CGX Cryptographic Module Version 3.21.1

Security Policy Level 2

Security Rule Specification

Approved Document

Revision: 3.3
Date: 13 March 2006
Status: Approved

SafeNet, Inc. maintains a website with up-to-date technical documentation for our customers. Contact SafeNet for access:

www.safenet-inc.com

For Further Information:

siliconsales@safenet-inc.com

SafeNet, Inc.
100 Conifer Hill Drive, Suite 513
Danvers, MA 01923
U.S.A.
Phone: (+1) 978-539-4800 Fax: (+1) 978 739-5698

SafeNet BV
The Netherlands
Boxtelseweg 26A
5261 NE Vught
P.O. Box 22
5260 AA Vught
The Netherlands
Phone: +31-73-6581900 Fax: +31-73-6581999

SafeNet, Inc.
Corporate Headquarters
8029 Corporate Drive
Baltimore MD, 21236
Phone: (410) 931-7500 Fax: (410) 931-7524

Revision History

Rev	Page(s)	Date	Author	Purpose of Revision
1.0	All	10/24/03	DP	Initial submission
2.0	All	11/22/04		Corrections
3.0	All	5/5/04		Corrections from comments
3.1	All	1/27/06		Corrections from comments
3.2	All	3/6/06		Corrections from comments
3.3	All	3/13/06		Corrections from comments

TABLE OF CONTENTS

1 Overview

The SafeNet CGX (**C**rypto**G**raphic e**X**tensions) library is a suite cryptographic functions, which are available to applications which require security services. The cryptographic module consists of 18 kernel functions (listed in Table C3), implemented entirely in software as the shared library object files libCgxSupport.so on Solaris 8. A list of Approved and non-Approved algorithms used by the CGX is given in Section 4.2 Security Rules in this document. The Solaris 8 operating system is configured to operate in single user mode.

2 CGX Functional Specifications

The SafeNet security software which is made available to applications running on the Solaris 8 platform is designated the SafeNet CGX Kernel. To simplify application-level access to crypto functions, an Application Programming Interface (API) is provided to the CGX Kernel. The CGX Command Interface defines the boundaries between the security functions (which the CGX Kernel implements) and the externally running applications. One of the primary goals of the CGX software is to abstract the CGX Kernel from the application in a secure and efficient manner. The CGX interface is designed so that it can be viewed as a Crypto Library with a C-structure like interface with argument and pointer-passing. To make a CGX command call a structure is populated with arguments and a call is made to the CGX kernel, passing a pointer to the structure.

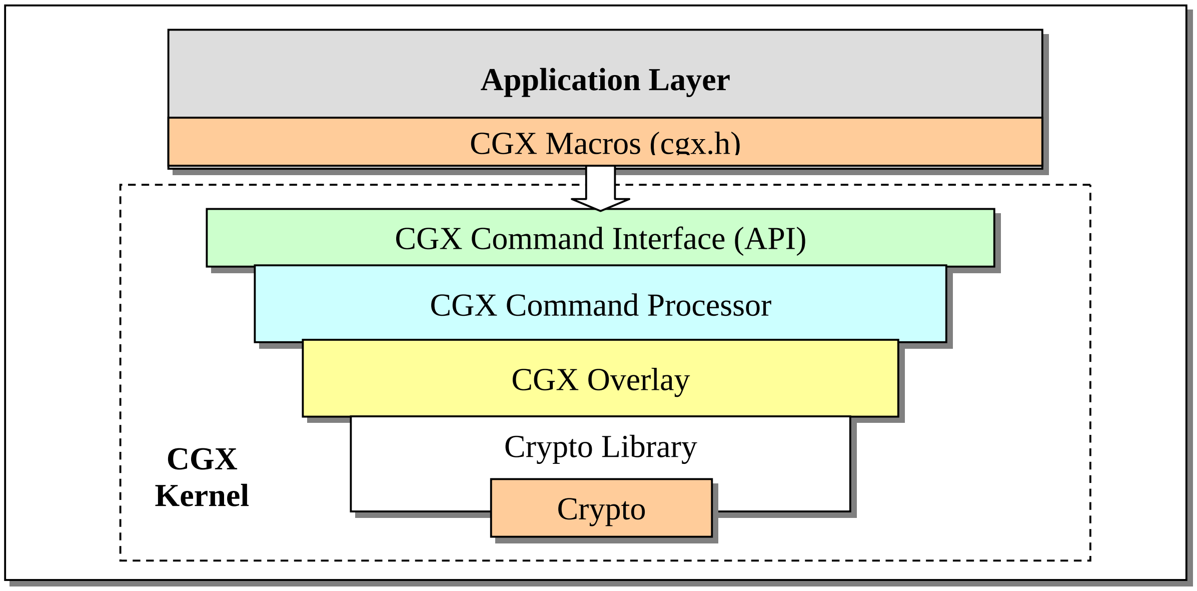

Figure 1 *CGX layers*

To execute a CGX command, a structure is populated with arguments and a 'C' language call is made to the CGX Kernel, passing a pointer to the structure. Alternatively, a macro from the header file cgx.h may be used with the appropriate arguments for the command. The macro for the command will insert the arguments into the control block structure and invoke the CGX Kernel.

The CGX software resides within the dashed line illustrated in Figure 1. The application uses the CGX Command Interface as an API to access the CGX command set. To better understand the software architecture of the CGX security software, a description of each layer is provided in the sections below.

2.1 APPLICATION LAYER

The application layer is where the actual application program and data space reside. The application can implement anything from a router security co-processor to a V.34 modem data pump, but for purposes of this FIPS 140-2 certification the application is assumed to be a software-only application.

In order to access the cryptographic services, the application must invoke the CGX command interface and supply a command code and arguments. It is likely that the application layer will include a '*CGX Processing Manager'* which accepts host-originated requests, formats them, and then issues the call to CGX for processing.

Residing as part of the application layer are the macro functions that SafeNet provides in its cgx.h and ecgx.h files. These optional macros assist the application in preparing the command messages prior to calling the CGX Kernel.

2.2 CGX COMMAND INTERFACE LAYER

The CGX command interface layer is an Application Programming Interface (API) that defines the boundaries between the application and the CGX Kernel. The CGX command interface provides the mechanism to enter and exit the CGX Kernel in order to execute a specific cryptographic command.

The software interface to the CGX Kernel is via the *kernel block* and the *command block*. The *kernel block* is a simple structure that specifies memory modes and provides a pointer to the *command block*, allowing flexible placement in memory. It also contains a status element that the application can read to determine the result of a requested cryptographic service. The *command block* is used to request a specific cryptographic command and to provide a means of supplying arguments.

Therefore, all communications between the application and the CGX Kernel is via the command interface and a *kernel block* and *command block*. The command interface is discussed in more detail in Chapter 3.

2.3 CGX COMMAND PROCESSOR LAYER

The CGX Command Processor implements a secure Operating System responsible for processing application requests for various cryptographic services. Once the CGX Kernel is active, it can process the requested cryptographic function specified in the *kernel block* & *command block* defined as part of the CGX command interface layer.

2.4 CGX OVERLAY LAYER

The CGX overlay layer is provided as the interface into SafeNet's CryptoLIB software. The CryptoLIB software is a library that is designed for multiple platforms, ranging from the PC to embedded systems. The CGX overlay acts as the 'wrap code' to enable the library to execute on a platform unmodified.

Figure 2 illustrates the data flow through the CGX overlay layer.

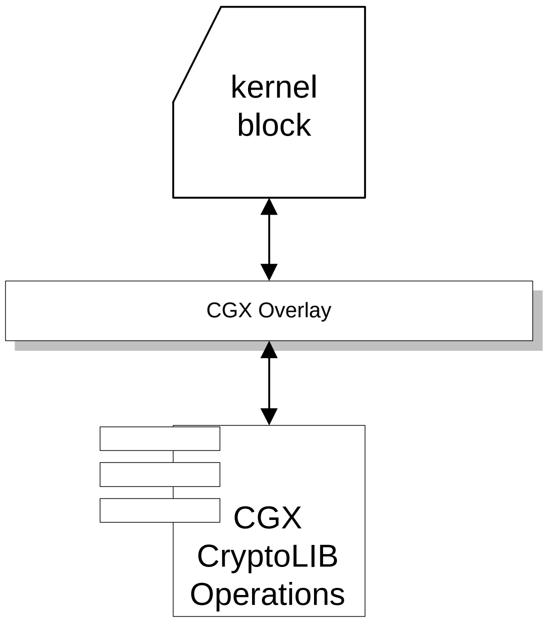

Figure 2 *CGX overlay interface*

When a cryptographic request is received, the CGX Command Processor parses the *kernel block* to determine the cryptographic command to execute. The CGX Command Processor executes a CGX overlay operation from a table, based on the command value embedded in the *command block* portion of the *kernel block*. The CGX overlay operation is responsible for extracting the arguments from the *kernel block* and invoking the proper CryptoLIB operations. In some cases, the CGX overlay operation may invoke several CryptoLIB operations. In effect, this is an object-oriented approach where the CGX overlay class is the parent class to the CryptoLIB classes.

2.5 CRYPTOLIB LAYER

The CryptoLIB layer contains SafeNet's Crypto Library software. The CryptoLIB software is a library of many cryptographic classes implementing various cryptographic algorithms including symmetrical encryption algorithms, one-way hash functions, and public key operations.

2.6 CRYPTOGRAPHIC BOUNDARY

Figure 3 illustrates the cryptographic boundary of the CGX module, interconnections among major components of the CGX module and between the CGX module and equipment or components outside of the cryptographic boundary. For the purpose of completeness, the Hardware Provider Interface (HPI) is included inside of the cryptographic boundary; however, since it's purpose is to provide communication between the CGX kernel and an installed cryptographic accelerator chip, for the purposes of this certification the only function performed by the HPI is to confirm the absence of hardware acceleration.

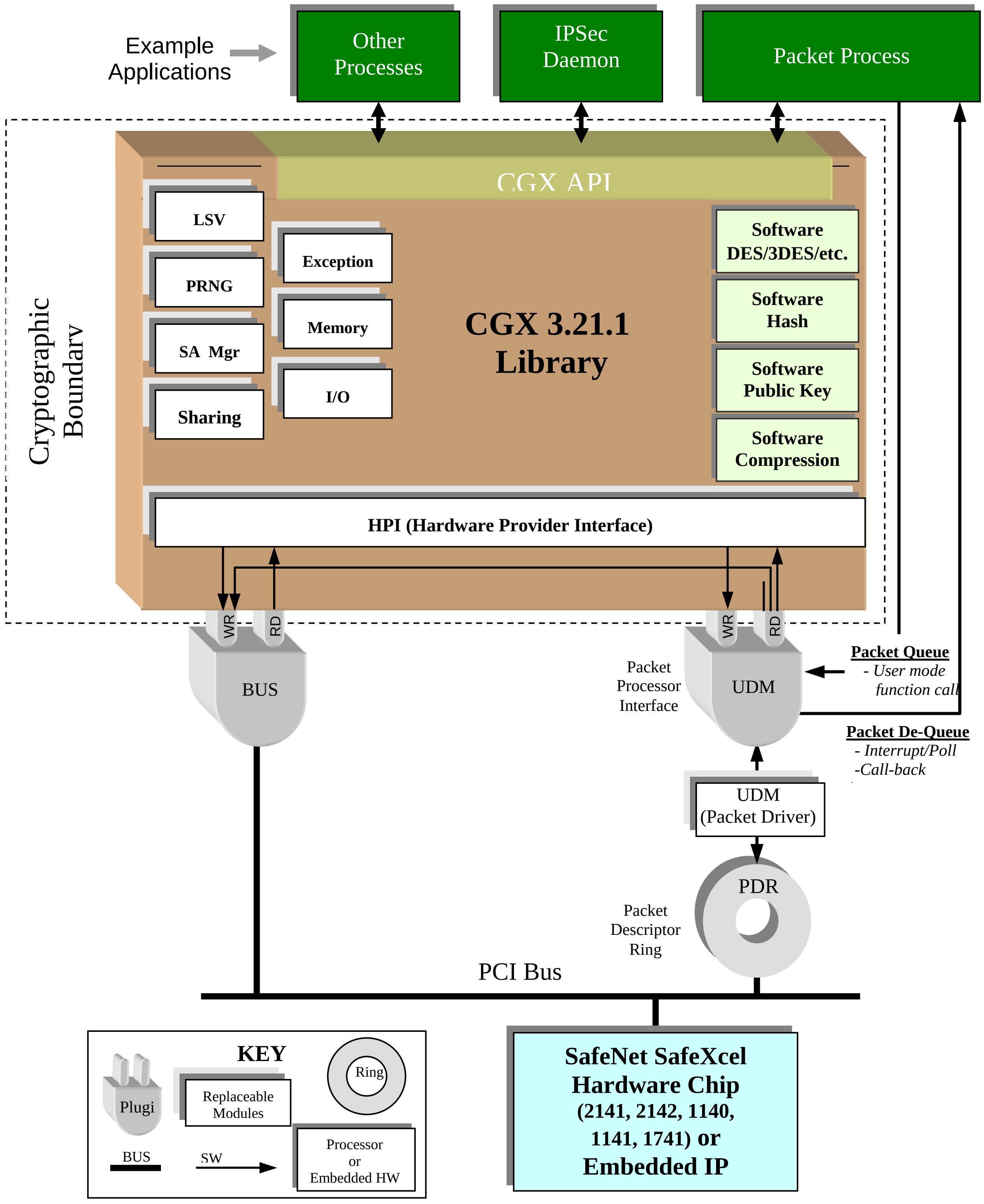

Figure 3 *Definition of Cryptographic Boundary*

3 Security Level

The cryptographic module meets the overall requirements applicable to Level 2 security of FIPS 140-2 when running on any on a SunBLADE 2000 Workstation running Solaris 8 2/02 operating system.
.

Table 1. *Module Security Level 2 Specification*

Security Requirements Section	Level
Cryptographic Module Specification	2
Module Ports and Interfaces	2
Roles, Services and Authentication	2
Finite State Machine	2
Physical Security	2
Operational Environment	2
Cryptographic Key Management	2
EMI/EMC	3
Self Tests	2
Design Assurance	3
Mitigation of Other Attacks	2

Solaris 8 2/02 has been Common Criteria Certified with certificate number P182 dated April 2003.
http://www.cesg.gov.uk/site/iacs/itsec/media/certreps/CRP182.pdf

4 Roles and Services

The CGX cryptographic module shall support two distinct operator roles. These operator roles are:

1. User Role (CryptoUser)
2. Cryptographic Officer Role (CryptoOfficer)

When running on Solaris 8 2/02, the cryptographic module enforces role-based operator authentication. Solaris 8 provides authenticated login which it enforces on the operator and uses Sun Enterprise Authentication Method (SEAM), which is an implementation of the Kerberos v5 protocols for the Solaris Operating Environment with some additional capabilities.

Role	**Type of Authentication**	**Authentication Data**
User Role	Logon	Password
Cryptographic Officer Role	Logon	Password

Table C1. *Roles and Required Identification and Authentication*

Operating System	**Mechanism**	**Strength of Mechanism**
Solaris 8 2/02	Sun Enterprise Authentication Method (SEAM)	DES 56 Bit

Table C2. *Strengths of Authentication Mechanisms*

The following table lists the CGX kernel commands and their applicable roles.

CGX Command	Crypto Officer Role	User Role
`CGX_INIT`	X	
`CGX_DEFAULT`	X	
`CGX_RANDOM`	X	
`CGX_ZEROIZE_KEYS`	X	
Symmetric Key Commands		
`CGX_UNCOVER_KEY`		X
`CGX_GEN_KEK`		X
`CGX_GEN_KEY`		X
`CGX_LOAD_KEY`		X
`CGX_DERIVE_KEY`		X
`CGX_ENCRYPT`		X
`CGX_DECRYPT`		X
Asymmetric Key Commands		
`CGX_GEN_PUBKEY`		X
`CGX_PUBKEY_ENCRYPT`		X
`CGX_PUBKEY_DECRYPT`		X
`CGX_IMPORT_PUBKEY`		X
Hash Commands		
`CGX_HASH_INIT`		X
`CGX_HASH_DATA`		X
Math Commands		
`CGX_MATH`		X

Table C3. *Services Authorized for Roles*

4.1 CGX Kernel Command Descriptions

The CGX module interface in it's simplest form consists of the kernel block, the command block and a single module interface, **cgx_transfer_secure_kernel**. Based on how the kernel and command blocks are populated when **cgx_transfer_secure_kernel** is invoked, the CGX kernel will perform the requested function and return output as appropriate. To aid the programmer in accessing CGX functionality, the CGX library is packaged with a set of macros that will populate the command and kernel blocks appropriately and call **cgx_transfer_secure_kernel**. These macros can be used for direct access to the shared library libCgxSupport.so.

The following is a brief description of the functionality available within the CGX module, referenced by the SafeNet-defined macro names.

CGX _INIT initializes the CGX Kernel, runs a set of basic self-tests, and allows the caller to configure two classes of configuration settings:

- Increase the default number of Key Cache Registers (from 15 up to 700)
- Specify various configuration parameters associated with the CGX Kernel (via the Kernel Configuration String)

CGX _DEFAULT initializes the CGX Kernel, and restores application-definable settings to factory defaults. This command is typically used to reset any customized settings which may have previously been selected using CGX _INIT.

CGX _RANDOM gets bytes of random data from the pseudo random number generator.

CGX _ZEROIZE_KEYS is used to delete all of the KCRs including the LSV from KCR 0. Furthermore, it exits from the CGX library.

CGX _GEN_KEK generates an internal key encryption key using the CGX's pseudo random number generator and places it into the specified Key Cache Register.

CGX _GEN_KEY generates a symmetrical key using the CGX's pseudo random number generator and places it into the specified Key Cache Register. Optionally, the newly generated key may be returned to the caller in a Black (DES or TDES encrypted) form. The random key bits are transformed into the secret key form as directed by the type of secret key specified in the argument interface.

CGX _DERIVE_KEY (non-FIPS compliant) allows a user secret key to be created from an application's pass-phrase. The secret key is derived by taking the one-way Hash of the application's pass phrase and using the resulting message digest as the secret key bits. The 'raw' message digest bits are transformed into the secret key form as directed by the type of secret key specified (i.e. key_type) in the argument interface and placed into the specified Key Cache Register.

CGX _UNCOVER_KEY decrypts the Black secretkey, bk, to a Red form and places it into the key cache register (KCR) indicated by the input argument, destkey. A Black secret key is defined as a key stored in SafeNet internal format (which has therefore been encrypted and authenticated with a keyed hash). This allows an application to securely store Black secret keys outside of CGX for later use by the CGX kernel.

CGX _LOAD_KEY is used to load a plaintext user secret key into a specified Key Cache Register. The secret key to be loaded is in the Red form. Depending on the value of the use argument, the key can be used as either a KEK or as a DEK. This key is known as a user key to the CGX Kernel and can never be covered by the LSV (the CGX Kernel does not allow it).

CGX _ENCRYPT is used to perform the symmetrical encryption of plain-text data and return the cipher-text to the application in the specified buffer.

CGX _DECRYPT is used to perform the symmetrical decryption of cipher-text data and returning the plain-text to the application in the specified buffer.

CGX _HASH_INIT (Initialize Hash) is used to initialize a Hash context block (data structure type hash_cntxt.) The command is used in preparation for a Hash function computation. After initialization, the Hash context block may subsequently be used as a parameter to a sequence of one or more CGX operations, such as CGX _HASH_DATA, which perform the Hash computation. At any given time, an application may have several separate independent hash computations in various stages of completion. Each hash computation will have its own dedicated hash context; each context contains the current state information of its corresponding Hash computation. The computation types supported are SHA-1 and MD5 one way Hash algorithms. Both Hash algorithms have a limit of $2^{64} – 1$ bits cumulative input data length. Upon completion of this operation, the hash context will contain a NULL value in the digest member of the hash_cntxt object (since the hash isn't 'closed'). When the hash computation is completed and the context is closed, the digest member will contain a valid hash digest: i.e., the result of the hash computation.

CGX _HASH_DATA (Hash Data) is used to calculate a Hash value over data supplied by the calling application. The hash value is computed over a stream of data octets (8-bit data bytes) which optionally may begin with a key whose octets are treated as data to be hashed (thus creating a 'keyed hash'), then may include a virtually unlimited number of non-key data octets and optionally concludes with a trailing key whose octets are also treated as data to be hashed. If both leading and trailing data keys are included in the hashed data stream, they may be the same or different. For security reasons, a key may not be inserted into the middle of the data being hashed.

CGX _GEN_PUBKEY will generate an entire public keyset comprised of the modulus, private, and public blocks. This operation can create public keysets for several public key algorithms. This interface is over-loaded and currently supports Diffie-Hellman, and RSA, public keys. The returned keyset will consist of data stored in little endian order.

CGX _IMPORT_PUBKEY allows the application to move an external public key back into CGX in the SafeNet public keyset form. The external form must be covered either with a secret key or public key, this is specified by the application via the command arguments.

CGX _PUBKEY_ENCRYPT is used to encrypt the application's data using the RSA encryption algorithms. This operation also may be used to perform RSA signature verification using the public key component of a public keyset.

CGX _PUBKEY_DECRYPT is used to decrypt the application's data using the RSA encryption algorithms. This operation also may be used to perform RSA signing using the private key of a public keyset.

CGX _MATH is a set of cgx commands that perform various mathematical functions.

4.2 Security Rules

This section documents the security rules enforced by the cryptographic module to implement the security requirements for the FIPS 140-2 Level 2 module except as noted.

1. The cryptographic module shall provide two distinct operator roles by virtue of the type of operation being performed. These are the User Role, and the Cryptographic Officer Role.

2. The cryptographic Module provides role-based authentication via the Solaris 8 2/02 logon mechanism**.**

3. When the module has not been properly initialized, the operator shall not have access to any cryptographic services and CGX will remain in the Error State.

4. Upon the application of power or when commanded by the operator, the cryptographic module shall perform the following tests:
   Triple DES Encryption/Decryption Algorithm Known Answer Test
   DES Encryption/Decryption Algorithm Known Answer Test
   SHA-1 Algorithm Known Answer Test
   AES Encryption/Decryption Known Answer Test
   HMAC Known Answer Test
   Pseudo Random Number Generator Known Statistical Test
   Pseudo Random Number Generator Known Answer Test
   RSA Known Answer
   Diffie-Hellman Known Answer Test
   Software/firmware integrity check

   Conditional Tests
   RSA Pairwise consistency test

5. At any time the module is in an idle state, the operator shall be capable of commanding the module to perform the power-up self test.

6. CGX utilizes the following cryptographic and hashing algorithms:
   FIPS-Approved:

Cert#393	TDES	FIPS 46-3
	Electronic Code Book (ECB)	
	Cipher Block Chaining (CBC)	
	64-bit Output Feedback (OFB)	
	64-bit Cipher Feedback (CFB)	
Cert#329	AES	FIPS 197
	128, 192 and 256 bit keys	
	Electronic Code Book (ECB)	
	Cipher Block Chaining (CBC)	
Cert#148	HMAC	FIPS 198
Cert #49	RNG	X9.31 A.2.4

   Non FIPS-Approved
   DES (non-compliant)
   RC5

RSA encrypt/decrypt
signatures (non-compliant)
Diffie-Hellman (Key agreement; key establishment methodology provides between 56 and 150 bits of encryption strength)
MD2
MD5
RIPEMD-128
RIPEMD-160

7. Prior to each use, the internal Random Number Generator shall be tested using the Conditional test specified in FIPS 140-2 §4.9.1.

8. The CGX cryptographic module must always be properly initialized prior to it being used. If an operator attempts to execute a CGX command without first executing the CGX_Init command, then CGX will automatically execute CXG_INIT on its own prior to processing the requested command.

9. Unencrypted (Red) keys can never be returned by CGX. All keys passed back to the caller are always encrypted under a higher level KEK

10. Applications utilizing the CGX cryptographic module must conform to the requirements in FIPS 140-2. It is the responsibility of the application not of the CGX cryptographic module to handle red key entry.

11. The CGX cryptographic module was written in the C high level language.

12. To operate the CGX module in FIPS mode, the following must be observed:

    - Only the functions listed in section 4.1, CGX Kernel Command Descriptions should be used.
    - Only the algorithms listed above as FIPS-approved should be used.
    - The initialization block must be configured with an external HostKcrCache for 255 KCRs, and a SA_Configuration setting of 1, SA_Entries of 0 and SA_Addr set to NULL.
    - The Kernel Configuration String must contain the following settings:

    ```
    flipsha = CGX_FLIP_SHA1_FINAL
    flipgxy = TRUE
    fips140_1 = TRUE
    fips_enable = (CGX_FIPS_ENABLE_ALL_LOWER &
    ~CGX_FIPS_BIST_ENABLE_PRAM) |
    CGX_FIPS_ENABLE_STRONG_PRNG |
    CGX_FIPS_ENABLE_REDKEY |
    CGX_FIPS_ENABLE_PAIRWISE |
    CGX_FIPS_LOAD_ALTERNATE_LSV
    fips_enable_upper = CGX_FIPS_ENABLE_ALL_UPPER
    ```

4.3 Definition of Security Relevant Data Items (FIPS and Non-FIPS)

4.3.1 SYMMETRIC KEYS

- Data Encryption Key (DEK): This is a DES, Triple-DES or AES key used to encrypt user traffic.
- Key Encryption Key (KEK): This is a DES, Triple-DES or AES key used only to encrypt other keys.
- Generator Key Encryption Key (GKEK): This is a special Triple-DES key used only to encrypt other keys, and is itself protected by the Local Storage Variable (LSV).
- Local Storage Variable (LSV): This is a unique Triple-DES key used as the root key to recover other keys after a power outage. The LSV is always loaded into Key Cache Register #0. It cannot be exported from CGX.
- HMAC Key: This is the key used in HMAC-SHA-1.

4.3.2 ASYMMETRIC KEYS

- Public Key: This is the public component of an RSA or Diffie-Hellman key pair.
- Private Key: This is the private component of an RSA or Diffie-Hellman key pair.

4.3.3 OTHER OBJECTS

- Initialization Vector (IV): This is a 64 bit random number used to initialize the DES, TDES and AES encryption algorithms. Each algorithm is initialized with a unique IV, supplied by the application or from the PRNG, for each message encrypted.
- Kernel Configuration String (KCS): This is a configuration string that sets-up certain features of the CGX kernel during the Initialization process. Two of the relevant configuration options are:
  - Enable FIPS 140-2 compliant RNG (certificate # 49). This parameter turns on the ANSI X9.31 A.2.4 RNG. This feature <u>must</u> be enabled for the FIPS 140-2 compliant version of CGX. The random number entropy source is application dependent. For this application, it is received from the /dev/random system device.
  - Allows the Crypto Officer to enable/disable red key parity checking from the Kernel Configuration String.
- Key Attribute Bits: This is a bit-mapped field which is attached to any key and specifies its Type and Use. The key type specifies whether the key is a DEK, KEK, etc.
- Key Cache Register (KCR): This is a volatile key storage house for a fixed number of secret keys. The volatile key area is also referred to as the actively working keys. All cryptographic commands operate only on the active volatile working keys.
- Authentication Data: SEAM and password.

4.4 Service to SRDI Access Operation

	SRDI									
User Service	DEK	KEK	GKEK	LSV	PublicKey Public Component	PublicKeyPrivate Component	IV	KCS	KeyAttribute Bits	KCR
CGX_INIT				R				R		W
CGX_DEFAULT				R				R		W
CGX_RANDOM										
CGX_ZEROIZE_KEYS										D
Symmetric Key Commands										
CGX_UNCOVER_KEY	RW	RW	RW				RW		RW	RW
CGX_GEN_KEK		RW		R			RW		RW	RW
CGX_GEN_KEY	RW	RW	R				RW		RW	RW
CGX_LOAD_KEY	RW	RW	R				RW		RW	RW
CGX_DERIVE_KEY	RW	RW	R				RW		RW	RW
CGX_ENCRYPT	R						RW		R	R
CGX_DECRYPT	R						RW		R	R
Asymmetric Key Commands										
CGX_GEN_PUBKEY		R	R		W	RW	R		R	R
CGX_PUBKEY_ENCRYPT		R			R	R	R		R	R
CGX_PUBKEY_DECRYPT		R			R	R	R		R	R
CGX_IMPORT_PUBKEY		R				M	R		R	R
Hash Commands										
CGX_HASH_INIT										
CGX_HASH_DATA										
Math Commands										
CGX_MATH										

R = Read W = Write M = Modify D = Delete

Table C4. *Access Rights within Services*

5 Physical Security

Physical security on a SunBLADE 2000 Workstation running Solaris 8 2/02 is achieved through the use of Tamper Evident Seals to create a tamper-proof enclosure. The four tamper evident are located on the seams of the removable cover and must be in place to meet the level 2 requirement. Additional security can be provided by using a padlock.

Physical Security Mechanisms	**Recommended Frequency of Inspection/Test**	**Inspection/Test Guidance Details**
Tamper Evident Seals	Daily	By Observation

Table C4. *Table of Physical Security Mechanisms*

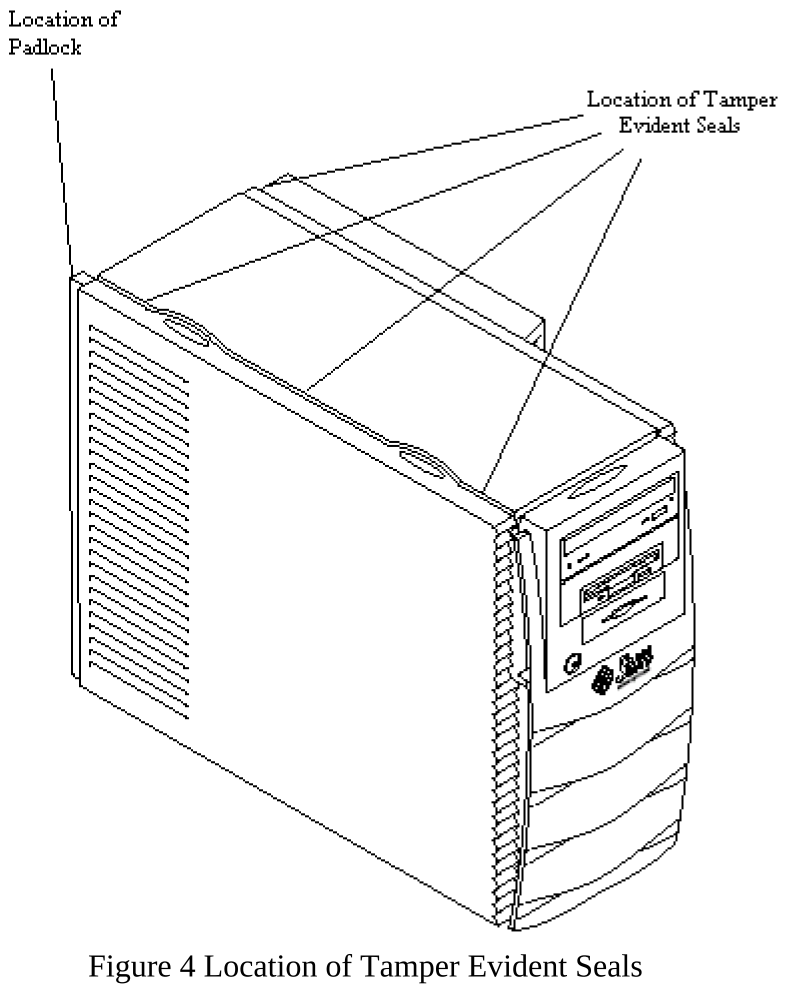

Figure 4 Location of Tamper Evident Seals

6 Mitigation of Other Attacks

Other Attacks	Mitigation Mechanism	Specific Limitations
Kocher Timing Analysis Attack	Check for attack during Diffie-Hellman key generation.	None.

The Kocher Timing Analysis Attack is a timing attack theory developed by Paul Kocher, President of Cryptography Research Inc. The basis of the theory is that by carefully measuring the amount of time required to perform private key generation, attackers may be able to find the fixed Diffie-Hellman exponents, factor RSA keys, and break other cryptosystems. CGX mitigates this type of attack by always performing the multiply in the inner square loop of exponentiation which results in constant time, and has implemented RSA blinding for RSA decryption.

Appendix A List of Acronyms

Acronym	Description
AES	Advanced Encryption Standard
API	Application Programming Interface
CGX	**C**rypto**G**raphic e**X**tensions
DEK	Data Encryption Key
DES	Data Encryption Standard
D-H	Diffie-Hellman
EMC	Electromagnetic Compatibility
EMI	Electromagnetic Interference
FIPS	Federal Information Processing Standard
GKEK	Generator Key Encryption Key
HMAC	Hash Message Authentication Code
IPsec	Internet Protocol Security
ITSEC	Information Technology Security Evaluation Criteria
IV	Initialization Vector
KAT	Known Answer test
KCR	Key Cache Register
KCS	Kernel Configuration String
KEK	Key Encryption Key
KG	Key Generator
LSV	Local Storage Variable
MD5	Message Digest 5
OS	Operating System
PC	Personal Computer
PCDB	Program Control Data Bit
PRAM	Program Random Access Memory
PRF	Pseudo Random Function
PRNG	PseudoRandom Number Generator
RAM	Random Access Memory
RNG	Random Number Generator
RSA	Rivest Shamir Adleman
SEAM	Solaris Enterprise Authentication Method
SRDI	Security Relevant Data Items
SHA-1	Secure Hash Algorithm