FORTEZZA CRYPTO CARD

SECURITY POLICY

SPYRUS®

Fortezza Crypto Card

Security Policy

Document # 540-105000-A1
Revision A1
31 December 1998

SPYRUS®
<info@spyrus.com> <http://www.spyrus.com>

Contents

1 INTRODUCTION 1

1.1 Scope 1

2 OVERVIEW 1

3 FUNCTION FORMATS 1

3.1 Function Description 1
3.2 Parameter Formats 2
3.3 Data Structures 2
3.4 Function Organization 3

4 FUNCTION DESCRIPTIONS 4

4.1 CI_ChangePIN 4
4.2 CI_CheckPIN 4
4.3 CI_Decrypt 5
4.4 CI_DeleteCertificate 5
4.5 CI_DeleteKey 5
4.6 CI_Encrypt 6
4.7 CI_ExtractX 6
4.8 CI_FirmwareUpdate 7
4.9 CI_GenerateIV 8
4.10 CI_GenerateMEK 8
4.11 CI_GenerateRa 9
4.12 CI_GenerateRandom 9
4.13 CI_GenerateTEK 9
4.14 CI_GenerateX 9
4.15 CI_GetCertificate 9
4.16 CI_GetConfiguration 10
4.17 CI_GetHash 11
4.18 CI_GetPersonalityList 11
4.19 CI_GetState 13
4.20 CI_GetStatus 13
4.21 CI_GetTime 13
4.22 CI_Hash 13
4.23 CI_Initialize 14
4.24 CI_InitializeHash 14
4.25 CI_InstallX 14
4.26 CI_LoadCertificate 14
4.27 CI_LoadDSAParameters 15
4.28 CI_LoadInitValues 15
4.29 CI_LoadIV 15
4.30 CI_LoadX 16
4.31 CI_Lock 16
4.32 CI_Open 16
4.33 CI_RelayX 16

4.34 CI_Reset 16
4.35 CI_Restore 17
4.36 CI_Save 17
4.37 CI_Select 17
4.38 CI_SetConfiguration 17
4.39 CI_SetKey 17
4.40 CI_SetMode 18
4.41 CI_SetPersonality 18
4.42 CI_SetTime 18
4.43 CI_Sign 18
4.44 CI_Terminate 18
4.45 CI_TimeStamp 19
4.46 CI_Unlock 19
4.47 CI_UnwrapKey 19
4.48 CI_VerifySignature 19
4.49 CI_VerifyTimeStamp 19
4.50 CI_WrapKey 20
4.51 CI_Zeroize 20

5 ROLES 20

5.1 Crypto-Officer Role 20
5.2 User Role 21
5.3 Self-test 23

Revision History

REV. #	DATE	DESCRIPTION
A1	31 Dec 98	Original

1 Introduction

The Fortezza Crypto Card (which will be referred to as the PCcard for the rest of this document) is a small electronic device developed by SPYRUS for the U.S. Government. The devices is primarily used within the U.S. Government with the Defense Messaging System (DMS). The PCcard communicates with the host computer via a standard PCMCIA interface.

The PCcard is a cryptographic module which implements the following Cryptographic algorithms Digital Signature Algorithm (DSA) FIPS PUB 186, Secure Hash Algorithm (SHA-1) FIPS PUB 180-1, Key Exchange algorithm (KEA) , and SKIPJACK. The PCcard also contains a PCMCIA interface which complies with PCMCIA Standard 2.1.

1.1 Scope

This document describes the security policy for the PCcard.

2 Overview

The PCcard supports a set of commands that the host computer sends to it via the PCMCIA interface. These commands are used to support cryptographic based authentication and encryption applications.

The PCcard is a multi-chip standalone module based on the Fortezza Interface Control Document (ICD) version 1.52.

3 Function Formats

3.1 Function Description

Each function description contains:

- Section header
- Textual description of the function
- Parameter passing notation
- C language definition
- Parameter definition
- Return values (italic for card response, normal for library response) Note: Not all possible return values are listed for each command. Section 2.1.3, General Error Notices, defines some potential errors which usually are not listed with any function.
- The Cards State Transitions (if applicable)

The Parameter passing notation is:

Function Name: [Input Parameters]
{Output Parameters}

If the function does not require any input then [none] is used.
All functions return an integer result.

The C language definitions mostly conform to the ANSI C standard. One exception may be the use of the keyword ‘far’. The keyword ‘far’ is used for the MS DOS environment and may be omitted in all other environments. To effectively remove the keyword ‘far’ the ANSI C preprocessor command:

```
#define far
```

is used. This will be done in the any header file that contains the ‘far’ keyword.

Note: Currently the CI Library does not use the ‘far’ directive. It is the responsibility of the person building the CI Library to ensure that the source code is compiled correctly. For the MS DOS and MS Windows environments, this includes ensuring that the appropriate memory model and data alignment (single or double byte) is used. The other exception is the time functions do not use ANSI compliant commands, although the commands are supported on all major industry standard platforms.

3.2 Parameter Formats

The target systems for the CI Library are MS DOS, UNIX and Macintosh. MS DOS uses little endian numeric representation. Macintosh and most UNIX implementations use big endian numeric representation. The CI Library is written so that the functions may be called using the host system’s native numeric representation. The Card uses a 32 bit word in the big endian numeric format. For MS DOS, the CI Library will convert all 16 bit values into 32 bit values and convert the little endian number into big endian transparently to the user of the CI Library.

For larger data objects, such as **CI_PIN**, the data must be placed into the buffer such that the first byte of the data object is located into the first byte of the buffer where the first byte of the buffer has the lowest address. A PIN phrase such as “CRYPTOGRAPHY” must be in a buffer as follows:

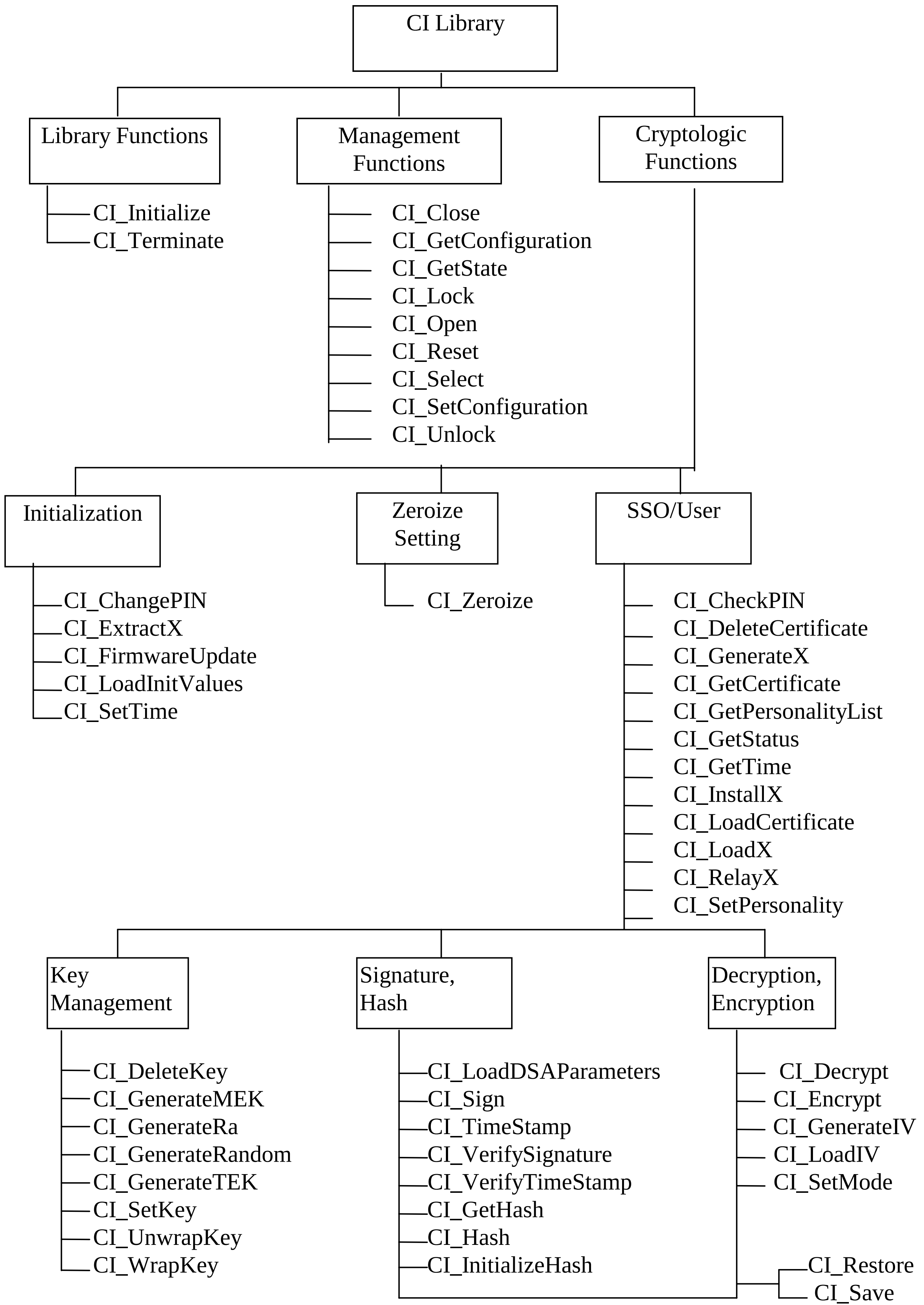
```
PIN[0] = ‘C’; PIN[1] = ‘R’; PIN[2] = ‘Y’; PIN[3] = ‘P’; PIN[4] = ‘T’; PIN[5] = ‘O’;
PIN[6] = ‘G’; PIN[7] = ‘R’; PIN[8] = ‘A’; PIN[9] = ‘P’; PIN[10] = ‘H’; PIN[11] = ‘Y’
```

where PIN[0] is the first byte, the byte with the lowest address, of the PIN phrase.

3.3 Data Structures

When data objects are loaded from ASCII/ANSI text files, the text should be evaluated from left to right, top to bottom. The first two characters of the ASCII hex string are converted to an unsigned eight bit byte and placed into the first byte of the buffer. The remaining ASCII hex character pairs are converted to unsigned byte values and placed into consecutively higher addresses of the buffer. This must be done on the byte level, otherwise byte swapping may occur.

3.4 Function Organization



4 Function Descriptions

4.1 CI_ChangePIN

The **CI_ChangePIN** function allows the SSO Enabled User to change the SSO or User PIN phrase given the current PIN phrase. The parameter PINType specifies if the PIN phrase is the SSO or User PIN. The constant **CI_PIN_SIZE** is defined to be 12 bytes and **CI_PIN** is a 16 byte character array: **CI_PIN_SIZE** + a 1 NULL byte terminator + a 3 byte pad (to assure byte alignment on word boundaries). The CI Library will pad the PIN phrases with ‘space’ characters (0x20) to **CI_PIN_SIZE** (12) bytes before passing it to the Card.

4.2 CI_CheckPIN

The **CI_CheckPIN** function determines if the PIN phrase is valid. The parameter PINType specifies if the PIN phrase is the SSO or User PIN. The constant **CI_PIN_SIZE** is defined to be 12 bytes and **CI_PIN** is a 16 byte character array: **CI_PIN_SIZE** + a 1 NULL byte terminator + a 3 byte pad (to assure byte alignment on word boundaries). The CI Library will pad the PIN phrase with ‘space’ characters (0x20) to **CI_PIN_SIZE** (12) bytes before passing it to the Card.

Card Notice: The Card allows only 9 consecutive incorrect PIN values. If a user enters the User PIN wrong 10 consecutive times, the Card transitions to the LAW Initialized State. No data on the Card is lost but the user must take the card back to the SSO. If a User enters an SSO PIN incorrectly 10 consecutive times, the Card will transition to the Zeroized State whereby all data on the Card is lost.

PIN Types	Description
CI_SSO_PIN	Check the SSO PIN phrase.
CI_USER_PIN	Check the User PIN phrase.

4.3 CI_Decrypt

The **CI_Decrypt** function decrypts the data pointed to by pCipher and places it in the buffer pointed to by pPlain. The CipherSize parameters specifies the number of bytes to decrypt and the number of bytes in the buffer pointed to by pPlain. The pointer pPlain may point to the same buffer as pCipher so that the plaintext will overwrite the ciphertext. Due to the limited amount of space on the Card, large data sets may be decrypted with multiple calls to **CI_Decrypt**. Use the **CI_GetConfiguration** function to determine the amount of user memory on the Card.

Prior to executing the **CI_Decrypt** function the decryption mode needs to be set, the decryption key loaded into the cryptologic and an IV needs to be loaded. The mode is set by the **CI_SetMode** function. The default decryption mode is 64 bit Cipher Block Chaining (**CI_CBC64**). The **CI_LoadIV** function is used to load the IV. For multi-call decryption sessions the IV only needs to be loaded prior to the first call to **CI_Decrypt**. The key is set with the **CI_SetKey** function.

4.4 CI_DeleteCertificate

The **CI_DeleteCertificate** function zeroizes the Certificate and Certificate Label and any Private Component (X), Public Component (Y) and Public Key Parameters (p, q, and g) associated with the Certificate specified by the CertificateIndex. The Card only allows an SSO Enabled User to delete Certificate Index 0. Deleting an unused Certificate Index is permitted.

4.5 CI_DeleteKey

The **CI_DeleteKey** function zeroizes the Key Register specified by RegisterIndex. Deleting an unused Key Register is permitted.

The Card uses Key Register zero (0) to hold it’s Storage key, Ks. The Card only allows the SSO Enabled User to delete this register.

4.6 CI_Encrypt

The **CI_Encrypt** function encrypts the data pointed to by pPlain and places it into the buffer pointed to by pCipher. The PlainSize parameter specifies the number of bytes to encrypt and the number of bytes in the buffer pointed to by pCipher. The pointer pCipher may point to the same buffer as pPlain so that the ciphertext will overwrite the plaintext. Due to the limited amount of space on the Card, large data sets may be encrypted with multiple calls to **CI_Encrypt**. Use the **CI_GetConfiguration** function to determine the amount of user memory on the Card.

Prior to executing the **CI_Encrypt** function the encryption mode needs to be set, the encryption key loaded into the cryptologic and an IV must be generated. The mode is set by the **CI_SetMode** function. The default encryption mode is 64 bit Cipher Block Chaining (**CI_CBC64**). The **CI_GenerateIV** function is used to generate the IV. For a multi-call encryption session the IV is only generated prior to the first call to **CI_Encrypt**. The key is set with the **CI_SetKey** function.

4.7 CI_ExtractX

The **CI_ExtractX** function allows the SSO Enabled User to retrieve a Private (X) value covered using the Public Key Exchange protocol. Only those Private values loaded or generated under the SSO PIN may be extracted. A valid personality must be set (via **CI_SetPersonality**) before this function is executed.

The constant **CI_PASSWORD_SIZE** is defined to be 24 bytes and **CI_PASSWORD** is a 28 byte character array: **CI_PASSWORD_SIZE** + a 1 NULL byte terminator + a 3 byte pad (to assure byte alignment on word boundaries). Also, the CI Library pads the password with zero (0) to **CI_PASSWORD_SIZE** (24) bytes before passing it to the Card.

4.8 CI_FirmwareUpdate

The **CI_FirmwareUpdate** function loads a complete new set of application software onto the Card.

Flags	Description
CI_DESTRUCTIVE_FLAG	The non-volatile memory of the Card is to be zeroized. It may not be used with the **CI_NONDESTRUCTIVE_FLAG**.
CI_LAST_BLOCK_FLAG	This is the last block of the firmware. It may not be used with the **CI_NOT_LAST_BLOCK_FLAG**.
CI_NONDESTRUCTIVE_FLAG	The non-volatile memory of the Card is **not** to be zeroized. It may not be used with the **CI_DESTRUCTIVE_FLAG**.
CI_NOT_LAST_BLOCK_FLAG	This is **not** the last block of the firmware. It may not be used with the **CI_LAST_BLOCK_FLAG**.

CI_FirmwareUpdate: [Flags, Cksum, CksumLength, DataSize, pData] {return value}

int CI_FirmwareUpdate(long Flags, long Cksum, unsigned int CksumLength, unsigned int DataSize, CI_DATA pData)

Parameter	Type	Description
Flags	unsigned long	Options for the firmware update from the list above.
Cksum	long	The checksum value.
CksumLength	unsigned int	The length of the checksum.
DataSize	unsigned int	Number of bytes in the buffer pointed to by pData.
pData	**CI_DATA**	Pointer to the buffer containing the firmware to load.
return value	int	The function’s completion code.

Value	Meaning
CI_OK	*The firmware was updated.*
CI_FAIL	*The firmware was not updated.*

CI_CHECKWORD_FAIL	*The checkword did not match.*
CI_INV_SIZE	*The data size is not valid.*
CI_INV_STATE	*This function may not be executed in this Card state.*
CI_INV_POINTER	*The Card detected an invalid pointer.*
CI_TIME_OUT	The Card failed to complete the command.
CI_NULL_PTR	The pointer to the firmware is NULL.
CI_NO_CARD	The Card was not found.
CI_NO_SOCKET	A Socket has not been opened.

Entry State	Exit State
Uninitialized	Uninitialized-Card is reset
Initialized	Uninitialized-Card is reset
SSO Initialized	SSO Initialized-Card is reset if Non Destructive or Uninitialized if Destructive
LAW Initialized	LAW Initialized-Card is reset if Non Destructive or Uninitialized if Destructive
User Initialized	User Initialized-Card is reset if Non Destructive or Uninitialized if Destructive

4.9 CI_GenerateIV

The **CI_GenerateIV** function generates an Initialization Vector (IV). The IV is stored in the cryptologic engine and returned in pIV. This function must be used before an encryption process (**CI_Encrypt**) can be performed.

4.10 CI_GenerateMEK

The **CI_GenerateMEK** function generates a random Message Encryption Key (MEK). The MEK will be placed into the register indicated by the RegisterIndex parameter. The **CI_GenerateMEK** is used prior to the **CI_Encrypt** function to generate a key.

4.11 CI_GenerateRa

The **CI_GenerateRa** function will generate a R_a. The 1024 bit (128 byte) R_a is returned in pRa. The R_a is used with **CI_GenerateTEK** function.

4.12 CI_GenerateRandom

The CI_GenerateRandom function will generate a random number and returns it in pRandom.

4.13 CI_GenerateTEK

The **CI_GenerateTEK** function generates a Token Encryption Key (TEK) for a public/private key exchange.

For some protocols the R_b must be set to a known value. To calculate an R_b with the value one (1) in the big endian numeric format (which is required for MSP): initialize the R_b with 0, then set the least significant bit of the last byte to 1. To create an R_b with the value one in little endian: initialize the R_b with zero, then set the least significant bit of the first byte to 1. The following C code fragment will create a 1024 bit R_b with the value one in big endian:

4.14 CI_GenerateX

The **CI_GenerateX** function generates a public key pair, Private Component (X) and Public Component (Y), of the type specified by AlgorithmType. The X is saved on the Card. The Y is returned in the pY parameter and the host may integrate it into a Certificate. The Certificate can then be loaded onto the Card at the Certificate index associated with the X.

4.15 CI_GetCertificate

The **CI_GetCertificate** function returns the certificate associated with the Certificate Index specified by CertificateIndex. The certificate is 2048 bytes.

4.16 CI_GetConfiguration

The **CI_GetConfiguration** function returns a **CI_CONFIG** structure which contains:

Field Name	Data Type	Description
LibraryVersion	integer	Crypto Interface Library Version.
ManufacturerVersion	integer	The Card's Hardware Version.
ManufacturerName	an array of **CI_NAME_SIZE** + 4 (36) characters	The Card Manufacturer's name.
ProductName	an array **CI_NAME_SIZE** + 4 (36) characters	The Card's product name.
ProcessorType	an array **CI_NAME_SIZE** + 4 (36) characters	The Card's processor type.
UserRAMSize	unsigned long integer	The number of bytes of User RAM.
LargestBlockSize	unsigned long integer	The size, in bytes of the largest block of data that may be passed to a function.
KeyRegisterCount	integer	The number of Key Registers on the Card.
CertificateCount	integer	The maximum number of Certificates that the Card can store (including Certificate 0).
CryptoCardFlag	integer	A flag that if non-zero indicates that there is a Crypto-Card in the socket. If this value is zero then there is **not** a Crypto-Card in the socket.
ICDVersion	integer	The ICD Compliance level.

For example, for an ICD Compliance level of "P1.5", this value is 0015H.

ManufacturerSWVer integer The Manufacturer's Software Version

For example, given 1234H, the Firmware Version is 12, and the Hardware Version is 34.

DriverVersion integer Fortezza Device Driver Version.

The constant **CI_NAME_SIZE** is defined to be 32.

Note that ManufacturerName, ProductName, and Processor Type are 36 byte character arrays **CI_NAME_SIZE** + a 1 NULL byte terminator + a 3 byte pad (to assure byte alignment on word boundaries).

CI_GetConfiguration: [none]
{pConfiguration, return value}
int CI_GetConfiguration(CI_CONFIG_PTR pConfiguration)

Parameter	Type	Description
pConfiguration	**CI_CONFIG_PTR**	Points to the buffer that will receive the configuration information.
return value	int	The function's completion code.

Value	Meaning
CI_OK	The Configuration was successfully retrieved.
CI_NULL_PTR	The pointer to Configuration is NULL.
CI_NO_CARD	The Card is not present.
CI_NO_SOCKET	A Socket has not been opened.

4.17 CI_GetHash

The **CI_GetHash** function hashes the last block of data and returns the final Hash Value. The Hash Value is 160 bits (20 bytes). The application must call **CI_InitializeHash** before calling **CI_Hash** or **CI_GetHash**. **CI_GetHash** is called when the last (or only) block of data ends on a non multiple of 64 or is exactly 0 bits.

4.18 CI_GetPersonalityList

The **CI_GetPersonalityList** function returns the list of **CI_PERSON** structures. The **CI_PERSON** structure contains a CertificateIndex and a Certificate Label. Use the **CI_GetConfiguration** function to determine the maximum number of certificates that the Card can hold. Use the **CI_GetCertificate** function to retrieve the certificate associated with a Certificate Index.

Note that the constant **CI_CERT_NAME_SIZE** is defined to be 32 bytes and **CI_CERT_STR** is a 36 byte character array: **CI_CERT_NAME_SIZE** + a 1 NULL byte terminator + a 3 byte pad (to assure byte alignment on word boundaries).

A **CI_PERSON** data structure is defined as:

Field Name	Data Type	Description

CertificateIndex	integer	Index of the certificate
CertLabel	**CI_CERT_STR**	Personality string of the certificate

Certificate Index zero (0) can not be selected so it is not returned in the personality list.

If EntryCount is greater than the maximum number of certificates that the Card can hold, then each additional personality structure will have its CertificateIndex set to zero (0) and Certificate Label filled with zero (0). Use the **CI_GetConfiguration** function to determine the maximum number of certificates that the Card can hold. If EntryCount is < 1, an error of CI_BAD_SIZE is returned.

Note that not all of the Certificate Indexes in the personality list are valid or useable Personalities. It is up to the host application to parse the Certificate Label to determine if the Certificate Index is an appropriate Personality. If a Certificate index does not contain a certificate, the index is returned with a NULL Label. Ex: If a card holds 6 (0-5, 0 is not displayed) personalities and the user sets EntryCount to 7 then:

Entry Counter	Certificate Index	Certificate Label

4.19 CI_GetState

The **CI_GetState** function returns the execution state of the Card in pState. This function may be called at any time, regardless of if the Card has been initialized or logged on to.

4.20 CI_GetStatus

The **CI_GetStatus** function returns the status of the Card. This command may be called before the **CI_CheckPIN** function.

4.21 CI_GetTime

The **CI_GetTime** function retrieves the current time from the Card's on-board real-time clock.

Note that an application may not want to check the time more than one (1) time per second. The Card may not increment during that time. If the Card does not detect an increase in time, it may consider the clock broken and disable the time functions.

4.22 CI_Hash

The **CI_Hash** function hashes the data pointed to by pData. The hash value may be set to an initial value or to an intermediate value by calling **CI_InitializeHash** (to start a hash process) or **CI_Restore** (to continue an interrupted hash process) functions, respectively. The **CI_Hash** function will continue to hash building on the current hash value. The size of the data must be a multiple of 512 bits (64 bytes) and not equal 0. Use the **CI_GetHash** function to hash the final block of data and retrieve the Hash Value.

4.23 CI_Initialize

The **CI_Initialize** function initializes the CI Library. All other function calls will return a **CI_LIB_NOT_INIT** error if they are called before this function. To close the CI Library call **CI_Terminate**.

Note that a Socket must be opened, with the **CI_Open** function, to issue any commands to the Card.

4.24 CI_InitializeHash

The **CI_InitializeHash** function will initialize the hash value according to the DSA standard for general hash use on various block sizes of data. It must be executed before beginning a hash process, either **CI_Hash** or **CI_GetHash**.

4.25 CI_InstallX

The **CI_InstallX** function will restore an archived Private Component (X) that was extracted by the **CI_ExtractX** function.

Note that the constant **CI_PASSWORD_SIZE** is defined to be 24 bytes and **CI_PASSWORD** is a 28 byte character array: **CI_PASSWORD_SIZE** + a 1 NULL byte terminator + a 3 byte pad (to assure byte alignment on word boundaries).

Note that the CI Library pads the password with zero (0) to **CI_PASSWORD_SIZE** (24) bytes before passing it to the Card.

4.26 CI_LoadCertificate

The **CI_LoadCertificate** function loads a certificate into the non-volatile memory of the Card. The CertificateIndex parameter specifies the location of where the certificate is to be loaded. The Certificate Index is used to bind the Certificate Label and the certificate to the public key pair that was generated or loaded with **CI_GenerateX** or **CI_LoadX** function. Use the **CI_GetConfiguration** function to determine the maximum number of certificates that the Card can hold.

The constant **CI_CERT_NAME_SIZE** is defined to be 32 bytes and **CI_CERT_STR** is a 36 byte character array: **CI_CERT_NAME_SIZE** + a 1 NULL byte terminator + a 3 byte pad (to assure byte alignment on word boundaries).

4.27 CI_LoadDSAParameters

The **CI_LoadDSAParameters** loads temporary DSA p, q, and g values into the volatile memory of the Card. These values are defined as follows:

- p - the prime modulus (512 - 1024 bits (64 - 128 bytes), as indicated by PSize)
- q - the prime divisor (160 - 320 bits (20 - 40 bytes), as indicated by QSize)
- g - a value (512 - 1024 bits (64 - 128 bytes), same size as p, as indicated by PSize)

Note that the DSA Parameters will be lost when the Card is reset.

If this command is successfully performed to allow verification of a message and if the Card is in Ready State, the Card will transition to the Standby State. The host must set the user personality.

4.28 CI_LoadInitValues

The **CI_LoadInitValues** function allows the SSO Enabled User to load or modifies the Card’s initialization parameters. These parameters include:

- Random Number Seed Value - 64 bit (8 byte) seed value
- User Storage Key Variable (K_S)- A plaintext value (80 bits (10 bytes))

The Random number is loaded into the Card and used to seed the internal random number generator. The User Storage Key Variable (K_S) is used in the user’s authentication algorithm. This K_S is wrapped by the hash of the User’s PIN phrase value. The K_S value is loaded in plaintext.

Note that the SSO PIN phrase must be changed after this command has successfully completed.

4.29 CI_LoadIV

The **CI_LoadIV** function loads an Initialization Vector (IV) onto the Card for decryption operations.

4.30 CI_LoadX

The **CI_LoadX** function loads the Private Component (X), into the non-volatile memory of the Card. The Private Component is given a Certificate Index of the corresponding Certificate. The Public Component (Y), is generated and returned in pY.

4.31 CI_Lock

The **CI_Lock** function grants an application with exclusive access to the currently selected socket and its Card. **CI_Unlock** releases a Lock. **CI_Close** will also release a Lock.

4.32 CI_Open

The **CI_Open** function opens a Socket. Sockets are numbered from one (1) to SocketCount. (SocketCount is returned by **CI_Initialize**). All subsequent commands will be issued to the socket opened. Use the **CI_Select** command to select from any of the currently open sockets. A card does not need to be in the socket before executing this command.

4.33 CI_RelayX

The **CI_RelayX** functions allows either the SSO Enabled User or a User to change the password and re-wrap an archived Private Component, X.

Note that the constant **CI_PASSWORD_SIZE** is defined to be 24 bytes and **CI_PASSWORD_PIN** is a 28 byte character array: **CI_PASSWORD_SIZE** + a 1 NULL byte terminator + a 3 byte pad (to assure byte alignment on word boundaries).

The CI Library will pad the password with zero (0) to **CI_PASSWORD_SIZE** (24) bytes before passing it to the Card.

4.34 CI_Reset

The **CI_Reset** functions will reset the Card. All registers and common memory are zeroized. The User or SSO Enabled User will be logged off of the Card. The **CI_Reset** function does not terminate the CI Library, use the **CI_Terminate** function to close the CI Library.

4.35 CI_Restore

The **CI_Restore** function will restore the state of the cryptologic operation specified by CryptoType. The state may be restored from either the Card's internal storage or from a memory buffer. The state information in the memory buffer must have been supplied by the **CI_Save** function for the same cryptologic operation. The user should execute **CI_GenerateIV** before this command to assure proper configuration of the Crypto Engine.

4.36 CI_Save

The **CI_Save** function will save the state of the cryptologic operation specified by CryptoType. The application has the option of receiving a copy of the state information. The information includes the Card's state, CryptoMode, encrypt decrypt flag, or its intermediate hash value and bits hashed. The state of the cryptologic operation may be restored by the **CI_Restore** function.

The Card can store only one copy of each cryptologic operation state. Also, regardless of the applications request for internal or external storage, the Card will always write or duplicate the current state into the Card's internal storage for the type of operation specified. For example,

4.37 CI_Select

The **CI_Select** function selects the socket specified by SocketIndex. All subsequent commands will be issued to the socket selected. Sockets are referenced by index, which range from 1 to SocketCount. (SocketCount is returned by **CI_Initialize**).

The CI Library no longer supports requests for SocketIndex 0. This request will result in an error, **CI_INV_SOCKET_INDEX**.

4.38 CI_SetConfiguration

The **CI_SetConfiguration** function is used to set the configuration of the Card.

4.39 CI_SetKey

The CI_SetKey function loads the key specified by RegisterIndex into the cryptologic for subsequent cryptologic functions that use an implicit Key Register.

4.40 CI_SetMode

The **CI_SetMode** function is used to set the cryptologic mode to the mode specified in CryptoMode for the cryptologic operation specified in CryptoType.

4.41 CI_SetPersonality

The **CI_SetPersonality** function sets the Personality of the Card to the Certificate Index specified by CertificateIndex. The Personality defines which Certificate Index will be used to locate a certificate or Private Component (X), for use with functions such as **CI_Sign**. Use the **CI_GetCertificate** function to retrieve the certificate associated with CertificateIndex. The Personality may be changed at any time after a successful log on.

Note that not all of the Certificate Indexes in the personality list are valid Personalities. It is up to the host application to parse the Certificate Label to determine if the Certificate Index is a valid Personality. Use the **CI_GetPersonalityList** function to get the list of Certificate Indexes and Certificate Labels.

4.42 CI_SetTime

The **CI_SetTime** function allows the SSO Enabled User to set the time and date on the on-board real-time clock.

4.43 CI_Sign

The **CI_Sign** function computes a Digital Signature, using the Digital Signature Algorithm (DSA), usually over the provided Hash Value. The Hash Value is signed with the Private Component (X) of the Personality and an internally generated random value, K. Use the **CI_VerifySignature** function to verify a Signature created with the DSA.

4.44 CI_Terminate

The **CI_Terminate** function closes the CI Library. The **CI_Terminate** function will first close any open Sockets, then close the communication link with the PCMCIA Socket Services.

4.45 CI_TimeStamp

The **CI_TimeStamp** function generates a Digital Signature over the provided Hash Value and the current time from the Card's on-board real-time clock.

4.46 CI_Unlock

The **CI_Unlock** function releases an application's exclusive access, established by **CI_Lock,** to the currently selected socket and its card.

4.47 CI_UnwrapKey

The **CI_UnwrapKey** function will unwrap the wrapped key in the buffer pointed to by pKey, using the key in the Key Register indicated by UnwrapIndex. After the key is unwrapped, the Card will perform a checkword test and compare the generated value to the unwrapped value. If they compare, the key will then be loaded into Key Register indicated by KeyIndex.

4.48 CI_VerifySignature

The **CI_VerifySignature** function validates a Digital Signature, usually against the Hash Value, and signers Public Component (Y). Digital Signatures can be created with the **CI_Sign** function.

4.49 CI_VerifyTimeStamp

The **CI_VerifyTimeStamp** validates the Hash Value and Time Stamp with the Public Component (Y) supplied.

CI_VerifyTimeStamp : [pHashValue, pSignature, pTimeStamp]
{return value}
int CI_VerifyTimeStamp(CI_HASHVALUE pHashValue, CI_SIGNATURE pSignature, CI_TIMESTAMP pTimeStamp)

4.50 CI_WrapKey

The **CI_WrapKey** function will wrap the plaintext key in the Key Register indicated by KeyIndex with the key in the Key Register indicated by WrapIndex. The resulting wrapped key is returned in pKey.

4.51 CI_Zeroize

The **CI_Zeroize** function will zeroize the Card's data buffers, internal buffers, key management information, personalities, all public key pairs (X and Y), all Key Registers and disallows user access. After execution of this command the Card will enter the zeroized (**CI_ZEROIZED**) state. The Card will need to be re-initialized by the SSO Enabled User. Note that the SSO Enabled User will need to use the Zeroize Default PIN to authenticate to the Card.

5 Roles

The PCcard supports the following roles:

- Crypto-Officer Role,
- User Role

The PCcard enforces the separation of roles by restricting the services available to each role.

5.1 Crypto-Officer Role

The Crypto-Officer is responsible for initializing the PCcard. The Crypto-Officer role is only available during card initialization on a Certificate Authority Workstation (CAW). The CAW is keep secure in accordance with the site security policy of the deploying organization. The Crypto-Officer has access to all services on the card.

The PCcard validates the Crypto-Officer role by requiring a Personal Identification Number (PIN) in order to access it. A valid PIN must be passed to the PCcard before it will accept any commands required to perform the initialization service.

The Crypto-officer is also responsible for the initialization of the PCcard for use as a Crypto token, Prior to delivering the unit to the user. A valid PIN must be passed to the

PCcard to authenticate the Crypto-officer before it will process the initialization command.

The Initialization services cause the PCcard to generate its public/private key pair, export the public key, and load the X.509 certificate into the root certificate slot on the PCcard.

The Crypto-officer must also set the PIN phrase for the user, and may change the PIN phrase for the Crypto-officer. Only the Crypto-officer may change a PIN phrase.

5.2 User Role

The PCcard supports a User role, for which the following service not allowed:

- Change Pin Phrase
- Extract X
- Load Initialization Values
- Set Time

The PCcard validates the User role by requiring a Personal Identification Number (PIN) in order to access it. Some services input data into the PCcard. The PCcard further validates the requester by checking the Service State on the PCcard.

The PCcard provides services by exchanging commands between itself and a host computer over the PCMCIA Interface. The proper library software must be installed on the host to access PCcard services.

Services	Roles		Cryptographic Functions	
	Crypto-Officer	User	Yes	No
Self-test	X	X	X	
Change Pin Phrase	X		X	
Check Pin Phrase	X	X	X	
Decrypt	X	X	X	
Delete Certificate	X	X		X
Delete Key	X	X		X
Encrypt	X	X	X	
Extract X	X		X	
Firmware Update	X	X	X	
Generate IV	X	X	X	
Generate Mek	X	X	X	
Generate Ra	X	X	X	
Generate Random Number	X	X		X
Generate TEK	X	X	X	
Generate X	X	X	X	
Get Certificate	X	X		X
Get Hash	X	X	X	
Get Personality List	X	X		X
Get Status	X	X		X
Get Time	X	X		X
Hash	X	X	X	
Initialize Hash	X	X	X	
Install X	X	X	X	
Load Certificate	X	X		X
Load DSA Parameters	X	X	X	
Load Initialization Values	X		X	
Load Iv	X	X	X	
Load X	X	X	X	
Relay	X	X	X	
Restore	X	X	X	
Save	X	X	X	
Set Key	X	X	X	
Set Mode	X	X	X	
Set Personality	X	X	X	
Set Time	X			X
Sign	X	X	X	
Timestamp	X	X	X	
Unwrap Key	X	X	X	
Verify Signature	X	X	X	
Verify Timestamp	X	X	X	
Wrap Key	X	X	X	
Zeroize	X	X	X	

Figure 4.1 Services vs. Roles vs. Cryptographic Functions Matrix

The cryptographic functions performed by the Fortezza Crypto Card in the processing of each service is summarized in figure 4.1, and described in the following sections.

5.3 Self-test

The PCcard performs self-test immediately upon power-up to insure the integrity of the device. Cryptographic and firmware checks are performed to insure that the device is operating properly prior to communicating with the host computer. Any failures will cause the PCcard to go into a non-operational error state. No authentication of the Crypto-officer or user is required for this service.