ANSI X9-31 A.2.4 approved deterministic random number generator
    - supports the commercially available Diffie-Hellman protocol for key establishment
- In the OpenSSL client:
    - RSA or DSA keys (minimum of 1024 bits) for SHA-1 based digital signature generation and verification
    - AES (128-bit key) for encryption
    - Triple-DES (168-bit, three key) for encryption
    - SHA-1 for hashing
    - HMAC-SHA-1 MACing
    - HMAC-MD5 for TLS key establishment only (per TLS protocol standard)
    - ANSI X9-31 A.2.4 approved deterministic random number generator
    - supports the commercially available Diffie-Helman protocol for key establishment

  The supported ciphersuites for TLS are:

  168 bit key strength
  TLS_RSA_WITH_3DES_EDE_CBC_SHA
  TLS_DHE_DSS_WITH_3DES_EDE_CBC_SHA

  128 bit key strength
  TLS_RSA_WITH_AES_128_CBC_SHA
  TLS_DHE_DSS_WITH_AES_128_CBC_SHA

Note: In FIPS mode, the cryptographic module's Kerberos client supports the following cryptographic algorithms in order to provide operator authentication only. Not all of these algorithms are FIPS approved.

- In the Kerberos client (operator authentication only)

  - Triple-DES (168-bit, three key)
  - DES (56-bit key)
  - HMAC-SHA-1 for MACing
  - SHA-1 for hashing
  - MD5 for hashing
  - MD4 for hashing
  - FIPS 186-2 approved deterministic random number generator provided by the operating system's crypto module.

The cryptographic module may be configured for FIPS mode via the Microsoft Group Policy Editor, specifying Allow non-FIPS-mode connections = Disabled. The user can determine if the cryptographic module is running in FIPS vs. non-FIPS mode via execution of the "Show Status" service.

Non-approved mode of operation

In non-FIPS mode, the cryptographic module provides non-FIPS Approved algorithms as follows:

- In the OpenSSH client (SSHv1 and SSHv2):
  - RSA and DSA keys (minimum of 512 bits) for digital signature generation and verification
  - RSA-1 keys for user authentication
  - Blowfish (128-bit key) for encryption
  - Arcfour (128-bit key) for encryption
  - CAST (128-bit key) for encryption
  - DES (56-bit key) for encryption
  - Ripemd160 for hashing
  - MD5 for hashing
- In the SSL/TLS encryption module:
  - Arcfour (40-, 56- or 128-bit key) for encryption
  - DES (56-bit key) for encryption
  - MD5 for hashing
  - MD4 for hashing

4. Ports and Interfaces

The cryptographic module's interfaces are defined by the standard PC enclosure. As tested, the ports used by the module are the keyboard port, monitor port, mouse port, a network port, serial/parallel/USB ports and a power port/plug. The operating system and application layer software map these ports to the logical interfaces described in table 3.

Table 3 - - physical port to logical interface map

Physical Port	Logical Interface
Keyboard port, mouse port, network port, serial/parallel/USB ports	Data input
Monitor port , network port	Data output
Keyboard port, mouse port, network port	Control Input
Monitor port , network port	Status output
Power interface	Power

RFIPS Interface	Inputs/Outputs
fipsInitialize – invokes self test services for all the individual modules. Performs checksum verification of the crypto module.	Status Output: success/fail
fipsAuthenticate – validates the authentication token passed by the user	Data input: plaintext authentication token Status Output: success/fail
fipsMode – returns true if system is in fipsMode.	Status Output: true/false(fipsMode)
fipsSatus – returns the fips state of the crypto module.	Status Output: fipsError/OK
DestroyFile – uses a Wiper class to overwrite and remove a file.	Data Input: handle to a Wiper, plaintext filename (and length of filename)

OpenSSH Interface (SSHv2 only)	Inputs/Outputs
fipsSelfTest – runs KAT's for RSA, DSA, AES, DES, TDES, SHA-1 crypto routines.	Status Output: success/fail
fipsZeroize – Disconnects any current connections and zeroizes all keys in RAM.	
newSSH – Initializes crypto module, authenticates user and returns class interface that enables access to SSH API's.	Data Input: plaintext authentication token (512 bits) Control Output: Issh class interface Status Output: success/fail
newSFTP - – Initializes crypto module, authenticates user and returns class interface that enables access to SFTP API's.	Data Input: plaintext authentication token (512 bits) Control Output: Isftp class interface Status Output: success/fail
Connect – starts an SSHv2 connection, authenticates remote host by receiving server public key and verifying digital signature and/or using Kerberos authentication. Establishes a shared secret via Diffie-Hellman and generates Traffic Encryption Keys based on the shared secret. Performs user authentication over an encrypted tunnel by using a password, public key cryptography or Kerberos authentication Starts thread to receive further data input/output.	Data Input: host name, user name, status output method, password Status Output: success/fail
ConnectAsync – same as Connect, but users separate thread to perform functionality and returns	

OpenSSH Interface (SSHv2 only)	Inputs/Outputs
immediately.	
ConnectAsyncCleanup – cleans up ConnectAsync resources	
sftpmain – performs Connect functionality and then waits for input from the keyboard or batch file, process input and issues sftp commands.	Data Input: host name, user name, status output method, password, commands, plaintext data for file transfer Data Output: plaintext data for file transfer
Disconnect – terminates connection and removes all Traffic Encryption Keys from RAM.	
Write – sends plaintext data to remote host through an encrypted tunnel.	Data Input: plaintext, length Data Output: ciphertext, length Status Output: success/fail
Read - receives plaintext data from remote host through an encrypted tunnel.	Data Input: ciphertext, length Data Output: plaintext, length Status Output: success/fail
Setting – Displays dialog allowing configuration for an ssh session to a remote host. The dialog provides the ability to generate an RSA or DSA public/private key pair (minimum of 1024 bits) which can be used to perform user authentication.	Data Input: filename, number bits Data Output: RSA or DSA public/private key pair (at least 1024 bits) – not output from the physical boundary
InitSFTP – sends commands through the encrypted tunnel to initialize the sftp subsystem.	Status Output: success/fail
pwd – requests and receives the present working directory on the remote host	Data Output: servers present working directory
ls - requests and receives the directory listing for a directory on the remote host	Data Input: specified directory, formatting specifications Data Output: directory listing Status Output: success/fail
cd – requests that the present working directory be changed	Data Input: specified directory Status Output: success/fail
get – requests a file be transferred to the local machine from the remote host	Data Input: filename, ciphertext Data Output: plaintext Status Output: success/fail
put – requests a file be transferred from the local machine to the remote host.	Data Input: filename Data Output: ciphertext Status Output: success/fail
mkdir - requests that a directory be created on the remote host.	Data Input: specified directory name Status Output: success/fail
rm - requests that a file be deleted from the remote host	Data Input: specified file Status Output: success/fail
rmdir - requests that a directory on the remote host be deleted.	Data Input: specified directory Status Output: success/fail
rename - requests that the a filename be changed on the remote host.	Data Input: current filename, new filename Status Output: success/fail
ssh_keygen_main – creates or edits a public/private keypair.	Data Input: filename, number bits Data Output: RSA or DSA public/private key pair (at least 1024 bits) – not output from the physical boundary

OpenSSL Interface (TLS only)	Inputs/Outputs
GetIOpenSSL – Initializes crypto module, authenticates user and returns class interface that enables access to the SSL API.	Data Input: plaintext authentication token (512 bits) Control Output: IOpenSSL class interface Status Output: success/fail

OpenSSL Interface (TLS only)	Inputs/Outputs
fipsSelfTest – runs KAT's for RSA, DSA, AES, DES, TDES, SHA-1 crypto routines.	Status Output: success/fail
fipsZeroize – Disconnects any current connections and zeroizes all keys in RAM.	
setSecLevel – set's the minimum key length the module will negotiate for the data encryption algorithm. Default results in making all approved combinations available.	Data Input: plaintext socket ID Control Input: plaintext key length
setTLSTimeout – sets the timeout value for the handshake process.	Data Input: plaintext socket ID Control Input: timeout value
setProperty – sets other properties. A socket ID must be used to associate the property with a session. *useRwebProxy – true if the cx is routed through the RWeb Proxy Server.* *RwebProxyToken – an identification token for authenticating to the Proxy Server.* *WRQAsyncSelect – fn to call for socket activity notifications.*	Control Input: socket ID (optional) Data Input: plaintext property type(below) and value:
startSecurity – starts the TLS handshake process based on an active socket connection. The handshake process negotiates with the server the key exchange method, the encryption algorithm and hash algorithm. After the handshake, this process authenticates the remote host by receiving server public key and verifying digital signature. A shared secret is established either with the module sending an RSA encrypted secret or both sides calculating the secret via Diffie-Hellman A master secret is calculated from the shared secret and used to generate Traffic Encryption Keys. The server can request a client certificate for Client Authentication during the handshake process. The module obtains the appropriate client certificate from the operating system's certificate store and sends it to the server after receiving the request. On successfully completing the handshake, this service starts a thread to receive further ciphertext data from the server.	Data Input: socket ID, host name Status Output: success/fail
shutdown – sends a TLS Close Notify message to the server and calls the Winsock shutdown function.	Data Input: plaintext socket ID, plaintext Winsock shutdown type
closesocket – terminates connection and removes all Traffic Encryption Keys from RAM.	Data Input: plaintext socket ID
send – sends plaintext data to remote host through an encrypted tunnel.	Data Input: plaintext, length Data Output: length Status Output: success/fail
recv - receives plaintext data (previously decrypted) from remote host.	Data Output: plaintext, length Status Output: success/fail
getSecLevel – returns the negotiated strength of encryption for the session identified by the socket ID.	Data Input: plaintext, socket ID Data Output: plaintext, encryption strength

OpenSSL Interface (TLS only)	Inputs/Outputs
setTLSTimeout – returns the timeout value for the handshake process.	Data Input: plaintext socket ID Data Output: plaintext timeout value
select – overrides the Winsock select function to provide information on whether decrypted data is available to the caller, and if the socket is ready to write.	Data Input: plaintext FDs of sockets to check, plaintext timeout value
FreeOpenssl – zeroizes any remaining CSPs and frees the library from memory.	Data Input: plaintext Interface address, plaintext Handle for the dll module.

Note: The following are interfaces in the kerberos crypto module that provide operator authentication only.

Kerberos Interface	Inputs/Outputs
csSelfTest – runs KAT's for kerberos crypto libraries – DES, TDES, MD5, SHA-1.	Status Output: success/fail
csZeroize – zeroizes all keys in RAM as well as any cache files on disk.	
csFipsMode – reports whether or not the module is in an approved mode.	
csInitialize - Initializes crypto module and authenticates the user.	Data Input: plaintext authentication token (512 bits) Status Output: success/fail
csEncrypt – encrypts plaintext data according to an encryption context.	Control Input: encryption context established by calling csUseType and csProcessKey Data Input: plaintext, length Data Output: ciphertext Status Output: success/fail
csBulkEncrypt – encrypts plaintext data according to an encryption context.	Control Input: encryption context established by calling csUseBulkType and csBulkProcessKey Data Input: plaintext, length Data Output: ciphertext Status Output: success/fail
csUseType – creates an encryption context and establishes the encryption algorithm to use for the context.	Data Input: encryption type Control Output: encryption context
csUseBulkType – creates an encryption context and establishes the encryption algorithm to use for the context.	Data Input: encryption type Control Output: encryption context
csProcessKey – sets the encryption key to use for an encryption context.	Control Input/Output: encryption context Data Input: key identifier
csBulkProcessKey - sets the encryption key to use for an encryption context.	Control Input/Output: encryption context Data Input: key identifier
csFinishKey – frees the resources of an encryption context	
csBulkFinishKey - frees the resources of an encryption context	
csEncryptCredEncPart – encodes a kerberos message using ASN.1, part of which contains encrypted data containing an encryption key.	Data Input: structured kerberos data (including ID of a key to be encrypted) keyID (used to encrypt parts of the message) Data Output: encoded data, part of which is ciphertext Status Output: success/fail
csEncryptAuthenticator – ASN.1 encodes a kerberos message, part of which contains encrypted data containing an encryption key.	Data Input: structured kerberos data (including ID of a key to be encrypted) keyID (used to encrypt parts of the message) Data Output: encoded data, part of which is

Kerberos Interface	Inputs/Outputs
	ciphertext Status Output: success/fail
csDecrypt – decrypts plaintext data according to an encryption context.	Control Input: encryption context established by calling csUseType and csProcessKey Data Input: ciphertext, length Data Output: plaintext Status Output: success/fail
csBulkDecrypt – decrypts plaintext data according to an encryption context.	Control Input: encryption context established by calling csUseBulkType and csBulkProcessKey Data Input: plaintext, length Data Output: ciphertext Status Output: success/fail
csUseType – creates an encryption context and establishes the encryption algorithm to use for the context.	Data Input: encryption type Control Output: encryption context
csUseBulkType – creates an encryption context and establishes the encryption algorithm to use for the context.	Data Input: encryption type Control Output: encryption context
csProcessKey – sets the encryption key to use for an encryption context.	Control Input/Output: encryption context Data Input: key identifier
csBulkProcessKey - sets the encryption key to use for an encryption context.	Control Input/Output: encryption context Data Input: key identifier
csFinishKey – frees the resources of an encryption context	
csBulkFinishKey - frees the resources of an encryption context	
csDecryptKDCReply – decrypts a kerberos message and decodes it using ASN.1. Part of this message contains a new encryption key, which is stored in RAM and assigned a new ID.	Data Input: ciphertext, keyID Data Output: structured kerberos data, key ID Status Output: success/fail
csDecryptAPReply - decrypts a kerberos message and decodes it using ASN.1. Part of this message contains a new encryption key, which is stored in RAM and assigned a new ID.	Data Input: ciphertext, keyID Data Output: structured kerberos data, key ID Status Output: success/fail
csCalculateChkSum – computes the checksum of the given data	Data Input: checksum type, data, length Data Output: checksum Status Output: success/fail
csVerifyChkSum - computes the checksum of the given data	Data Input: checksum type, data, length, checksum Status Output: success/fail
csStrToKey – derives an encryption key from a given string	Data Input: encryption type, string (typically a user's password) Data Output: key ID Status Output: success/fail
csGenerateRandomKey – generates a new encryption key	Data Input: seed Data Output: keyID Status Output: success/fail
csBulkGenerateRandomKey – generates a new encryption key	Data Input: seed Data Output: keyID Status Output: success/fail
csIsValidEncKey – tests a key to make sure it is valid and cryptographically strong	Data Input: key ID Status Output: success/fail
csIsValidBulkEncKey – tests a key to make sure it is valid and cryptographically strong	Data Input: key ID Status Output: success/fail
csAddKeyVariant – does an exclusive or on the	Data Input: key ID

Kerberos Interface	Inputs/Outputs
key data with a known constant	
csValidType – determines if the specified encryption type is supported	Data Input: encryption type Status Output: success/fail
csValidBulkEncType – determines if the specified encryption type is supported	Data Input: encryption type Status Output: success/fail
csRandConfounder – gets a random number, intended to be used as a unique session identifier or an IV	Data Input: number of bytes Data Output: random bytes
csCopyKeyBlockContents – copies the key data of the specified ID, stores the copy in RAM with a different ID	Data Input: key ID Data Output: key ID
csCopyKeyBlock– copies the key data of the specified ID, stores the copy in RAM with a different ID	Data Input: key ID Data Output: key ID
csFreeKeyBlock – zeroizes the key data and frees its resources	Data Input: key ID
csCleanupKeyBlock – zeroizes the key data and frees its resources	Data Input: key ID
FileCachePtr – returns an interface pointer to the file management class	Control Output: interface pointer to file management cache
MemoryCachePtr– returns an interface pointer to the memory management class	Control Output: interface pointer to memory management cache
Open – opens a cache for read/write access	Data Input: cache name Control Output: cache handle Status Output: success/fail
Close – closes an opened cache.	Control Input: cache handle
Destroy – destroys a cache.	Data Input: cache name
Seek – advances cache handles current read/write location.	Control Input: cache handle Status Output: success/fail
SkipVer – sets the cache handles current read/write location just after the cache version number.	Control Input: cache handle Status Output: success/fail
ReadKey – reads in bytes from the cache and store them in RAM as key data. Assigns an ID to the key data that is returned to the user	Data Output: key ID Status Output: success/fail Control Input: cache handle
Write – writes out data to the cache.	Control Input: cache handle Data Input: data, number bytes Status Output: success/fail
WriteKey – writes out the specified key data to the cache in plaintext form.	Control Input: cache handle Data Input: key ID Status Output: success/fail

5. Identification and Authentication Policy

Assumption of roles

The Reflection for Windows Cryptographic module has a single Authenticated User (programmatic entity) which may access all services implemented in the cryptographic module through the proprietary API. In addition, the module identifies a Crypto Officer role responsible for zeroization services. The module also provides role based authentication through the Kerberos services.

The module shall not support a maintenance role.

Table 4 - Strength of authentication mechanisms

Authentication Mechanism	Strength of Mechanism
Kerberos operator authentication	The strength of authentication is a minimum of 1 in 2^56 (1 in 72,057,594,037,927,936) for a single guess.
Authentication Bitstring	The strength of authentication is a minimum of 1 in 32^16 (1 in 1,208,925,819,614,629,174,706,176) for a single guess.

6. Access Control Policy

Services are programmatically invoked through a C or C++ based Application Programming Interface.

Roles and Services

Table 5 – Services for Roles

Role	Services
Authenticated User: This role shall provide all of the services necessary for the secure transport of data over an insecure TCP/IP network.	**OpenSSH (SSHv2)** • **Connect** Establishes a network connection with a remote host, authenticates the server using public key cryptography, uses Diffie-Hellman to establish a shared secret which is used to establish keys for an encrypted tunnel. Also passes user authentication data through the encrypted tunnel. • **Disconnect** Terminates connection with remote host. • **Send Data** Sends data through an encrypted tunnel. • **Receive Data** Receives data through an encrypted tunnel. • **Generate RSA/DSA key pair** Creates an RSA or DSA public/private key pair of a specified key length (at least 1024 bits). • **SFTP commands** Processes a variety of file management commands through an encrypted tunnel. **OpenSSL (TLS only)** • **ConfigureSession**

Role	Services
	Configures the parameters for creating a new TLS session. • **StartSecurity** Establishes a network connection with a remote host, authenticates the server using public key cryptography, uses RSA or Diffie-Hellman to exchange/establish a shared secret which is used to establish keys for an encrypted tunnel. • **Disconnect** Terminates connection with remote host. • **Send Data** Sends data through an encrypted tunnel. • **Receive Data** Receives data through an encrypted tunnel. • **Get Session Data** Obtains the value of the session parameters. • **Free Interface** Frees the interface from memory. ***Note: The following are interfaces that only provide operator authentication.*** **<u>Kerberos</u>** • **Encrypt** Encrypts data (DES/TDES). • **Encode and Encrypt** Encodes and encrypts data (DES/TDES). • **Decrypt** Decrypts data (DES/TDES). • **Decrypt and Decode** Decrypts (DES/TDES) and decodes data. • **Calculate CheckSum** Calculates the checksum of the given data. • **Verify CheckSum** Calculates the checksum of the given data and compares it with a given checksum. • **String To Key** Derives a key (DES/TDES) from a string (typically a users password). • **Generate Key** Uses Microsoft's RNG, CryptGenRandom, to produce a symmetric key of the specified key type (DES/TDES). • **Is Key Valid** Verifies that a key (DES/TDES) is of the correct form. • **Add Key Variant** Adds variance to a key by xoring a known constant to the key. • **Is Valid Encryption Type** Determines if a specified ID correctly identifies a valid encryption type supported within the module. • **Random Number** Uses Microsoft's RNG, CryptGenRandom, to produce a specified number of random bytes. • **Copy Key** Makes a duplicate copy of a key in memory. • **Cleanup Key** Zeroizes and removes a key from memory.

Role	Services
	• **Get File Cache** Returns an interface pointer that enables access to file cache management routines. • **Get Memory Cache** Returns and interface pointer that enables access to memory cache management routines. • **Open** Prepares a specified cache for read/write access. • **Close** Closes a specified cache. • **Destroy** Zeroizes and deletes a specified cache. • **Exists** Determines if a specified cache exists. • **Seek** Advances the file descriptor for specified cache. • **Read** Reads in the data from the cache. • **ReadKey** Reads in data from the cache and uses them as a key (DES/TDES). • **Write** Writes out data to the cache. • **WriteKey** Writes out key data to the cache.
CryptoOfficer	• **Zeroize** This service actively destroys all plaintext critical security parameters. In addition to calling this service, the Crypto Officer must manually delete the associated files described in Crypto Officer Guide document. • **Zeroize File** Zeroizes a specified file

Services Not Requiring Authentication:

The cryptographic module supports the following services, which do not require authentication:

- Initialize: Runs the Self-tests and verifies that the module has not been modified.
- Authenticate: Validates the authentication token passed by the user
- Show status: This service provides the current status of the cryptographic module.
- Zeroize: This service actively destroys all plaintext critical security parameters.
- Zeroize File: Zeroizes a given file

Kerberos services not requiring authentication

- ASN1 Encoding: Encodes kerberos data structures
- ASN1 Decoding: Decodes data into kerberos structures
- Memory Allocation
- Memory Deallocation
- Utility Functions

Table 6 - Specification of Service Inputs & Outputs

Service	Control Input	Data Input	Data Output	Status Output
		RFIPS Services		
Initialize				success/fail
Authenticate		auth token		success/fail
Show status				error or OK
Zeroize		plaintext data		
Zeroize file		plaintext data		
		OpenSSH Services		
Get SSH interface	auth token			success/fail
Connect	connection parameters			success/fail
Disconnect				success/fail
Send Data		plaintext data	ciphertext data	error on fail
Receive Data		ciphertext data	plaintext data	error on fail
Generate RSA/DSA key pair	key type	filename number bits	public key private key (not output from physical boundary)	success/fail
SFTP commands		plaintext/ciphertext	ciphertext/plaintext	success/fail
		OpenSSL Services		
Get SSL interface	auth token			error on fail
Configure Session	Session parameters			success/fail
StartSecurity		plaintext data		success/fail
Disconnect		plaintext data		success/fail
Send Data		plaintext data	ciphertext data	error on fail
Receive Data		ciphertext data	plaintext data	error on fail
Get Session Data		plaintext data	plaintext data	success/fail
Free interface		plaintext data		success/fail
		Kerberos Services ***Note: The following interfaces provide operator authentication only.***		
Encrypt		plaintext	ciphertext	success/fail
Encode and Encrypt		structured plaintext	ciphertext	success/fail
Decrypt		ciphertext	plaintext	success/fail
Decrypt and Decode		ciphertext	structured plaintext	success/fail
Calculate CheckSum		plaintext/ciphertext	checksum	success/fail
Verify CheckSum		plaintext/ciphertext, checksum		success/fail
String To Key	key type	plaintext		success/fail
Generate Key	key type, key ID (seed)			success/fail
Is Valid Key	key ID			success/fail
Add Key Variance	key ID			success/fail
Is Valid Encryption Type	ID			success/fail
Random Number	number bytes		random bytes	success/fail
Copy Key	key ID			success/fail

Service	Control Input	Data Input	Data Output	Status Output
Cleanup Key	key ID			success/fail
Get File Cache				success/fail
Get Memory Cache				success/fail
Open	cache name			success/fail
Close	handle			success/fail
Destroy	cache name			success/fail
Exists	cache name			success/fail
Seek	handle, number bytes			success/fail
Read	handle, number bytes	plaintext/ciphertext	plaintext/ciphertext	success/fail
ReadKey	handle, key type	plaintext		success/fail
Write	handle	plaintext/ciphertext	plaintext/ciphertext	success/fail
WriteKey	handle, key ID		plaintext	success/fail

Definition of Critical Security Parameters (CSPs)

The following are CSPs contained in the module:

- Traffic Encryption Key (TEK): This is a TDES or AES key used to encrypt or digitally sign data. There are four different TEK's used in OpenSSH (inbound decryption, outbound encryption, inbound signature, outbound signature). All of theses TEK's are separate and different from each other, but are derived from the same Diffie-Helman shared secret. For OpenSSL, there are the same four TEK types. These two are separate and different from each other, and are derived from the TLS master secret computed during the TLS handshake process.
- Kerberos Traffic Encryption Key (KTEK): This is a DES or TDES key used to encrypt a message.
- User's Private Key: This is the private part of a user's DSA or RSA Public/Private key pair. It may be generated by the module and must be at least 1024 bits. It is used to generate DSA or RSA signatures.
- Authentication Bitstring: This is 128 bit hexadecimal string used to authenticate a user.
- User's Password: This is a password that is entered by the user via the keyboard.
- DH Private Exponent : The client's private exponent that is used to help compute the DH Shared Secret. This value is obtained by using the modules PRNG.
- DH Shared Secret: This is the secret shared between the client and the server once the Diffie-Helman key establishment protocol is finished.
- ANSI X9.31 PRNG internal state: This is the set of internal information related to the modules PRNG.

Definition of Public Keys:

The following are the public and private keys contained or input in the module:

- User's Public Key: This is the public part of a user's DSA or RSA Public/Private key pair. It is generated by the module for use by other entities.
- Server's Public Key: This the public part of a server's RSA or DSA Public/Private key pair used to verify the server's public key signature.

- Diffie-Helman Public Key: This is the public part of a Diffie-Helman key exchange used for establishing the shared secret key.

Definition of CSPs Modes of Access

Table 7 defines the relationship between access to CSPs and the different module services. The modes of access shown in the table are defined as follows:

- Get Authentication Bitstring: This operation reads in the authentication bitstring (Auth token in table 7) from a plaintext file.
- Receive Server's Public Key: This operation reads in the server's public key (RSA or DSA) from the network and is later used to verify the server's signature.
- Generate DH Private Exponent : This operation uses the modules DRNG to obtain an appropriate exponent that can be used in the Diffie-Helman key establishment protocol.
- Establish DH Secret: This operation uses the private exponent and other publicly known parameters to establish the Diffie-Helman shared secret with a remote server.
- Generate TEK: This operation generates the Traffic Encryption Key's for data that is to be transmitted and received through out the session or until a new key is generated. There are four different TEK's used in OpenSSH (inbound decryption, outbound encryption, inbound signature, outbound signature). All of theses TEK's are separate and different from each other, but are derived from the same Diffie-Helman shared secret.
- Generate IV: This operation generates the Initialization Vector that will be used to initialize the encryption algorithm prior to encrypting data.
- Select User's Private Key: This operation reads in the user's private key (RSA or DSA) that is to be used to compute the signature of some data as a means to authenticate the user.
- Get Password: This operation prompts the user for a password, which is entered via the keyboard.
- Encrypt Password: This operation use a TEK to encrypt the users ssh password which is to be sent to the server.
- Destroy TEK: This operation erases the Traffic Encryption Key that was used to encrypt data throughout the session. This operation is performed at the end of the session or when a new TEK has been established.
- Get TEK: This operation accesses the TEK from RAM. It is used to either decrypt or encrypt data.
- Generate User's Public/Private Key Pair: This operation generates an RSA or DSA public/private key pair of at least 1024 bits. The private key is stored on disk and can be used to compute authentication data.
- Store User's Public/Private Key Pair: This operation stores a public/private key pair on disk.
- Derive KTEK: This operation uses a password to derive a DES or TDES Kerberos Traffic Encryption Key, which in turn is used to encrypt the initial kerberos authentication message.
- Generate KTEK: This operation generates a Kerberos Traffic Encryption Key to be used to encrypt data.
- Add Variance to KTEK: This operation adds key variance to a Kerberos Traffic Encryption Key by xoring a known constant to the key.
- Wrap KTEK: This operation encrypts a Kerberos Traffic Encryption Key using an previously established KTEK.
- Unwrap KTEK: This operation decrypts the Wrapped KTEK that was received with the incoming

message. A previously established KTEK is used to do this.

- Get KTEK: This operation gets the KTEK associated with a specified ID and returns the key data.
- Set KTEK: This operation sets the KTEK to be used to encrypt a message.
- Destroy KTEK: This operation deletes the KTEK used to encrypt a message.
- Copy KTEK: This operation makes a duplicate copy of a KTEK.
- Read KTEK: This operation reads in a plaintext KTEK from the disk or shared memory.
- Write KTEK: This operation writes out a plaintext KTEK to disk or shared memory.
- Zeroize TEK : This operation zeroizes all TEKs stored in. In addition to the zeroization API call, the Crypto Officer must manually delete the associated files described in Crypto Officer Guide document.
- Zeroize KTEK: This operation zeroizes all KTEKs stored in memory.
- Zeroize Cache: This operation zeroizes all file based KTEKs stored on the computer's file system.
- Access ANSI X9.31 PRNG : This opertation accesses internal information related to the PRNG once it has been intialized.

Table 7 - CSP Access Rights within Roles & Services

Service	Cryptographic Keys and CSPs Access Operation
RFIPS Services	
Zeroize	Zeroize TEK Zeroize KTEK Zeroize Cache
Zeroize File	Zeroize Cache
OpenSSH Services	
Get SSH interface	Get Auth Token
Connect	Receive Servers Public Key Generate DH Private Exponent Access ANSI X9.31 PRNG Establish DH Shared Secret Generate TEK's Access ANSI X9.31 PRNG Generate IV's Access ANSI X9.31 PRNG Select User's Private Key Get Password Get Kerberos Authentication Data
Disconnect	Destroy TEK's
Send Data	Get TEK
Receive Data	Get TEK
Generate RSA/DSA key pair	Generate User's Public/Private Key Pair Access ANSI X9.31 PRNG Store User's Public/Private Key Pair
SFTP commands	Get TEK
OpenSSL Services	
Get SSL interface	Get Auth Token
Configure Session	

Service	Cryptographic Keys and CSPs Access Operation
StartSecurity	Receive Servers Public Key Generate DH Private Exponent Access ANSI X9.31 PRNG Establish DH Shared Secret Generate TEK's Access ANSI X9.31 PRNG Generate IV's Access ANSI X9.31 PRNG Select User's Private Key
Disconnect	Destroy TEK's
Send Data	Get TEK
Receive Data	Get TEK
Get Session Data	
Free interface	Destroy TEK's
Kerberos Services *Note: The following are interfaces provide operator authentication only.*	
Encrypt	Set KTEK Destroy KTEK
Encode and Encrypt	Set KTEK Wrap KTEK Destroy KTEK
Decrypt	Set KTEK Destroy KTEK
Decrypt and Decode	Set KTEK Unwrap KTEK Destroy KTEK
Calculate CheckSum	Set KTEK Destroy KTEK
Verify CheckSum	Set KTEK Destroy KTEK
String To Key	Derive KTEK
Generate Key	Generate KTEK
Is Valid Key	Get KTEK
Add Key Variance	Get KTEK Add Key Variance to KTEK
Is Valid Encryption type	
Random Number	
Copy Key	Copy KTEK
Cleanup Key	Destroy KTEK
Get File Cache	
Get Memory Cache	
Open	
Close	
Destroy	
Exists	
Seek	
Read	
ReadKey	Read KTEK
Write	
WriteKey	Write KTEK

7. Operational Environment

The cryptographic module is designed to run as DLLs (Dynamic Link Libraries) on Microsoft Windows NT kernel x86 based systems (Windows 2000 and XP) in Single User Mode. Multiple concurrent operators are not supported. The module was tested on a Windows 2000 Professional workstation with SP3 and Q326886 Hotfix, configured as specified in EAL-4 Augmented Common Criteria report (CCEVS-VR-02-0025).

8. Security Rules

The example cryptographic module's design corresponds to the example cryptographic module's security rules. This section documents the security rules enforced by the cryptographic module to implement the security requirements of this FIPS 140-2 Level 1 module.

1. The cryptographic module shall provide one operator role. This is the Authenticated User role.
2. When the module has not been placed in a valid role, the operator shall not have access to any cryptographic services.
3. The cryptographic module shall encrypt session communications traffic using either 3DES or AES algorithms. DES algorithms are included for legacy use only.
4. The cryptographic module shall perform the following tests:

   A. Power up Self-Tests:

      1. Software Integrity Test (HMAC SHA-1 hash verification)
      2. Cryptographic algorithm tests:

      Note: OpenSSL and OpenSSH use a separate, but identical copy of OpenSSL's crypto library. Kerberos has a unique and separate implementation. Each module performs self-tests for algorithms it has implemented and uses in approved mode. The following list shows the algorithms that each module implements as well as the self-test it performs.

         OpenSSH and OpenSSL

         1. 3DES Known Answer Test
         2. AES Known Answer Test
         3. HMAC-SHA-1 Known Answer Test
         4. SHA-1 Known Answer Test
         5. RSA Known Answer Test
         6. DSA Known Answer Test

         Kerberos

         1. 3DES Known Answer Test
         2. DES Known Answer Test
         3. HMAC-SHA-1 Known Answer Test
         4. SHA-1 Known Answer Test
         5. MD5 Known Answer Test
         6. MD4 Known Answer Test
         7. CRC Known Answer Test

B. Conditional Self-Tests:

OpenSSH and OpenSSL

1. Continuous Random Number Generator (RNG) test – performed on OpenSSH and OpenSSL DRNG
2. RSA pairwise consistency test
3. DSA pairwise consistency test

5. At any time the cryptographic module is in an idle state, the operator can command the module to perform the power-up self-test by calling the SelfTest service available with the rfips.dll.
6. Prior to each use, the internal RNG shall be tested using the conditional test specified in FIPS 140-2 §4.9.2.
7. Data output shall be inhibited during key generation, self-tests, zeroization, and error states.
9. Status information shall not contain CSPs or sensitive data that, if misused, could lead to a compromise of the module.

This section documents the security rules imposed by the vendor:

1. The module shall not support the update of the logical serial number (DLL file version).
2. The module shall not send or receive “application data” until the appropriate protocol handshake has succeeded.
3. The module does not support multiple concurrent operators.
4. The module shall enforce a timed access protection mechanism that supports at most 60 authentication attempts per minute.
5. The Kerberos client only ensures FIPS-approved authentication of the operator and does not provide FIPS-approved data confidentiality or integrity.
6. Kerberos administrators need to ensure that their password policy is defined with the following minimum characteristics, in order to ensure that strength of authentication meets FIPS 140-2 requirements:
    a. Minimum password length 6
    b. Must include at least one each of upper and lower case alpha, numeric and special characters.

9. Physical Security Policy

The Reflection security component is a cryptographic module that are implemented completely in software, hence the physical security section of FIPS 140-2 is not applicable. The tested platform is a Dell OptiPlex GX1 personal computer which meets applicable Federal Communication Commission (FCC) Electromagnetic Interference (EMI) and Electromagnetic Compatibility (EMC) requirements for Class B home or business use as defined in Subpart B of FCC Part 15.

10. Mitigation of Other Attacks Policy

This section does not apply. The module was not designed to mitigate attacks outside of the scope of FIPS 140-2.

11. References

DES algorithm – "Data Encryption Standard (DES)", FIPS Pub 46-3 [October 25, 1999].
AES algorithm – "Advanced Encryption Standard (AES)", FIPS Pub 197 [November 26, 2001].
SHA algorithm – "Secure Hash Standard", FIPS Pub 180-2 [August 1, 2002].
DRNG algorithm – "Digital Signature Standard (DSS), FIPS pub 186-2 [January 27, 2000].
OpenSSH – "SSH Protocol Architecture", IETF SecSH WG Draft #15 [October, 2003].
"SSH Transport Layer Protocol", IETF SecSH WG Draft #15 [October, 2003].
"SSH Authentication Protocol", IETF SecSH WG Draft #18 [September, 2002].
"SSH Connection Protocol", IETF SecSH WG Draft #18 [October, 2003].
"SSH file Transfer Protocol", IETF SecSH WG Draft #5 [January, 2004].
"GSSAPI Authentication and Key Exchange for the Secure Shell Protocol", IETF SecSH WG Draft #7 [September 12, 2003].
Kerberos – "The Kerberos Network Authentication Service (V5)", IETF RFC 1510 [January, 1993].
"Telnet Authentication – Version 5", IETF RFC 2942 [September, 2000].
GSSAPI – "The Kerberos Version 5 GSS-API Mechanism", IETF RFC 1964 [June, 1996]
TLS – "The TLS Protocol, Version 1.0", IETF RFC 2246 [January, 1999]
"Advanced Encryption Standard (AES) Ciphersuites for Transport Layer Security", IETF RFC 3268 [June 2002]

12. Definitions and Acronyms

3DES (Triple DES) – The particular block cipher which is the U.S. Data Encryption Standard for DES, with two or three different keys. The 56-bit algorithm is used three times in sequence, usually encrypting with first 56-bit key, decrypting with the second 56-bit key and encrypting with the either a last or the first 56-bit key.

AES (Advanced Encryption Standard) – NIST recently selected encryption algorithm to use as the newest U.S. data encryption standard in FIPS 197. The algorithm selected is the Rijndael algorithm, which is a variable block and key-length algorithm. The algorithm was selected in a public competition from five finalists.

Approved Mode – The state of the cryptographic modules with allows only the use of FIPS approved algorithms and protocols

DES (Data Encryption Standard) – The particular block cipher which is the U.S. Data Encryption Standard as established by NIST in FIPS 46-1. A 64-bit block cipher with a 56-bit key organized as 16 rounds of operations. The DES algorithm has been published and extensively scrutinized for potential weaknesses. It has never been broken, but has been cracked by computerized brute force attack.

Diffie-Hellman key exchange – A method of establishing a shared key over an insecure medium, named after the inventors. The security of Diffie-Hellman relies on the difficulty of the discrete logarithm problem (which is believed to be computationally equivalent to factoring large integers). This algorithm is commonly accepted and commercially available.

DSA (Digital Signature Algorithm) – A digital signature algorithm as established by NIST in FIPS 186-2.

GSSAPI (Generic Security Service Application Programmers Interface) – a library and a set of C-binding routines that may be used for authentication, integrity checking and encryption as defined in IETF RFC 1964.

HMAC (Keyed-Hashed Message Authentication) – A mechanism for message authentication using cryptographic hash functions, as defined in IETF RFC 2104.

Kerberos – A DES-based authentication system developed at MIT as part of project Athena and subsequently incorporated into a growing collection of commercial products (including Microsoft's Windows 2000 and XP). Kerberos assumes that the systems themselves can be secured (KDC and application servers) but the network between them is insecure. It is specified in IETF RFCs 1510 and 2942.

MD-4™ – A proprietary (to RSA Data Security) message digest function that is a three-pass algorithm that produces a 128-bit digest.

MD-5™ – A proprietary (to RSA Data Security) message digest function that is a four-pass algorithm that produces a 128-bit digest.

PKCS #12 (Public-Key Cryptography System) – A document produced and distributed by RSA Data Security (a Security Dynamics company), proposing techniques for storing and transporting a user's private keys, certificates or other secrets in a safe and interoperable manner. Defined in IEEE P1363 and RSA's PKCS #12 specification.

DRNG (Deterministic Random Number Generator) – A standard computational tool which creates a sequence of apparently unrelated numbers used in cryptography for secret keys, but whose sequence of operations is fully determined by its initial state. Each result, and each next state, is directly determined by the previous state of the mechanism, all the way back to the original seed. Such a sequence is often called *pseudo*-random, to distinguish it from a really random sequence somehow composed of independent unrelated values. A FIPS-approved DRNG is defined In FIPS 186-2.

RC-4™ - Another proprietary (patented by RSA Data Security) secret key stream algorithm that effectively produces an unbounded length pseudorandom stream from a varying key length. Common key lengths are 40 and 128-bit. Named after its inventor, Ron Rivest, the acronym stands for Rivest's Cipher #4. This algorithm is the most commonly used one for secure web transactions. Arcfour is the claimed public domain equivalent of the RC-4 algorithm.

RSA™ – The name of an algorithm published by Ron Rivest, Adi Shamir, and Len Adleman (thus, R.S.A.). The first major public key system. Based on number-theoretic concepts and using huge numerical values, a RSA key must be perhaps ten times or more as long as a secret key for similar security. The RSA cryptographic algorithms are trademarked and patented, and so licensing and royalty fees must be paid to RSA Data Security (a Security Dynamics company) for use. Many of the US patents expired in the year 2000. Since the RSA algorithm is proprietary and has not been publicly scrutinized, it is unknown whether it can be broken or cracked, or if it might contain back-door access.

SHA-1 (Secure Hash Algorithm) – A 160-bit message digest function defined by the NIST in FIPS 180-1.

SSL (Secure Sockets Layer) – A secure communications protocol developed by Netscape to provide authentication via public key, message integrity checking via SHA or MD5 and encryption with RC4 or DES algorithms. Current version is 3.1.

ssh™ (Secure Shell) – A protocol that provides support for secure remote login, secure file transfer, and secure TCP/IP and X11 forwardings. It can automatically encrypt, authenticate, and compress transmitted data. SSH is developed by SSH Communications Security Ltd. in Finland and the name "ssh" is trademarked.

SSHv2, SSH2 – A protocol that is used to secure terminal sessions and arbitrary TCP-connections. SSH2-protocol is based on SSH1-protocol, developed by Tatu Ylönen. This is an evolving standard protocol being worked on by the Secure Shell working group of the IETF.

TLS (Transport Layer Security) – The new name for the next generation SSL protocol defined as a public standard in IETF RFC 2246. The protocol is composed of two layers: the TLS Record Protocol and the TLS Handshake Protocol. At the lowest level, layered on top of some reliable transport protocol (e.g., TCP), is the TLS Record Protocol which provides connection security. Above that the TLS Handshake Protocol, allows the server and client to authenticate each other and to negotiate an encryption algorithm and cryptographic keys before the application protocol transmits or receives its first byte of data. Current version is 1.0 and does not, by default, interoperate with SSL 3.x although a backward compatibility mechanism is incorporated into the standard.

Unapproved Mode – The state of the cryptographic modules with allows the use of non-FIPS approved algorithms and protocols.