CCORE Module FIPS 140-2 Level 1 Security Policy

Document Version: 4.1
Date: April 13, 2010

Prepared for: CellCrypt Limited

Revision History

Version	***Date***	***Author***
0.1	07-11-2008	C. Davies (Cell Telecom Ltd.)
0.2	07-27-2008	C. Davies (Cell Telecom Ltd.)
0.3	08-12-2008	C. Davies (Cell Telecom Ltd.)
0.4	11-12-2008	C. Davies (Cell Telecom Ltd.)
1.0	11-12-2008	C. Davies (Cell Telecom Ltd.)
2.0	19-12-2008	C. Davies (Cell Telecom Ltd.)
3.0	22-12-2008	Saffire Systems
3.1	12-01-2009	Saffire Systems
3.2	16-01-2009	Saffire Systems
3.3	02-02-2009	Saffire Systems
3.4	02-18-2009	Saffire Systems
3.5	05-01-2009	Saffire Systems
3.6	06-11-2009	Saffire Systems
3.7	06-17-2009	Saffire Systems
3.8	06-30-2009	Saffire Systems
4.0	01-27-2010	Saffire Systems
4.1	04-13-2010	J. Smith (CygnaCom Solutions)

Table of Contents

Table of Tables

Table of Figures

1 INTRODUCTION

1.1 Purpose

This non-proprietary document specifies the Security Policy for Cellcrypt's CCORE Module and was developed as part of the Federal Information Processing Standard (FIPS) 140-2 Level 1 validation of CCORE version 0.6.0-rc3 (hereafter CCORE or CCORE Module). This document may be freely reproduced and distributed as published.

The FIPS 140-2 standard, and information on the CMVP, can be found at http://csrc.nist.gov/cryptval.

1.2 Security Level Summary

FIPS 140-2 Section	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
Electromagnetic Interference/Electromagnetic Compatibility	1
Self-tests	1
Design Assurance	1
Mitigation of Other Attacks	N/A

Table 1 Security Level Summary

2 CCORE MODULE SPECIFICATION

The CCORE module is a Software Cryptographic Module. For the purposes of FIPS 140-2 validation, the CCORE module is delivered as a discrete unit of binary object code. The binary object code is generated from a specific set and revision level of C language source files. These platform-portable source files are compiled to create this object code for execution on a general purpose computer for use by an application to provide cryptographic services. The purpose of the CCORE module is to provide, through an API library, cryptographic service support for Cellcrypt's secure Voice-Over-IP products. The major cryptographic services provided by the module are:

- Initialization of Critical Security Parameters for peer to peer communication
- Key establishment between two peers
- Encryption of a maximum 4KB block of user data at a time
- Decryption of a maximum 4KB block of user data at a time

The CCORE module is distributed in its binary object code form as part of Cellcrypt's software package applications. Since the CCORE module is a Software

Cryptographic Module, it is conceptually comprised of two sub-elements: a Physical Cryptographic Module (PCM) and the Logical Cryptographic Module (LCM). The above services execute/operate within the PCM that contain the processor or central processing unit (CPU) and memory devices interconnected with a system bus. For the purposes of FIPS 140-2 level 1 validation the CCORE module can be considered a multi-chip standalone module.

The CCORE module library implements a secure and reliable transport layer. It contains all functions for handling encryption and decryption of data. The library is used by the originating call handler (the initiator) and the terminating call handler (the responder). The main services of key establishment and data encryption by an initiator or responder instance of the user application are performed by invoking CCORE functions.

The module establishes an encrypted tunnel and data sent through that tunnel is processed twice. In the first pass output data is processed with 256-bit RC4 obfuscation, which is treated as plaintext manipulation for the purpose of FIPS 140-2 validation. For this 1st pass keys are established using the Elliptic Curve Diffie-Hellman (ECDH) protocol and authentication between peers is performed with ECDSA digital signing. The second pass is a FIPS 140-2 compliant encryption, using 256 bit AES. For the 2nd pass, keys are established using RSA key transport and authentication between peers is performed with RSA Digital signing. All data output from the module is processed by both passes, so it is first obfuscated using RC4 and then encrypted using AES.

2.1 Hardware Platform

The module was tested for FIPS 140-2 level 1 requirements on a Ubuntu Server 8.04.1 operating system running on a Pentium 4 computer platform.

2.2 Module Boundary

The logical cryptographic boundary for the CCORE module is the LCM boundary which surrounds the discrete contiguous block of binary object code containing the machine instructions and data (that resides on the local file system) generated from the CCORE source code, as used by the user application. The logical cryptographic boundary is defined in terms of an Application Programming Interface (API) which is the logical interface to the CCORE module.

The CCORE module interfaces to the underlying modules SIPm and RIPm. SIPm deals with the network signalling and RIPm deals with the media transport. They are only involved in the transport of data to and from the network and so are outside of the cryptographic boundary.

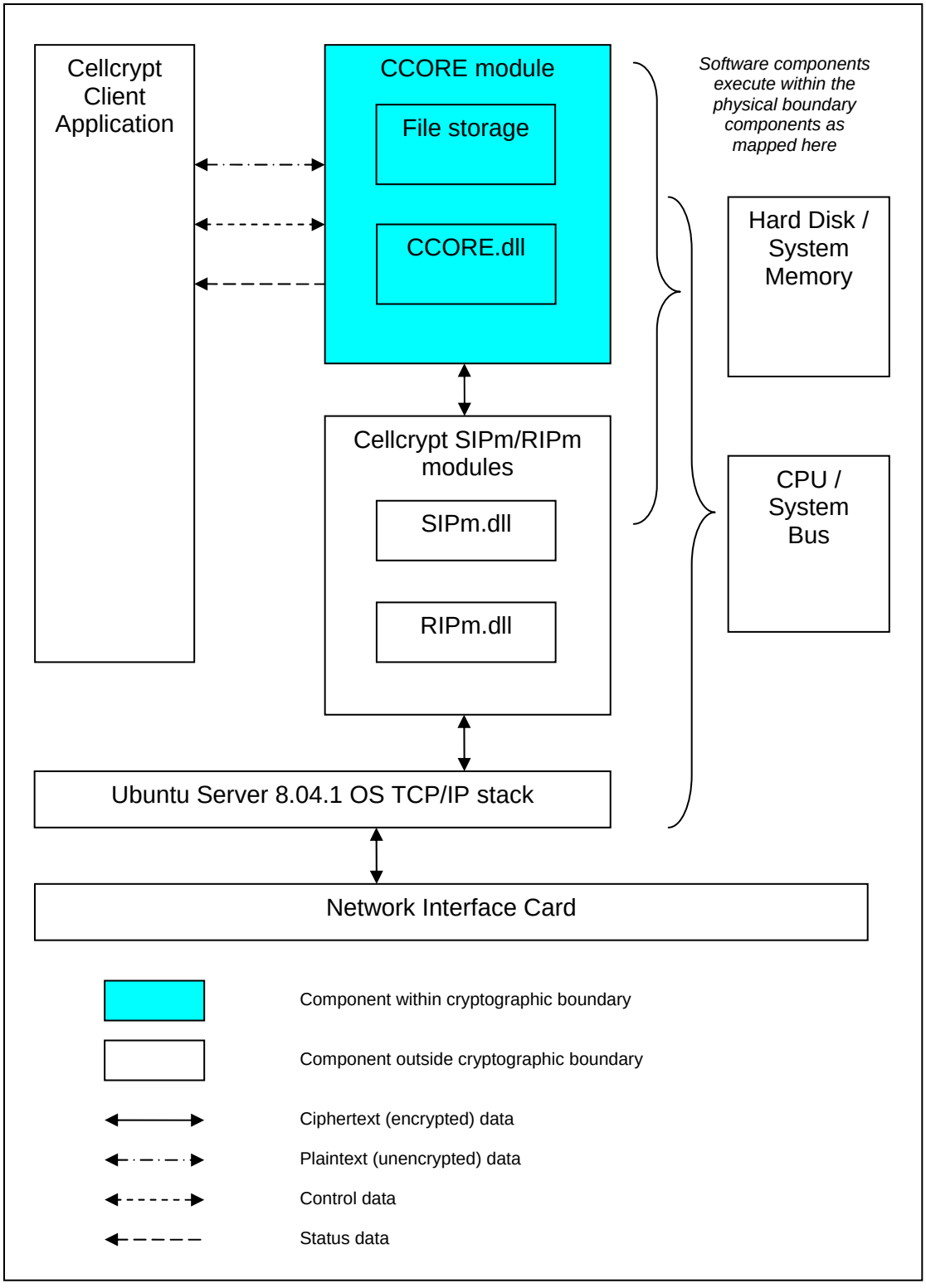

Figure 1 Logical Diagram

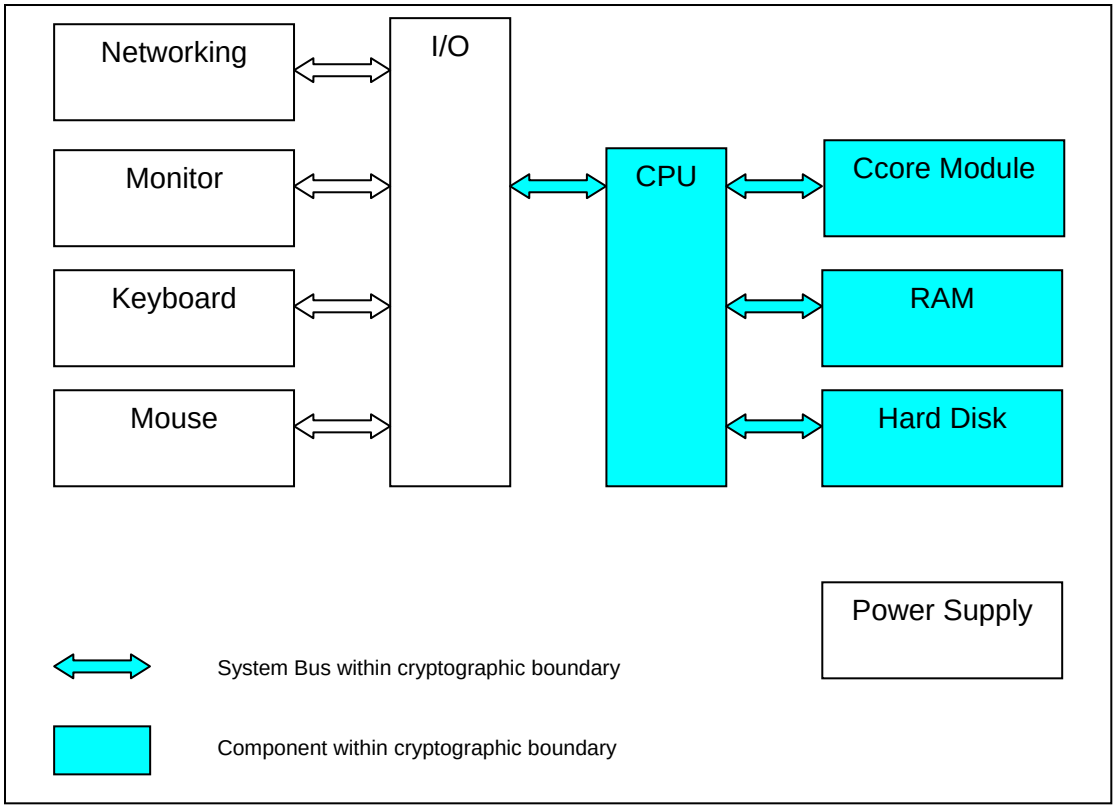

Figure 2 Physical Cryptographic Module

2.3 FIPS-Approved Mode of Operation

The module is designed to only work in an Approved FIPS mode of operation. There is no special configuration required to operate in a FIPS-Approved mode. The services of the module, when used by an application, provide FIPS 140-2 level 1 compliant cryptographic operations.

3 PORTS AND INTERFACES

The following table describes the ports and interfaces to the CCORE Module.

FIPS Logical Input/Output	Logical Interface	Physical Entry/Exit Port
Data input	API function and the context passed with it contain the data input to the module. The data input will be provided by either the network protocol stack for data from the network port or from system memory for data provided by a user application.	System Bus towards Network Port / System memory

FIPS Logical Input/Output	Logical Interface	Physical Entry/Exit Port
Data output	API function and the context passed with it contain the data output from the module. The data output from the module will either be output through the network protocol stack for data to be sent from a network port or output to system memory for data to be provided to a user application.	System Bus from Network Port / System memory
Control input	API function call provides the control data input to the module.	System memory
Status output	Return codes (both error and non-error) from called functions.	System memory

Table 2 Ports and Interfaces

The operating system software is responsible for keeping logical interfaces that share the same physical port separate. For example, encrypted data sent to and received from the network will pass through the network port of the general purpose computer. Received and sent data from/to the network is processed by the network protocol stack software of the operating system and delivered to or read from the CCORE module as a data structure.

4 ROLES, SERVICES AND AUTHENTICATION

The cryptographic services are provided through the API of the module. There are two roles supported with the module. They are the User role and Crypto Officer role. Roles are implicitly assumed by an application that uses the services implemented in the module. The application switches roles implicitly depending on which API call is issued by the application.

The Crypto Officer role is implicitly assumed when the application requests initialization of the module or zeroization of the data in the module. Self tests are performed as a part of the initialization.

After the module has been initialized, the User role is implicitly assumed when the application requests services implemented by the module that makes use of services defined through the module's API, except initialization, executing self-tests, and zeroization.

Note that only a single application may access the module services at any one time.

The services provided by the module are listed in the following table.

Role	Service	API/Function call
Crypto Officer	Initialization / Perform self-tests	CC_new()
Crypto Officer	Zeroization of CSPs (session keys, RNG seed) and all session data[2]	CC_deinit()
User	Create new peer	CC_newpeer()
User	Show Status	CC_FIPS_geterror()
User	Zeroization of peer session keys and data	CC_deinitpeer()
User	Key establishment (initiator) with peer to peer authentication using digital signing	CC_encode()
User	Key establishment (responder) with peer to peer authentication using digital signing	CC_decode()
User	Encryption of a maximum of 4KB block of user data at a time	CC_encode()
User	Decryption of a maximum of 4KB block of user data at a time	CC_decode()
User	Process timeouts	CC_tick()

Table 3 Services Authorized for Roles

Service	Cryptographic Key/CSP	Type of Access to Key/CSP
Initialization / Perform Self-tests	RNG seed file, Asymmetric private keys (RSA and ECDSA)	W
Zeroization of CSPs (session keys, RNG seed) and all session data[3]	Asymmetric public and private keys (RSA and ECDH), Asymmetric public key (ECDSA), Symmetric keys (AES and RC4)	W

Table 4 Access Rights within Services for Crypto Officer

[2] Note: The module's RSA key file must be manually deleted. See Section 10.1, Crypto Officer Guidance.
[3] Note: The module's RSA key file must be manually deleted. See Section 10.1, Crypto Officer Guidance.

Service	Cryptographic Key/CSP	Type of Access to Key/CSP
Create new peer	None	N/A
Show Status	None	N/A
Zeroization of peer session keys and data	Asymmetric public keys (RSA and ECDH), Asymmetric public key (ECDSA), Symmetric keys (AES and RC4)	W
Key establishment with peer to peer authentication using digital signing	RNG seed file	R
	Asymmetric public and private keys (RSA, ECDH and ECDSA)	R, W, E
	Symmetric keys (AES and RC4)	R, W
Encryption/Decryption (2nd pass)	Symmetric key (AES)	E
Obfuscation/De-obfuscation (1st pass)	Symmetric key (RC4)	E
Process timeouts	None	N/A

Table 5 Access Rights within Services for User

Note that the Obfuscation/De-obfuscation as used for the 1st pass is regarded as manipulation of plaintext user data for FIPS 140-2 level 1 validation purposes.

While authentication of peers for the creation of the encrypted tunnel is provided, authentication of User Role or Crypto Officer Role, or of operators using the application which assumes those roles, is not provided by the CCORE module. It is the responsibility of the operator of the general purpose computer that is running the user application, or the application itself, that uses the services of the module to provide a secure environment in terms of restricting non-authorized use.

5 PHYSICAL SECURITY

This section is not applicable, as physical security is not required for software modules.

6 OPERATIONAL ENVIRONMENT

The module was tested using the Ubuntu operating system on a Pentium 4 platform. In accordance with IG[4] Section G.5, the module will remain compliant with the FIPS 140-2 validation when operating on the following operating systems provided that the general purpose computer (GPC) uses the specified single user operating system/mode specified on the validation certificate, or another compatible single user operating system:

- Red Hat
- Debian
- Gentoo
- CentOS
- Fedora
- Gnu
- Suse

[4] Implementation Guidance for FIPS PUB 140-2 and the Cryptographic Module Validation Program.

During operation, the keys, seed values and other CSPs are used in the process space of the CCORE module. The operating system uses its inherent memory management mechanisms to ensure that other processes do not access the process space used by the module.

Outside of operation the module stores a seed file and keys on the local file system. It is the responsibility of the operator of the general purpose computer that is running the user application that uses the services of the module to provide a secure environment in terms of protecting this data.

7 CRYPTOGRAPHIC KEY MANAGEMENT

Other than a seed file that the application must provide, all Cryptographic Keys and Critical Security Parameters are internally managed by the module, i.e. they cannot be input or output to/from the module.

The module manages a tunnel with two processing passes. The first pass obfuscates data using 256-bit RC4. Keys are established using the Elliptic Curve Diffie-Hellman (ECDH) protocol. Authentication between peers is performed with ECDSA digital signing.

The second pass encrypts data using 256-bit AES encryption. Keys are established using RSA key transport. Authentication between peers is performed with RSA digital signing.

7.1 FIPS Approved Algorithms

Algorithm Type	**Algorithm**	**Standard**	**Use**	**FIPS Certificate #**
Symmetric Key	AES CTR Mode (256 bits)	FIPS 197	Encrypt/Decrypt user tunnel traffic	Cert #1089
Hashing	SHA-384	FIPS 180-3	Hashing	Cert #1022
Hashing	SHA-512	FIPS 180-3	Hashing, key derivation	Cert #1022
HMAC	HMAC (SHA-512)	FIPS 198-1	Software integrity check	Cert #612
RNG	RNG (ANSI X9.31)	ANSI X9.31	Key generation	Cert #611
Asymmetric Key	RSA	RSASSA-PKCS1_V1_5	Signing, Verification	Cert #514

Table 6 FIPS Approved Algorithms

7.2 FIPS Allowed Algorithms

Algorithm Type	Algorithm	Standard	Use	FIPS Certificate #
Asymmetric Key	RSA (key wrapping; key establishment methodology provides 112 bits of encryption strength)	N/A	Session key establishment	N/A

Table 7 FIPS Allowed Algorithms

7.3 FIPS Non-Approved and Other Algorithms

Algorithm Type	Algorithm	Use	Notes
Symmetric Key	RC4 (256 bits)	Obfuscate/De-obfuscate user tunnel traffic	RC4 is used for the 1st pass which is regarded as manipulation of plaintext user data for FIPS 140-2 level 1 validation purposes.
Hashing	MD5	Hashing, Key generation	MD5 is used for the 1st pass which is regarded as manipulation of plaintext user data for FIPS 140-2 level 1 validation purposes.
Key Agreement	Elliptic Curve Diffie-Hellman (ECDH) (key agreement; key establishment methodology provides 112 bits of encryption strength)	Session key agreement	The *Elliptic Curve Diffie-Hellman (ECDH)* algorithm is a non-compliant implementation.
Asymmetric Key	Elliptic Curve DSA	Signing, verification	The ECDSA algorithm is a non-compliant implementation.

Table 8 FIPS Non-Approved and Other Algorithms

7.4 Cryptographic Keys and Critical Security Parameters

Cryptographic Key/CSP	Storage	Use
ECDSA Public Key	RAM	Verification by peer
ECDSA Private Key	Non-volatile memory	Signing
ECDH Public Key	RAM	Session key agreement (RC4)
ECDH Private Key	RAM	Session key agreement (RC4)
RSA Public Key (2048 bits)	RAM (see note 1)	Session key establishment (AES), verification by peer
RSA Private Key	Non-volatile memory	Session key establishment (AES), signing
Integrity Check HMAC Key	Non-volatile memory	Software integrity check

(160 bits)		
AES Session Key (256 bits)	RAM	Encrypt/Decrypt FIPS tunnel traffic
RC4 Session Key (256 bits)	RAM	Obfuscate/De-obfuscate user tunnel traffic
RNG Seed	Non-volatile memory	Seed used for RNG

Table 9 Cryptographic Keys and Critical Security Parameters

The RSA Public key and ECDSA Public key of any peer that the module communicates with are stored in non-volatile memory. These keys are used to verify the signatures in the 2nd and 1st passes respectively.

8 SELF TESTS

The CCORE Module performs self tests at initialization. The tests are as follows:

8.1 Power-up Self-Tests

The following algorithm tests are performed at power-up of the CCORE module. Note that KAT tests are Known Answer Tests.

Algorithm	**Test**
AES CTR Mode (256 bits)	KAT for encryption only
RNG/PRGN (ANSI X9.31)	ANSI x9.31 KAT
RSA (digital signature)	KAT for digital signature generation and verification
RSA (encryption)	KAT for encryption and decryption
HMAC (SHA-512)	Integrity Test (no separate KAT required – IG Section 9.3)
SHA-512	No separate KAT required – IG Section 9.3
SHA-384	No separate KAT required – IG Section 9.2

Table 10 Power-Up Self Tests

In addition to algorithm tests, HMAC (SHA-512) is used to perform a software integrity test at power up.

8.2 Conditional Self-Tests

The following conditional self tests are provided.

Algorithm	**Test**
RSA key generation	Pairwise consistency test verifying that the key pair is correct for encryption/decryption and for signature/verification.
RNG (X9.31)	Check each 4 byte chunk to verify it is different than the previous 4 byte chunk and discard the first 4 byte chunk from the seed value.

Table 11 Conditional Self Tests

9 MITIGATION OF OTHER ATTACKS

This is not applicable. The module does not mitigate against other attacks.

10 OPERATIONAL GUIDANCE

10.1 Crypto Officer Guidance

The Crypto Officer role initializes the module. This role is assumed by the application that uses the module on the operating system of the general purpose computer. The interface to the module for the application is through the logical interface of the CCORE API library.

The following file needs to be created by the application and stored in the base directory that is specified by the function *CC_new* when it is invoked by the application. The file contents are used as a seed (256 bytes) for the RNG used by CCORE.

- cf_seed.dat which stores the RNG seed

Before initialization the application needs to:

- Create a session context structure
- Set the Maximum Transmission Unit (MTU) for the network (CC_SET_CTX_MTU) and the MTU for the audio transmission (CC_SET_AUDIO_MTU)

To fulfil the role at initialization the application needs to invoke the *CC_new* function of the CCORE API library. This function initializes the following security related information:

- Private RSA key (stored in file referenced through header definition CC_KEY_NAME)
- Private Elliptic Curve Digital Signature Algorithm key (stored in file referenced through header definition CC_ECDSA_KEY_NAME)

Both during and outside of operation these files are stored on the local file system. It is the responsibility of the operator of the general purpose computer that is running the user application that uses the services of the module to provide a secure environment in terms of protecting these files.

The *CC_new* function additionally performs self tests. See the self test matrices in Section 8 for details. If a self test fails then the module will report an error and the module will zeroize all context data including all keys and CSPs stored in RAM/volatile memory. It is the responsibility of the operator of the general purpose computer that is running the user application, or the application itself, to decide what to do next.

Zeroization of keys and CSPs should be performed when the module is no longer required to operate. To fulfil this, the Crypto Officer role needs to invoke the *CC_deinit* function of the CCORE API library to zeroize the keys from memory.

Additionally the content of the files stored on the local file system need to be zeroized and then deleted.

Authentication of the operator in the Crypto Officer role is not provided. It is the responsibility of the operator of the general purpose computer that is running the user application, or the application itself, that uses the services of the module to provide a secure environment in terms of restricting non-authorized use.

10.2 User Guidance

The User role is implicitly assumed by an application that uses the services implemented in the module and is executing on the general purpose computer that makes use of the services defined through the module's API. The interface to the module for the User application is through the logical interface of the CCORE API library.

An operator executing an application that uses the module will be able to communicate with another individual over a secure transport layer tunnel encrypted using 256-bit AES where keys have been established using RSA key transport.

The User role that is assumed by the application needs to set the CCORE flag CC_SET_INITIATOR if it is to act as the initiator.

The User role (as the initiator or responder) that is assumed by the application needs to invoke the *CC_newpeer* function of the CCORE API library as the first thing that it does in a session. This function initializes all structures to be used.

The functions *CC_encode* and *CC_decode* of the CCORE API library serve different purposes as and when invoked by an application. Their initial invocation will result in the execution of the CCORE Cryptographic Key Handshake Protocol.

The first invocation of the *CC_decode* function will initiate cryptographic key establishment for the initiator using the CCORE Cryptographic Key Handshake Protocol. The responder's invocation of *CC_decode* function will perform the cryptographic key establishment by responding to the initiator using the CCORE Cryptographic Key Handshake Protocol. This is internally driven through flags and states.

When communication is made with a peer for the first time then a directory is created for that peer. The following files are created in the peer directory.

- *rsa_pub-tmp.bin* which stores the received RSA public key of the peer before authentication of the peer
- *rsa_pub.bin* which stores the received RSA public key of the peer after authentication of the peer
- *ec_pub.bin* which stores the received ECDSA public key of the peer.

When the User role that is assumed by the application (as the initiator or responder) needs to send encrypted data, it invokes the *CC_encode* function.

When the User role that is assumed by the application (as the initiator or responder) needs to receive encrypted data, it invokes the *CC_decode* function.

The User role that is assumed by the application must invoke the *CC_tick* function in order that the CCORE module can process timeouts. This must be at least every second.

When the User role that is assumed by the application needs to know the status of the module it invokes the *CC_FIPS_geterror* function of the CCORE API library.

At the completion of a session the User role that is assumed by the application needs to invoke the *CC_deinitpeer* function of the CCORE API library. The function clears/zeroizes all data used in the session.

When an error is reported to the application, the module will zeroize all context data including all keys and CSPs stored in RAM/volatile memory. Calls to all I/O are blocked. Calls to *CC_FIPS_geterror*, *CC_deinit* and *CC_deinitpeer* are not blocked.

Authentication of the operator in the User Role is not provided. It is the responsibility of the operator of the general purpose computer that is running the user application, or the application itself, that uses the services of the module to provide a secure environment in terms of restricting non-authorized use.

11 GLOSSARY

ACRONYM	DEFINITION
AES	Advanced Encryption Standard
API	Application Programming Interface
CSP	Critical Security Parameter
CTR	Counter
DSA	Digital Signature Algorithm
E	Execute
ECDH	Elliptic Curve Diffie-Hellman
ECDSA	Elliptic Curve Digital Signature Algorithm
FIPS	Federal Information Processing Standards
GPC	General Purpose Computer
HMAC	keyed-Hash Message Authentication Code
I/O	Input/Output
KAT	Known Answer Test
LCM	Logical Cryptographic Module
OS	Operating System
PBX	Private Branch Exchange
PCM	Physical Cryptographic Module
R	Read
RAM	Random Access Memory
RNG	Random Number Generator
RIPm	Cellcrypt Real Time Protocol for Mobile
SHA	Secure Hash Algorithm
SIP	Service Information Protocol
SIPm	Cellcrypt Service Information Protocol for Mobile
W	Write