Fortezza Cryptologic Interface Programmers Guide

The National Security Agency
Workstation Security Products

January 30, 1996

Revision 1.52
for The Fortezza Crypto Card ICD Revision P1.5
Product Designation: MD4000501-1.52

This document defines the commands of the Fortezza Cryptologic Interface (CI) Library. The CI Library provides the software developer with an interface to the Fortezza Crypto Card (hereafter referred to as the "Card") while isolating the developer from the cryptologic details of the Card.

Table Of Contents

1. Revision 1.51 vs. 1.52

Below is the significant changes from the Fortezza Cryptologic Interface Programmers Guide revision 1.51 vs. revision 1.52.

1. **CI_ChangePIN, CI_CheckPIN, CI_ExtractX, CI_GetPersonalityList, CI_InstallX, CI_LoadCertificate**- Appropriate values have a 1 NULL byte terminator + a 3 byte pad (to assure byte alignment on word boundaries).
2. **CI_Close**- The "**CI_NO_RESET_FLAG**" is replaced by "**CI_NO_LOG_OFF_FLAG**"
3. **CI_DeleteKey**- Note that the function can be called even if the key indexed is empty.
4. **CI_GetConfiguration**- **CI_CONFIG** structure UserRAMSize and LargestBlockSize are "unsigned long" integers. They were just "long."
5. **CI_GetPersonalityList**- **CI_BAD_SIZE** is returned if EntryCount <1. The description of data processing and formatting was clarified.
6. **CI_GetStatus**- The Serial Number of the Crypto Engine Chip is defined as an "Unsigned Character" from an "Array of 8 characters".
7. **CI_GetTime, CI_SetTime**- The last 2 bytes of time are defined as 0x3030.
8. **CI_Hash**- A data length of 0 should not be sent. Data of 0 length should use **CI_GetHash**.
9. **CI_Initialize**- New error was created: **CI_LIB_ALRDY_INIT**, if **CI_Initialize** is called when the CI Library is already initialized.
10. **CI_Lock**- Functional added to CI Library, no change though, to the CIPG.
11. **CI_Open**- The Card does not have to be in the specified socket when that socket is opened. The Card is analyzed during the first call to the Card.
12. **CI_SetConfiguration**- The library can now be used to set the Card's Worst_Case/Typical mode. Also bit 4 controls Fast Slow mode vs. bit 7 and Fast = 1, Slow = 0. This is now consistent with the Fortezza ICD.
13. **CI_Select**- Choosing socket 0 is invalid at all times.
14. **CI_Unlock**- Functional added to CI Library, no change though, to the CIPG.
15. **CI_SRVR_ERROR**- This new error may occur if a UNIX based library has a failure in the interface with the workstation server.
16. Several commands have had **CI_BAD_CARD** added as a return code if the CI Library can not identify the card as a Fortezza Crypto Card.
17. A new section has been added, Section 2.1.3, to describe a number of CI Library errors which may be returned from a function but is not specifically listed in the description of the function.
18. A new section, Section 2.2.7, has been added to describe the Certificate Labeling Specification. This information was part of a separate document, "Certificate Labeling Specification". That document will be discontinued and all the needed information will be in the CIPG and the Implementors Guide.
19. State changes have been added for each function which interacts with the Card.
20. The type of function, Library, Management, Cryptologic is no longer noted in each function description. The functions still can be categorized under those classes, but new diagram further groups the cryptologic functions.
21. Many textual changes throughout the document for added clarity.

2. Scope

2.1 Introduction

The Cryptologic Interface Library (CI Library) provides a "C" based interface to the services of the Card. The CI Library gives applications access to the services of the Card yet abstracts the data formatting, protocols, and PCMCIA interfacing requirements (as defined in the Fortezza Interface Control Document and the PCMCIA PC Card specifications, respectively).
The Card is a PC Card based cryptographic module that implements:

- digital signatures
- public-private key exchanges,
- encryption-decryption processes

The suite of cryptologic algorithms embedded in the Card may also be used to implement:

- key management and
- lower protocols
- Identification and Authentication

2.2 Reference Documents

“Fortezza Crypto Card Interface Control Document”, Revision P1.5, December 22, 1994, NSA X21

“Fortezza Application Implementors Guide”, Revision 1.01, April, 1995, NSA, X22

2.3 Glossary

0x0000	The ANSI C convention for specifying numbers in hexadecimal (base 16). It uses the digits 0 through 9 and A through F for the 16 possible digits. The first character is a 0, followed by an 'x', followed by any number of hexadecimal digits.
API	Application Programmers Interface.
ASN.1	Abstract Syntax Notation One. A data encoding (not encryption) scheme which allows data to be transferred among many diverse computers.
Big Endian	The order in which the bytes of a number are stored in memory where the first byte, the byte with the lowest address, contains the most significant byte of the number.
Block	The amount of data which can be loaded onto the Card for processing (ex., data to hash).
Card, the	The Fortezza Crypto Card.
CAW	Certificate Authority Workstation. Workstation where cards are programmed. It was formerly called the LAW.
Certificate	A certificate contains Public Key information required for performing a key exchange or signature verification operations. The Fortezza Certificate is based on X.509.

Certificate Index	The ordinal value used to access certificates on the Card. The Certificate Index is used to bind a Certificate Label, a Certificate and a set of Private Components together.
Certificate Label	The ASCII/ANSI string that is a human readable alias for a certificate. The Certificate Label is always 32 bytes long.
CIS	Card Information Structure. The information on a PC Card used to define the cards parameters.
Cryptographic Engine	A suite of cryptographic algorithms.
Crypto Engine Chip	The chip that contains and executes the cryptographic algorithms on the Card. Each crypto engine chip has a unique serial number.
Data Block	See “Block.”
Double-word	A double-word is 32 bits (four bytes) long. Objects that have memory addresses that are multiples of four are on Double-word boundaries.
DSA	Digital Signature Algorithm.
E-Mail	Electronic Mail.
Even Word	A value or address where the 2 Least Significant Bits are 00. Examples of 32 bit even word boundary addresses are: 0000 0000h, 0000 0004h, 0000 0008h, etc. All pointers on the Card require the use of 32 bit even word boundaries addressing.
g Parameter	A DSA and KEA value (between 64 and 128 bytes) defined by the SSO.
hex	Short for hexadecimal. The numbering system with a base of 16.
Hash	An algorithm to digest any amount of data to a fixed size.
IV	Initialization Vector. A value used in the encryption/decryption process.
K_S	The User’s Storage Key Variable.
KEA	Key Exchange Algorithm. The algorithm used for electronic public/private key exchange.
Key Register Index	Index parameter to specify use of a temporary key storage register. Valid registers indexes are 0-9.
Key Registers	A set of temporary storage registers inside the crypto engine for storage of encryption keys.
Long	A 32 bit integer value.
Little Endian	The order in which the bytes of a number are stored in memory where the first byte, the byte with the lowest address, contains the least significant byte of the number. Intel i86 line of microprocessors uses little endian.
Manufacturer Default PIN	The special SSO PIN phrase that must be entered for the first logon to the Card.
MEK	Message Encryption Key. Value generated by the Card and used for encryption and decryption.
MSP	Message Security Protocol.

Non-volatile	A memory device that retains its contents when power to the device has been removed.
NULL	Equivalent to zero. A NULL pointer has a value of zero to indicate that it does not point to anything.
p Parameter	The prime modulus used in DSA and KEA exchanges.
PCMCIA	Personal Computer Memory Card International Association.
Personality	Certificates assigned to an individual (e.g. SSO, Self and Department).
PIN Phrase	Personal Identification Number phrase used to log onto the Card.
PandGSize Parameter	The number of bytes in the p parameter and the g parameter.
q Parameter	The Prime Divisor size range in the DSA and KEA exchange
r Parameter	One of the two parameters defining the Digital Signature. The s Parameter is the other parameter.
R_a	The initiators random number generated for use with the Pairwise key exchange.
R_b	The recipients random number involved in a Pairwise key exchange.
s Parameter	One of the two parameters defining the Digital Signature. The r Parameter is the other parameter.
Signature	A value used to authenticate that the data came from a specific author and has not been modified. The signature is composed of two parts: r and s.
Socket	The physical connector that a PC Card is inserted into. The CI Library allows an application to open and select Sockets. All Cryptologic commands are issued to the currently selected Socket.
SSO	Site Security Officer.
SSO Enabled User	Site Security Officer Enabled User. A user logged in as an SSO (using to SSO PIN) can execute card maintenance functions which are denied to a user logged in with the User PIN.
TEK	Token Encryption Key.
Tuple	An information format defined by the PCMCIA specification.
Type 1	A Type 1 PCMCIA card which is 3.0 millimeter thick card.
Type 2	A Type 2 PCMCIA card which is 5.0 millimeter thick card.
Unwrap	A Process where one key is used to decrypt another key.
Wrap	A Process where one key is used to encrypt another key.
X	The Private component used in a DSA or KEA process.
X.509	A specification for a certificate.
Y	The Public component used in a DSA or KEA process.
Zeroize	A process that clears a storage location or device and removes any residual traces of the data.
Zeroize Default PIN	The SSO PIN Phrase required for a Zeroized card.

3. Implementation

3.1 Cryptologic Interface Library

3.1.1 Initialization

Regardless of the application, the CI Library requires the application to prepare the library for operation. The application must initialize the CI Library, and consequently begin a "session", with the **CI_Initialize** function. The **CI_Initialize** function establishes communication with PC Card or Socket Services. It returns the number of Sockets available to the user on the host system. If any of the library's other function are called before initialization, then that function will return the **CI_LIB_NOT_INIT** error.

After the CI Library is initialized and the number of possible sockets known, the application must open the Socket containing the desired card. A Socket is opened with the **CI_Open** function. More than one Socket may be open during a session. The **CI_Select** function is used to select between the open Sockets. The Socket opened in **CI_Open** will also automatically be the selected socket (**CI_Select** is not required immediately after a successful **CI_Open** function).

The **CI_Initialize**, **CI_Open**, and **CI_Select** functions can be performed without the Card inserted in a socket. Upon execution of the first function which accesses the Card, the CI Library will attempt to initialize its internal data structures and reset the Card. The internal data structures holds the configuration information about the Card. This information is necessary for the CI Library to communicate with the Card. The CI Library will reset the Card before the first function is issued, so the Card is in a known state. Note that the reset process may be a lengthy operation, taking several seconds. If the applications first command to be executed is **CI_Reset**, then the Card will be reset twice.

To facilitate the locating of a particular card, the **CI_GetStatus** function returns the **CI_STATUS** data structure which contains the serial number of the crypto engine chip on the Card. The **CI_GetStatus** function may be called before logging on to the Card.

3.1.2 Function Return Values

Every function of the CI Library returns an integer value. The return value specifies whether the function was executed successfully. When a command executes successfully it returns the code **CI_OK** (0x0000). Otherwise the return value is an error code. If the error code is greater than zero (0), then the Card is reporting an error. If the error code is less than zero (0), then the CI Library generated the error. In either case the command is aborted and no data is returned from the function.

3.1.3 General Error Notices

In the description of the functions, in Section 4, there are several error codes which are not listed for any particular function, yet may be returned by the CI Library. The application must potentially be prepared to handle these errors. The following table defines these common errors.

Error(s)	Reason(s)
CI_LIB_NOT_INIT	Will occur for any function called before **CI_Initialize**, or if **CI_Initialize** fails.
CI_ERROR	May occur anytime the CI Library is unstable.
CI_SRVR_ERROR	May occur any time a problem occurs between the Server and the CI Library.
CI_OUT_OF_MEMORY, **CI_BAD_IOCTL**	May occur when the CI Library is initializing.
CI_CARD_NOT_READY, **CI_CARD_IN_USE**, **CI_BAD_CARD**, **CI_BAD_TUPLES**, **CI_NOT_A_CRYPTO_CARD**	May appear on the first function to access the Card.
CI_BAD_READ, **CI_BAD_SEEK**, **CI_BAD_WRITE**, **CI_BAD_FLUSH**, **CI_BAD_IOSEEK**, **CI_BAD_ADDR**	May occur during an access to the Card (including the first function to access the Card).
CI_NO_EXECUTE, **CI_INVALID_FUNCTION**, **CI_OUT_OF_RESOURCES**	May occur on any function.
CI_BAD_PARAMETER	May occur on any function sending or receiving data from the Card.

3.1.4 Data Structure

The data structures of the CI Library are double-word (4 byte) aligned with the exception of when the host machine has a word size less than 4 bytes, then integer values are aligned to the host's word size.

3.2 The Fortezza Crypto Card

The Card contains a crypto engine chip, shared memory, non-volatile memory and working memory. The crypto engine chip contains the cryptographic engine that executes the commands that are generated by the CI Library. The CI Library communicates with the Card via shared memory. The shared memory is made up of the PCMCIA common and attribute memory. The Card uses its non-volatile memory to store the covered PIN phrases, certificates, user data and any Card specific data. The working memory is used by the cryptographic engine for the execution of its own commands.

3.2.1 Inserting The Card

When the Card is inserted it enters the Power Up State and performs its power on self test. The Card then inspects its internal contents to determine what state to transition to. The Card will transition to any state except the Standby or Ready State. It may also detect an internal failure. The current state of the Card may be obtained at any time by calling the **CI_GetState** function.

3.2.2 New Card Initialization

When the initial recipient (SSO Enabled User) receives the Card from the manufacture, the Card will transition to the Uninitialized State and accept only the Manufacturer Default PIN as the SSO PIN phrase. Once the Manufacture Default PIN phrase has been accepted, the SSO Enabled User

must load the initialization parameters with the **CI_LoadInitValues function** to transition the Card into the Initialized State. The SSO Enabled User then sets the SSO PIN phrase and the Card moves into the SSO Initialized State. The SSO Enabled User then must load a Certificate into Certificate Index 0 to transition the Card into the LAW Initialized State. Finally the SSO Enabled User loads the User PIN phrase. The Card is now in the User Initialized State and can have other certificates loaded and given to the user.

3.2.3 Card States

When the user inserts the Card, it will be in the Power Up State and will transition into the user Initialized State where it waits for a PIN phrase. Once a valid PIN phrase has been accepted the Card moves to the Standby State, waiting for the Personality to be set. Setting a Personality transitions the Card into the Ready State. If the user fails to logon to the Card in 10 consecutive tries, the Card will zeroize the User PIN and then transitions to the LAW Initialize State. Then the SSO Enabled User will have to reload the initialization parameters and User PIN phrase. If the SSO Enabled User ever fails to logon to the Card in 10 consecutive attempts, the Card will zeroize all of the certificates, Private Components, Key Registers and disallow User access. When the Card is inserted after a zeroize, it will power up and transition to the Zeroize State, where it will only accept the Zeroize Default PIN phrase. After the Zeroize Default PIN phrase has been accepted the Card transitions to the Uninitialized State and must be reinitialized.

3.2.4 Card/ CI Library Command Execution

The CI Library will construct the command and place it into the shared memory of the Card. The CI Library then writes to a register on the Card that indicates that there is a command in the shared memory to execute. The CI Library then polls the register to determine when the Card has completed execution of the command. When the Card completes the execution of a command the CI Library retrieves the response code of the command. If the command was completed successfully then any data generated by the command is then retrieved.
the Card executes commands independent of the CI Library. The Card will define the longest time it needs to execute a command (this value is stored in the Attribute Memory of the Card). If a command has not completed after this amount of time, then CI Library will assume that the Card may be faulty and will return the error code **CI_TIME_OUT**. It is up to the application to reset the Card or request action from the user.

3.2.5 Accessing Key Registers

The Card defines a set of Key Registers within its working memory. These registers are used to store keys and are accessed by their Register Index. The Card normally contains 10 Key Registers with the indexes 0 through 9. The **CI_GetConfiguration** function will return the actual number of Key Registers on the Card. Key Register zero (0) is used to hold the Card's Storage Key (K_S). The user may not delete, generate, load or unwrap a key into Key Register zero (0). After a key in a Key Register has been used it must be cleared with the **CI_DeleteKey** function before another key can be loaded into the register. Failure to clear a Key Register will result in the function returning the **CI_REG_IN_USE** error.

3.2.6 Accessing Certificates

The certificates stored in the non-volatile memory of the Card are referenced by their Certificate Index. To determine the maximum number of certificates that the Card can hold, use the **CI_GetConfiguration** function. The **CI_GetStatus** function will return the number of certificates currently loaded on the Card. The Certificate Index is an ordinal value in the range of

zero (0) to the Certificate Count that is returned by **CI_GetConfiguration**. Note that Certificate Index zero (0) can only be written to by the SSO Enabled User. The Certificate Index is used to associate a Certificate Label (Personality String), a certificate, and a public key pair for both DSA and KEA. The **CI_SetPersonality** function is used to set the current personality of the Card. The **CI_GetPersonalityList** will return the list of personalities on the Card. Note that the Certificate at index 0 will not be returned in the personality list.

3.2.7 Certificate Labeling Specification

The Personality String is a 32 byte value. The bytes are defined as three (3) fields: Usage/Equipment Specifier (4 bytes, bytes 1-4), Parent Certificate (4 bytes, bytes 5-8), and ASCII Label Field (24 bytes, bytes 9-32).

Usage/ Equipment Specifiers: The Convention for this field is uppercase ASCII characters. The NSA currently has defined 9 Usage/ Equipment Specifiers:

INKS- Individual X.509 Certificate with the signature (DSS) and key exchange (KEA) keys.
INKX- Individual X.509 Certificate with KEA key, but no signature (DSS) key.
ONKS- Organizational X.509 Certificate with the signature (DSS) and key exchange (KEA) keys.
ONKX- Organizational X.509 Certificate with KEA key, but no signature (DSS) key.
RRXX- Root Registry/ PAA X.509 Certificate.
RTXX- Root/PCA X.509 Certificate.
LAXX- Local Authority/CA X.509 Certificate.
STXX- Secure Telephone Unit CIK Data.
FAXX- FAX data.
MCXX- Mobile Certificate cache (defined in Fortezza Certification Requirements for Electronic Mail Applications)

Developers are not bound to these Specifiers, though should register/reserve any new specifiers with the NSA to avoid possible confusion with other applications.

The Parent Certificate is a 4 byte number. It can be used to for certificate authentication hierarchy. An application uses these values to determine an X.509 certificate's Issuer.

The ASCII Field can be uppercase or lowercase alphanumeric characters.

The Fortezza Application Implementors Guide provides examples of these fields.
The host application must parse the Personality String to determine if the Certificate Index contains a valid personality, as other applications may store non-certificate data in the certificates. A Personality must be set to use the cryptologic functions of the Card that require a public key pair. The current personality may be changed at any time during the session.

4. Function Formats

4.1 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.

4.2 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:

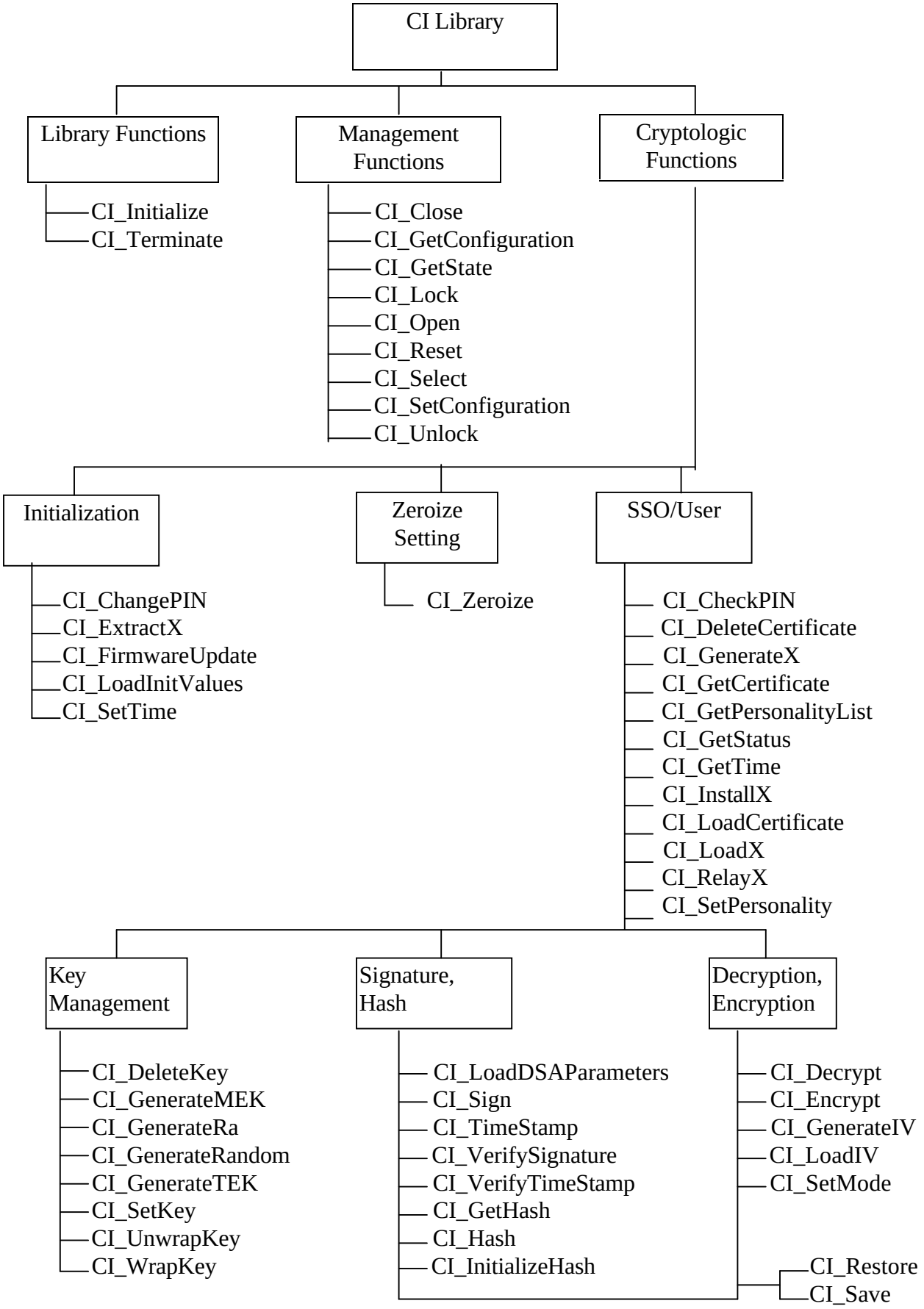
PIN[0] = 'C'; PIN[1] = 'R'; PIN[2] = 'Y'; PIN[3] = 'P'; PIN[4] = 'T'; PIN[5] = 'O';
PIN[6] = ‘G’; PIN[7] = ‘R’; PIN[8] = ‘A’; PIN[9] = ‘P’; PIN[10] = ‘H’; PIN[11] = ‘Y’

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

4.3 **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.

4.4 Function Organization



5. Function Descriptions

5.1 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. Also, both the CI Library and the Card require an old PIN value even if the PIN has never been set (there is no old value).

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

CI_ChangePIN: [PINType, pOldPIN, pNewPIN]
{return value}

int CI_ChangePIN(int PINType, CI_PIN pOldPIN, CI_PIN pNewPIN)

Parameter	Type	Description
PINType	int	The type of PIN phrase to change from the list above.
pOldPIN	**CI_PIN**	Points to a original NULL terminated PIN phrase.
pNewPIN	**CI_PIN**	Points to a new NULL terminated PIN phrase.
return value	int	The function's completion code.

Value	Meaning
CI_OK	*The PIN phrase was changed.*
CI_FAIL	*PIN phrase was not changed.*
CI_INV_TYPE	*The PIN Type is invalid.*
CI_INV_STATE	*This function may not be executed in this Card state.*
CI_EXEC_FAIL	*The command failed to execute.*
CI_INV_POINTER	*This card detected an invalid pointer.*
CI_TIME_OUT	The Card failed to complete the command.
CI_NULL_PTR	The pointer to a PIN phrase is NULL.
CI_NO_CARD	The Card was not found.
CI_NO_SOCKET	A Socket has not been opened.

Entry State	Exit State
Initialized	SSO Initialized when SSO PIN is changed from Default
SSO Initialized	SSO Initialized
LAW Initialized	User Initialized when User PIN is changed. SSO is logged out.
User Initialized	User Initialized

5.2 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.

CI_CheckPIN: [PINType, pPIN]
{return value}

int CI_CheckPIN(int PINType, CI_PIN pPIN)

Parameter	Type	Description
PINType	int	The type of PIN phrase to check from the list above.
PIN	**CI_PIN**	Points to a the NULL terminated PIN phrase to check.
return value	int	The function's completion code.

Value	Meaning
CI_OK	*The PIN phrase is correct.*
CI_FAIL	*PIN phrase does not match.*
CI_INV_TYPE	*The PIN Type is invalid.*
CI_INV_STATE	*This function may not be executed in this Card state.*
CI_EXEC_FAIL	*The command failed to execute.*
CI_INV_POINTER	*This card detected an invalid pointer.*
CI_TIME_OUT	The Card failed to complete the command.
CI_NULL_PTR	The pointer to the PIN phrase is NULL.
CI_NO_CARD	The Card was not found.
CI_NO_SOCKET	A Socket has not been opened.

Entry State	Exit State
Uninitialized-Type = SSO using SSO Default PIN	Uninitialized Logged out if PIN fails
Zeroized-Type = SSO using Zeroize PIN	Uninitialized and Logged Out
Initialized -Type = SSO using SSO Default PIN	Initialized Logged out if PIN fails
SSO Initialized-Type = SSO	SSO Initialized Logged out if PIN fails
LAW Initialized-Type = SSO	LAW Initialized Logged out if PIN fails
User Initialized-Type = SSO	User Initialized Logged out if PIN fails
User Initialized-Type = USER	Standby User initialized if PIN fails
Standby-Type = SSO	User Initialized Logged out if PIN fails
Standby-Type = User	Standby User initialized if PIN fails
Ready-Type = SSO	User Initialized Logged out if PIN fails
Ready-Type = User	Ready User Initialized if PIN fails
Any state and fail SSO PIN 10 times	Zeroized
User Initialized, Ready or Standby and fail User PIN 10 times	LAW Initialized

5.3 CI_Close

The **CI_Close** function closes the specified Socket.

Flags	Description
CI_NULL_FLAG	No options. The Card is reset by default.
CI_POWER_DOWN_FLAG	The Socket should be powered down after it is closed.
CI_NO_LOG_OFF_FLAG	The Socket (the Card) should not be reset after it is closed.

CI_Close: [Flags, SocketIndex]
{return value}

int CI_Close(unsigned int Flags, int SocketIndex)

Parameter	Type	Description
Flags	unsigned int	Function options.
SocketIndex	int	The index of the Socket to close.
return value	int	The function's completion code.

Value	Meaning
CI_OK	The Socket was successfully closed.
CI_INV_SOCKET_INDEX	The Socket Index is invalid.
CI_SOCKET_NOT_OPENED	The specified Socket is not open.

5.4 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.

CI_Decrypt: [CipherSize, pCipher]
{pPlain, return value}

int CI_Decrypt(unsigned int CipherSize, CI_DATA pCipher, CI_DATA pPlain)

Parameter	Type	Description
CipherSize	unsigned int	The number of bytes to decrypt from pCipher and in the buffer pointed to by pPlain.
pCipher	**CI_DATA**	Points to the data to be decrypted.
pPlain	**CI_DATA**	Points to the buffer, of CipherSize bytes, where the decrypted data will be stored.
return value	int	The function's completion code.

Value	Meaning
CI_OK	*The data was successfully decrypted.*
CI_INV_SIZE	*An invalid Data Size was specified.*
CI_INV_STATE	*This function may not be executed in this Card state.*
CI_EXEC_FAIL	*The command failed to execute.*
CI_NO_KEY	*The key has not been loaded.*
CI_NO_IV	*The IV has not been loaded.*
CI_INV_POINTER	*This card detected an invalid pointer.*
CI_TIME_OUT	The Card failed to complete the command.
CI_NULL_PTR	The pointer to the ciphertext is NULL.
CI_NO_DECRYPT	The Card failed to decrypt the data.
CI_NO_CARD	The Card was not found.
CI_BAD_CARD	The Card is invalid.
CI_NO_SOCKET	A Socket has not been opened.

Entry State	Exit State
Standby	Standby
Ready	Ready

5.5 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.

CI_DeleteCertificate: [CertificateIndex]
{return value}

int CI_DeleteCertificate(int CertificateIndex)

Parameter	Type	Description
CertificateIndex	int	Specifies the Certificate to zeroize.
return value	int	The function's completion code.

Value	Meaning
CI_OK	*The Certificate was successfully deleted.*
CI_INV_CERT_INDEX	*The Certificate Index is invalid.*
CI_INV_STATE	*This function may not be executed in this Card state.*
CI_INV_POINTER	*This card detected an invalid pointer.*
CI_TIME_OUT	The Card failed to complete the command.
CI_NO_CARD	The Card was not found.
CI_NO_SOCKET	A Socket has not been opened.

Entry State	Exit State
LAW Initialized	LAW Initialized SSO Initialized if certificate index 0 is deleted
User Initialized	User Initialized SSO Initialized if certificate index 0 is deleted
Standby	Standby
Ready	Ready

5.6 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.

CI_DeleteKey: [RegisterIndex]
{return value}

int CI_DeleteKey(int RegisterIndex)

Parameter	Type	Description
RegisterIndex	int	Specifies the Key Register to zeroize.
return value	int	The function's completion code.

Value	Meaning
CI_OK	*The key was successfully deleted.*
CI_INV_KEY_INDEX	*The Register Index is invalid.*
CI_INV_STATE	*This function may not be executed in this Card state.*
CI_EXEC_FAIL	*The command failed to execute.*
CI_INV_POINTER	*This card detected an invalid pointer.*
CI_TIME_OUT	The Card failed to complete the command.
CI_NO_CARD	The Card was not found.
CI_BAD_CARD	The Card is invalid.
CI_NO_SOCKET	A Socket has not been opened.

Entry State	Exit State
Standby	Standby
Ready	Ready

5.7 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.

CI_Encrypt: [PlainSize, pPlain]
{pCipher, return value}

int CI_Encrypt(unsigned int PlainSize, CI_DATA pPlain, CI_DATA pCipher)

Parameter	Type	Description
PlainSize	unsigned int	The number of bytes to encrypt and the number of bytes in the buffer pointed to by pCipher.
pPlain	**CI_DATA**	Points to the data to be encrypted.
pCipher	**CI_DATA**	Points to the buffer, of PlainSize bytes, where the encrypted data will be placed.
return value	int	The function's completion code.

Value	Meaning
CI_OK	***The data was successfully encrypted.***
CI_INV_SIZE	***An invalid Data Size was specified.***
CI_INV_STATE	***This function may not be executed in this Card state.***
CI_EXEC_FAIL	***The command failed to execute.***
CI_NO_KEY	*The key has not been loaded.*
CI_NO_IV	*The IV has not been loaded.*
CI_INV_POINTER	*This card detected an invalid pointer.*
CI_TIME_OUT	The Card failed to complete the command.
CI_NULL_PTR	The pointer to the plaintext is NULL.
CI_NO_ENCRYPT	The Card failed to encrypt the data.
CI_NO_CARD	The Card was not found.
CI_BAD_CARD	The Card is invalid.
CI_NO_SOCKET	A Socket has not been opened.

Entry State	Exit State
Standby	Standby
Ready	Ready

5.8 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.

Algorithm Types	Description
CI_DSA_TYPE	Extract the Private Component for the DSA algorithm.
CI_KEA_TYPE	Extract the Private Component for the KEA algorithm.

CI_ExtractX: [CertificateIndex, AlgorithmType, pPassword, YSize, pY, PandGSize, QSize] {pWrappedX, pRa, pP, pQ, pG, return value}

int CI_ExtractX(int CertificateIndex, int AlgorithmType, CI_PASSWORD pPassword, unsigned int YSize, CI_Y pY, CI_WRAPPED_X pX, CI_RA pRA, unsigned int PandGSize, unsigned int QSize, CI_P pP, CI_Q pQ, CI_G pG)

Parameter	Type	Description
CertificateIndex	int	The Certificate Index from which to extract the Private Component (X) from.
AlgorithmType	int	Specifies which Private Component (X) to extract either: **CI_DSA_TYPE** or **CI_KEA_TYPE**.
pPassword	**CI_PASSWORD**	Points to the buffer that contains the 'user' supplied password.
YSize	unsigned int	The number of bytes in the buffer pointed to by pY.
pY	**CI_Y**	Pointer to the buffer that contains the Public Component (Y) of the recipient.
pX	**CI_WRAPPED_X**	Points to the buffer that will receive the extracted, wrapped, Private Component (X).
pRa	**CI_RA**	Points to the buffer that will receive the Ra used to generate the TEK.
PandGSize	unsigned int	The number of bytes in the buffers pointed to by pP and pG.

QSize	unsigned int	The number of bytes in the buffer pointed to by pQ.
pP	**CI_P**	Pointer to the buffer that will receive the p parameter.
pQ	**CI_Q**	Pointer to the buffer that will receive the q parameter.
pG	**CI_G**	Pointer to the buffer that will receive the g parameter.
return value	int	The function's completion code.

Value	Meaning
CI_OK	*The X was successfully extracted.*
CI_INV_TYPE	*Invalid data type.*
CI_INV_CERT_INDEX	*Invalid certificate index.*
CI_INV_STATE	*This function may not be executed in this Card state.*
CI_EXEC_FAIL	*The command failed to execute.*
CI_NO_X	*There is no X.*
CI_INV_POINTER	*This card detected an invalid pointer.*
CI_NO_SOCKET	A Socket has not been opened.
CI_TIME_OUT	The Card failed to complete the command.
CI_NULL_PTR	A required pointer is NULL.
CI_BAD_SIZE	The size of pP, pQ or pG is too small.
CI_NO_CARD	The Card was not found.
CI_BAD_CARD	The Card is invalid.

Entry State	Exit State
LAW Initialized	LAW Initialized
User Initialized	User Initialized

5.9 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

5.10 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.

CI_GenerateIV: [none]
{pIV, return value}

int CI_GenerateIV(CI_IV pIV)

Parameter	Type	Description
pIV	**CI_IV**	Pointer to the buffer that will receive the 192 bit (24 byte) IV.
return value	int	The function's completion code.

Value	Meaning
CI_OK	*The IV was successfully generated.*
CI_INV_STATE	*This function may not be executed in this Card state.*
CI_NO_KEY	*There is no key.*
CI_INV_POINTER	*This card detected an invalid pointer.*
CI_TIME_OUT	The Card failed to complete the command.
CI_NULL_PTR	The pointer to the IV is NULL.
CI_NO_CARD	The Card was not found.
CI_BAD_CARD	The Card is invalid.
CI_NO_SOCKET	A Socket has not been opened.

Entry State	Exit State
Standby	Standby
Ready	Ready

5.11 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.

CI_GenerateMEK: [RegisterIndex, Reserved]
{return value}

int CI_GenerateMEK(int RegisterIndex, int Reserved)

Parameter	Type	Description
RegisterIndex	int	Specifies the Key Register that will receive the MEK.
Reserved	int	Reserved. Must be set to zero (0).
return value	int	The function's completion code.

Value	Meaning
CI_OK	*The MEK was successfully generated.*
CI_CHECKWORD_FAIL	*The checkword did not match.*
CI_INV_KEY_INDEX	*The Register Index is invalid.*
CI_INV_STATE	*This function may not be executed in this Card state.*
CI_EXEC_FAIL	*The command failed to execute.*
CI_REG_IN_USE	*The Key Register is already in use.*
CI_INV_POINTER	*This card detected an invalid pointer.*
CI_TIME_OUT	The Card failed to complete the command.
CI_NO_CARD	The Card was not found.
CI_BAD_CARD	The Card is invalid.
CI_NO_SOCKET	A Socket has not been opened.

Entry State	Exit State
Standby	Standby
Ready	Ready

5.12 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.

CI_GenerateRa: [none]
{pRa, return value}

int CI_GenerateRa(CI_RA pRa)

Parameter	Type	Description
pRa	**CI_RA**	Points to the buffer that will receive the R_a.
return value	int	The function's completion code.

Value	Meaning
CI_OK	*The R_a was successfully generated.*
CI_CHECKWORD_FAIL	*Checkword test on the Key or Private Component failed.*
CI_INV_STATE	*This function may not be executed in this Card state.*
CI_EXEC_FAIL	*The command failed to execute.*
CI_NO_X	*There is no X.*
CI_INV_POINTER	*This card detected an invalid pointer.*
CI_TIME_OUT	The Card failed to complete the command.
CI_NULL_PTR	The pointer to the Ra is NULL.
CI_NO_CARD	The Card was not found.
CI_BAD_CARD	The Card is invalid.
CI_NO_SOCKET	A Socket has not been opened.

Entry State	Exit State
Ready	Ready

5.13 CI_GenerateRandom

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

CI_GenerateRandom: [none]
{pRandom, return value}

int CI_GenerateRandom(CI_RANDOM pRandom)

Parameter	Type	Description
pRandom	**CI_RANDOM**	Points to the buffer that will receive the 160 bit (20 byte) random number.

return value	int	The function's completion code.

Value	Meaning
CI_OK	*The Random number was successfully generated.*
CI_INV_POINTER	*This card detected an invalid pointer.*
CI_TIME_OUT	The Card failed to complete the command.
CI_NULL_PTR	The pointer to the random number 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
Initialized	Initialized
SSO Initialized	SSO Initialized
LAW Initialized	LAW Initialized
User Initialized	User Initialized
Standby	Standby
Ready	Ready

5.14 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:

```
CI_RB   Rb;                     /* set Rb to big endian 1 */
memset( Rb, 0, sizeof( Rb ) );  /* set Rb to zero */
Rb[127] = 0x01;                 /* set bit 1 to one */
```

Flags	Description
CI_INITIATOR_FLAG	This command is used to initiate an exchange.
CI_RECIPIENT_FLAG	This command is used as the recipient of an exchange.

CI_GenerateTEK: [Flags, RegisterIndex, Ra, Rb, YSize, pY]
{return value}

int CI_GenerateTEK(int Flags, int RegisterIndex, CI_RA Ra, CI_RB Rb, unsigned int YSize, CI_Y pY)

Parameter	Type	Description
Flags	int	Options for generating the TEK from the list above.
RegisterIndex	int	Index of the Key Register to place the TEK into.
pRa	**CI_RA**	Pointer to the initiator generated Ra.
pRb	**CI_RB**	Pointer to the recipient generated Rb.
YSize	unsigned int	The number of bytes in the buffer pointed to by pY.
pY	**CI_Y**	Pointer to the other parties Public Component (Y).

return value int The function's completion code.

Value	Meaning
CI_OK	*The command succeeded.*
CI_CHECKWORD_FAIL	*The checkword did not match.*
CI_INV_KEY_INDEX	*Invalid key index.*
CI_INV_STATE	*This function may not be executed in this Card state.*
CI_EXEC_FAIL	*The command failed to execute.*
CI_NO_X	*There is no X.*
CI_REG_IN_USE	*The register is in use.*
CI_INV_POINTER	*This card detected an invalid pointer.*
CI_TIME_OUT	The Card failed to complete the command.
CI_NULL_PTR	A pointer is NULL.
CI_NO_CARD	The Card was not found.
CI_BAD_CARD	The Card is invalid.
CI_NO_SOCKET	A Socket has not been opened.

Entry State	Exit State
Ready	Ready

5.15 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.

Flags	Description
CI_DSA_TYPE	Generate the public key pair for DSA.
CI_KEA_TYPE	Generate the public key pair for KEA.
CI_DSA_KEA_TYPE	Generate the same public key pair for both DSA and KEA.

CI_GenerateX: [CertificateIndex, AlgorithmType, PandGSize, QSize, pP, pQ, pG, YSize] {pY, return value}

int CI_GenerateX(int CertificateIndex, int AlgorithmType, unsigned int PandGSize, unsigned int QSize, CI_P pP, CI_Q pQ, CI_G pG, unsigned int YSize, CI_Y pY)

Parameter	Type	Description
CertificateIndex	int	Specifies which Certificate Index to associate the X with. Certificate Indexes begin with one (1).
AlgorithmType	int	The algorithm type of public key pair to generate.
PandGSize	unsigned int	The number of bytes in the buffers pointed to by pP and pG.
Qsize	unsigned int	The number of bytes in the buffer pointed to by pQ.
pP	**CI_P**	Points to the buffer that contains the p parameter.
pQ	**CI_Q**	Points to the buffer that contains the q parameter.
pG	**CI_G**	Points to the buffer that contains the g parameter.
Ysize	unsigned int	The number of bytes in the buffer pointed to by pY.
pY	**CI_Y**	Points to the buffer to receive the 1024 bit (128 byte) Y of the X that was generated.
return value	int	The function's completion code.

Value	Meaning
CI_OK	*The public key pair was successfully generated.*
CI_INV_TYPE	*The Algorithm Type is invalid.*

CI_INV_CERT_INDEX *The Certificate Index is invalid.*
CI_INV_SIZE *The data size is invalid.*
CI_INV_STATE *This function may not be executed in this Card state.*
CI_EXEC_FAIL *The command failed to execute.*
CI_INV_POINTER *This card detected an invalid pointer.*
CI_TIME_OUT The Card failed to complete the command.
CI_NULL_PTR The pointer to the Y is NULL.
CI_NO_CARD The Card was not found.
CI_BAD_CARD The Card is invalid.
CI_NO_SOCKET A Socket has not been opened.

Entry State	Exit State
SSO Initialized	SSO Initialized
LAW Initialized	LAW Initialized
User Initialized	User Initialized
Standby	Standby
Ready	Ready

5.16 CI_GetCertificate

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

CI_GetCertificate : [CertificateIndex]
{pCertificate, return value}

int CI_GetCertificate(int CertificateIndex, CI_CERTIFICATE pCertificate)

Parameter	Type	Description
CertificateIndex	int	The Certificate Index of the certificate to retrieve. The first Certificate Index is zero (0).
pCertificate	**CI_CERTIFICATE**	Points to the buffer that will receive the certificate.
return value	int	The function's completion code.

Value	Meaning
CI_OK	*The Certificate was successfully retrieved.*
CI_INV_CERT_INDEX	*The Certificate Index is invalid.*
CI_INV_STATE	*This function may not be executed in this Card state.*
CI_INV_POINTER	*This card detected an invalid pointer.*
CI_TIME_OUT	The Card failed to complete the command.
CI_NULL_PTR	The pointer to the certificate is NULL.
CI_NO_CARD	The Card was not found.
CI_NO_SOCKET	A Socket has not been opened.

Entry State	Exit State
LAW Initialized	LAW Initialized
User Initialized	User Initialized
Standby	Standby
Ready	Ready

5.17 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.

5.18 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.

CI_GetHash: [DataSize, pData]
{pHashValue, return value}

int CI_GetHash(unsigned int DataSize, CI_DATA pData, CI_HASHVALUE pHashValue)

Parameter	Type	Description
DataSize	unsigned int	The number of bytes in pData.
pData	**CI_DATA**	Pointer to the data to hash.
pHashValue	**CI_HASHVALUE**	Points to the buffer that will receive the 160 bit (20 byte) Hash Value
return value	int	The function's completion code.

Value	Meaning
CI_OK	*The Hash Value was successfully retrieved.*
CI_INV_STATE	*This function may not be executed in this Card state.*
CI_INV_POINTER	*This card detected an invalid pointer.*
CI_TIME_OUT	The Card failed to complete the command.
CI_NULL_PTR	The pointer to the Hash Value is NULL.
CI_NO_CARD	The Card was not found.
CI_NO_SOCKET	A Socket has not been opened.

Entry State	Exit State
Standby	Standby
Ready	Ready

5.19 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
1	1	Root
2	2	CAW
3	3	User
4	4	<NULL String>
5	5	Another Personality
6	0	<NULL String>
7	0	<NULL String>

CI_GetPersonalityList: [EntryCount]
{pPersonalityList, return value}

int CI_GetPersonalityList(int EntryCount, CI_PERSON pPersonalityList[ ])

Parameter	Type	Description
EntryCount	int	Indicates the number of entries in the Personality List pointed to by pPersonalityList.
pPersonalityList	**CI_PERSON**	Points to the array of **CI_PERSON** that will receive the Personality List.
return value	int	The function's completion code.

Value	Meaning
CI_OK	*The Personality List was successfully retrieved.*
CI_INV_STATE	*This function may not be executed in this Card state.*
CI_INV_POINTER	*This card detected an invalid pointer.*
CI_TIME_OUT	The Card failed to complete the command.
CI_NULL_PTR	The pointer to Personality List is NULL.
CI_BAD_SIZE	If EntryCount < 1
CI_NO_CARD	The Card was not found.
CI_NO_SOCKET	A Socket has not been opened.

Entry State	Exit State
LAW Initialized	LAW Initialized
User Initialized	User Initialized
Standby	Standby
Ready	Ready

5.20 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.

Card States	Description
CI_POWER_UP	The Card is waiting for a PIN phrase to be entered.
CI_UNINITIALIZED	The Card is uninitialized or has been zeroized and the Zeroize Default PIN has been entered.
CI_INITIALIZED	The initialization parameters have been loaded.
CI_SSO_INITIALIZED	The SSO PIN has been loaded.
CI_LAW_INITIALIZED	The User's certificates and Private Components have been loaded.
CI_USER_INITIALIZED	The User PIN has been loaded.
CI_STANDBY	The Card is waiting for the Personality to be set.
CI_READY	The Card is ready to be used.
CI_ZEROIZE	The Card has been zeroized.
CI_INTERNAL_FAILURE	An internal error has been detected.

CI_GetState: [none]
{pState, return value}

int CI_GetState(CI_STATE_PTR pState)

Parameter	Type	Description
pState	**CI_STATE_PTR**	Points to the buffer that will receive the state.

return value	int	The function's completion code.

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

5.21 CI_GetStatus

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

CI_STATUS_PTR A pointer to void.

CI_STATUS A structure containing:

Field Name	Data Type	Description
CurrentSocket	int	The currently selected socket.
LockState	int	The lock status of the current socket.
SerialNumber	**CI_SERIAL_NUMBER**	The serial number of the Crypto Engine Chip.
CurrentState	**CI_STATE**	The state of the Card.
Decryption Mode	int	The mode of the Card.
Encryption Mode	int	The mode of the Card.
CurrentPersonality	int	The index of the current personality.
KeyRegisterCount	int	Count of Key Registers on the Card.
KeyRegisterFlags	**CI_REG_FLAGS**	A set of bit fields indicating register use.
CertificateCount	int	Count of certificates on the Card.
CertificateFlags	**CI_CERT_FLAGS**	A set of bit fields indicating certificate use.
Flags[4]	unsigned char	Flags[0] Bit 4. Clock speed: Fast = 1, Slow =0. Bit 6. Condition: Typical = 0, Worst Case = 1. All other bits and bytes are reserved. Use CI_SetConfiguration to set conditions.

Lock States	Description
CI_SOCKET_UNLOCKED	The socket is not locked.
CI_HOLD_LOCK	The application holds the lock to this socket.
CI_SOCKET_LOCKED	The socket is locked and therefore unavailable.

The flags fields are bit maps of the usage of the particular resource. The first item is represented by the high order bit of the first byte, the second item the second highest bit of the first byte and so on.

CI_GetStatus: [none]
{pStatus, return value}

int CI_GetStatus(CI_STATUS_PTR pStatus)

Parameter	Type	Description
pStatus	**CI_STATUS_PTR**	Points to the buffer that will receive the status information.

return value	int	The function's completion code.

Value	Meaning
CI_OK	*The Status was successfully retrieved.*
CI_INV_POINTER	*This card detected an invalid pointer.*
CI_NULL_PTR	A pointer 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
Zeroized	Zeroized
Initialized	Initialized
SSO Initialized	SSO Initialized
LAW Initialized	LAW Initialized
User Initialized	User Initialized
Standby	Standby
Ready	Ready

5.22 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.

CI_TIME An array of 16 characters.

The format of the **CI_TIME** data structure is YYYYMMDDhhmmssxx.

YYYY	Four digits of Year.
MM	Two digits of Month, '01' to '12'.
DD	Two digits of Day, '01' to '07'.
hh	Two digits of Hour, '00' to '24'.
mm	Two digits of Minutes, '00' to '60'.
ss	Two digits of Seconds, '00' to '60'.
xx	Two reserved digits, all '00' (0x3030)

CI_GetTime: [none]
{pTime, return value}

int CI_GetTime(CI_TIME pTime)

Parameter	Type	Description
pTime	**CI_TIME**	Points to the buffer that will receive the current time.

return value int The function's completion code.

Value	Meaning
CI_OK	*The Time was successfully retrieved.*
CI_INV_POINTER	*This card detected an invalid pointer.*
CI_BAD_CLOCK	*The clock failed.*
CI_TIME_OUT	The Card failed to complete the command.
CI_NULL_PTR	The pointer to Time 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
Initialized	Initialized
SSO Initialized	SSO Initialized
LAW Initialized	LAW Initialized
User Initialized	User Initialized
Standby	Standby
Ready	Ready

5.23 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.

CI_Hash: [DataSize, pData]
{return value}

int CI_Hash(unsigned int DataSize, CI_DATA pData)

Parameter	Type	Description
DataSize	unsigned int	The number of bytes to be hashed.
pData	**CI_DATA**	Points to the data to be hashed.
return value	int	The function's completion code.

Value	Meaning
CI_OK	*The Data was successfully hashed.*
CI_INV_SIZE	*Size is not a multiple of 64.*
CI_INV_STATE	*This function may not be executed in this Card state.*
CI_INV_POINTER	*This card detected an invalid pointer.*
CI_TIME_OUT	The Card failed to complete the command.
CI_NULL_PTR	The pointer to the data is NULL.
CI_NO_CARD	The Card was not found.
CI_NO_SOCKET	A Socket has not been opened.

Entry State	Exit State
Standby	Standby
Ready	Ready

5.24 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.

CI_Initialize: [none]
{SocketCount, return value}

int CI_Initialize(int *SocketCount)

Parameter	Type	Description
SocketCount	int *	Points to an integer that will receive the number of Sockets on the host system.

return value	int	The function's completion code.

Value	Meaning
CI_OK	The CI Library was successfully initialized.
CI_NO_DRIVER	The Cryptologic Engine Driver was not found.
CI_NO_CRDSRV	The PCMCIA Card Services were not found.
CI_NO_SCTSRV	The PCMCIA Socket Services were not found.
CI_LIB_ALRDY_INIT	The CI Library was already initialized.

5.25 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**.

CI_InitializeHash: [none]
{return value}

int CI_InitializeHash(void)

Parameter	Type	Description
return value	int	The function's completion code.

Value	Meaning
CI_OK	*The Hash Value was successfully initialized.*
CI_INV_STATE	*This function may not be executed in this Card state.*
CI_TIME_OUT	The Card failed to complete the command.
CI_NO_CARD	The Card was not found.
CI_NO_SOCKET	A Socket has not been opened.

Entry State	Exit State
Standby	Standby
Ready	Ready

5.26 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.

CI_InstallX: [CertificateIndex, AlgorithmType, pPassword, YSize, pY, pWrappedX, pRa, PandGSize, Qsize, pP, pQ, pG]
{return value}

int CI_InstallX(int CertificateIndex, int AlgorithmType, CI_PASSWORD pPassword, unsigned int YSize, CI_Y pY, CI_WRAPPED_X pWrappedX, CI_RA pRa, unsigned int PandGSize, unsigned int QSize, CI_P pP, CI_Q pQ, CI_G pG)

Parameter	Type	Description
CertificateIndex	int	Specifies the Certificate Index of the X.
AlgorithmType	int	Specifies the algorithm type of X.
pPassword	**CI_PASSWORD**	Points to a 'user' supplied password.
YSize	unsigned int	Number of bytes in pY.
pY	**CI_Y**	Points to the Users Public Component, Y.
pWrappedX	**CI_WRAPPED_X**	Points to the buffer that contains the covered Private Component, X.
pRa	**CI_RA**	Points to the buffer that contains the Ra.
PandGSize	unsigned int	The number of bytes in the buffers pP and pG.
QSize	unsigned int	The number of bytes in the buffer pointed to by pQ.
pP	**CI_P**	Pointer to the buffer that contains the p parameter.
pQ	**CI_Q**	Pointer to the buffer that contains the q parameter.

pG	**CI_G**	Pointer to the buffer that contains the g parameter.
return value	int	The function's completion code.

Value	Meaning
CI_OK	*The Private Component was successfully installed.*
CI_INV_CERT_INDEX	*Invalid certificate index.*
CI_INV_TYPE	*Invalid type.*
CI_INV_SIZE	*Invalid data size.*
CI_INV_STATE	*This function may not be executed in this Card state.*
CI_INV_POINTER	*This card detected an invalid pointer.*
CI_TIME_OUT	The Card failed to complete the command.
CI_NULL_PTR	A pointer is NULL.
CI_NO_CARD	The Card was not found.
CI_BAD_CARD	The Card is invalid.
CI_NO_SOCKET	A Socket has not been opened.

Entry State	Exit State
LAW Initialized	LAW Initialized
User Initialized	User Initialized
Standby	Standby
Ready	Ready

5.27 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).

Note that Certificate Index zero (0) is reserved for the SSO Enabled User.

CI_LoadCertificate:[CertificateIndex, pLabel, pCertificate, Reserved]
{return value}

int CI_LoadCertificate (int CertificateIndex, CI_CERT_STR pLabel, CI_CERTIFICATE pCertificate, long Reserved)

Parameter	Type	Description
CertificateIndex	int	Certificate Index of the certificate being loaded.
pLabel	**CI_CERT_STR**	Points to the NULL terminator Certificate Label.
pCertificate	**CI_CERTIFICATE**	Points the certificate.
Reserved	long	Reserved. Should be set to zero (0).
return value	int	The function's completion code.

Value	Meaning
CI_OK	*The Certificate was successfully loaded.*
CI_INV_CERT_INDEX	*CertificateIndex is not a valid Certificate Index.*
CI_INV_STATE	*This function may not be executed in this Card state.*
CI_EXEC_FAIL	*The command failed during execution.*
CI_INV_POINTER	*This card detected an invalid pointer.*
CI_TIME_OUT	The Card failed to complete the command.
CI_NULL_PTR	The pointer to Certificate is NULL.
CI_NO_CARD	The Card was not found.
CI_NO_SOCKET	A Socket has not been opened.

Entry State	Exit State
SSO Initialized	LAW Initialized (if Certificate Index 0 is selected by an SSO Enabled User)
LAW Initialized	LAW Initialized
User Initialized	User Initialized
Standby	Standby
Ready	Ready

5.28 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.

CI_LoadDSAParameters: [PandGSize, QSize, pP, pQ, pG]
{return value}

int CI_LoadDSAParameters(unsigned int PandGSize, unsigned int QSize, CI_P pP, CI_Q pQ, CI_G pG)

Parameter	Type	Description
PandGSize	unsigned int	The number of bytes in the buffers pP and pG.
QSize	unsigned int	The number of bytes in the buffer pQ.
pP	**CI_P**	Points to the p parameter.
pQ	**CI_Q**	Points to the q parameter.
pG	**CI_G**	Points to the g parameter.
return value	int	The function's completion code.

Value	Meaning
CI_OK	*The DSA Parameters were successfully loaded.*
CI_INV_SIZE	*The p, g or q size is invalid.*
CI_INV_STATE	*This function may not be executed in this Card state.*
CI_INV_POINTER	*This card detected an invalid pointer.*
CI_TIME_OUT	The Card failed to complete the command.
CI_NULL_PTR	A pointer is NULL.
CI_NO_CARD	The Card was not found.
CI_NO_SOCKET	A Socket has not been opened.

Entry State	Exit State
Standby	Standby
Ready	Standby

5.29 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.

CI_LoadInitValues: [pRandSeed, pKs]
{return value}

int CI_LoadInitValues(CI_RANDSEED pRandSeed, CI_KS pKs)

Parameter	Type	Description
pRandSeed	**CI_RANDSEED**	Points to a 64 bits (8 bytes) Random Number Seed.
pKs	**CI_KS**	Points to an 80 bits (10 bytes) K_S.
return value	int	The function's completion code.

Value	Meaning
CI_OK	*The Initialization Values where successfully loaded.*
CI_INV_STATE	*This function may not be executed in this Card state.*
CI_INV_POINTER	*This card detected an invalid pointer.*
CI_TIME_OUT	The Card failed to complete the command.
CI_NULL_PTR	The pointer to RandSeed or K_S is NULL.
CI_NO_CARD	The Card was not found.
CI_BAD_CARD	The Card is invalid.
CI_NO_SOCKET	A Socket has not been opened.

Entry State	Exit State
Unitialized	Initialized

5.30 CI_LoadIV

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

CI_LoadIV: [pIV]
{return value}

int CI_LoadIV(CI_IV pIV)

Parameter	Type	Description
pIV	**CI_IV**	Points to a 192 bit (24 byte) IV.

return value	int	The function's completion code.

Value	Meaning
CI_OK	*The IV was successfully loaded.*
CI_INV_STATE	*This function may not be executed in this Card state.*
CI_EXEC_FAIL	*The command failed during execution.*
CI_NO_KEY	*There is no key.*
CI_INV_POINTER	*This card detected an invalid pointer.*
CI_TIME_OUT	The Card failed to complete the command.
CI_NULL_PTR	The pointer to the IV is NULL.
CI_NO_CARD	The Card was not found.
CI_BAD_CARD	The Card is invalid.
CI_NO_SOCKET	A Socket has not been opened.

Entry State	Exit State
Standby	Standby
Ready	Ready

5.31 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.

Algorithm Types	Description
CI_DSA_TYPE	Load an X for DSA.
CI_KEA_TYPE	Load an X for KEA.
CI_DSA_KEA_TYPE	Load an X for DSA and KEA. Note that the same p, q, and g values are used for both DSA and KEA.

CI_LoadX: [CertificateIndex, AlgorithmType, PandGSize, QSize, pP, pQ, pG, pX, YSize] {pY, return value}

int CI_LoadX(int CertificateIndex, int AlgorithmType, unsigned int PandGSize, unsigned int QSize, CI_P pP, CI_Q pQ, CI_G pG, CI_X pX, unsigned int YSize, CI_Y pY)

Parameter	Type	Description
CertificateIndex	int	The Certificate Index to bind the X to.
AlgorithmType	int	The algorithm type of the X to load.
PandGSize	unsigned int	The number of bytes in the buffers pP and pG.
QSize	unsigned int	The number of bytes in the buffer pQ.
pP	**CI_P**	Points to the buffer that contains the p parameter.
pQ	**CI_Q**	Points to the buffer that contains the q parameter.
pG	**CI_G**	Points to the buffer that contains the g parameter.
pX	**CI_X**	Points to a buffer that contains an X value.
YSize	unsigned int	The number of bytes in the buffer pY.
pY	**CI_Y**	Points to a buffer that will receive the Y.

return value int The function's completion code.

Value	Meaning
CI_OK	*The X was successfully loaded.*
CI_INV_TYPE	*The Algorithm Type is invalid.*
CI_INV_CERT_INDEX	*The Certificate Index is invalid.*
CI_INV_SIZE	*Invalid data size.*
CI_INV_STATE	*This function may not be executed in this Card state.*
CI_EXEC_FAIL	*The command failed to execute.*
CI_INV_POINTER	*This card detected an invalid pointer.*
CI_TIME_OUT	The Card failed to complete the command.
CI_NULL_PTR	A pointer is NULL.
CI_NO_CARD	The Card was not found.
CI_BAD_CARD	The Card is invalid.
CI_NO_SOCKET	A Socket has not been opened.

Entry State	Exit State
SSO Initialized	SSO Initialized
LAW Initialized	LAW Initialized
User Initialized	User Initialized
Standby	Standby
Ready	Ready

5.32 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.

Flags	Description
CI_NULL_FLAG	No options.
CI_BLOCK_LOCK_FLAG	If the socket is currently locked, then wait until the socket can be locked for this session before returning. This option may not be available on all platforms. If the function is not supported the CI Library will return the error code **CI_SOCKET_IN_USE**.

CI_Lock: [Flags]
{return value}

int CI_Lock(int Flags)

Parameter	Type	Description
Flags	int	Options for locking the socket.
return value	int	The function's completion code.

Value	Meaning
CI_OK	The Socket was successfully locked.
CI_SOCKET_IN_USE	The Socket is already being used.
CI_NO_SOCKET	A Socket has not been opened.

5.33 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.

Flags	Description
CI_NULL_FLAG	No options.

Use the **CI_Close** function to close the communication link to the Socket.

CI_Open: [Flags, SocketIndex]
{return value}

int CI_Open(unsigned int Flags, int SocketIndex)

Parameter	Type	Description
Flags	unsigned int	Options for opening the Socket.
SocketIndex	int	The index of the Socket to open.
return value	int	The function's completion code.

Value	Meaning
CI_OK	The Socket was successfully opened or is already opened.
CI_INV_SOCKET_INDEX	The Socket Index is invalid.
CI_SOCKET_IN_USE	The Socket is already being used.
CI_SOCKET_NOT_OPENED	The Socket failed to open.

5.34 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.

CI_RelayX: [pOldPassword, OldYSize, pOldY, pOldRa, pOldWrappedX, pNewPassword, NewYSize, pNewY]
{pNewRa, pNewWrappedX, return value}

int CI_RelayX(CI_PASSWORD pOldPassword, unsigned int OldYSize, CI_Y pOldY, CI_RA pOldRa, CI_WRAPPED_X pOldWrappedX, CI_PASSWORD pNewPassword, unsigned int NewYSize, CI_Y pNewY, CI_RA pNewRa, CI_WRAPPED_X pNewWrappedX)

Parameter	Type	Description
pOldPassword	**CI_PASSWORD**	Points to the original ‘user’ supplied password.
OldYSize	unsigned int	The number of byte in the buffer pOldY.
pOldY	**CI_Y**	Points to the original Public Component, Y.
pOldRa	**CI_RA**	Points to the original Ra.
pOldWrappedX	**CI_WRAPPED_X**	Points to the original wrapped X.
pNewPassword	**CI_PASSWORD**	Points to the new ‘user’ supplied password.
NewYSize	unsigned int	Number of byte in the buffer pNewY.
pNewY	**CI_Y**	Points to the new Public Component, Y.
pNewRa	**CI_RA**	Points to the new Ra.
pNewWrappedX	**CI_WRAPPED_X**	Points to the new wrapped X.

return value int The function's completion code.

Value	Meaning
CI_OK	*The X was successfully relayed.*
CI_INV_STATE	*The state was not valid for this command.*
CI_EXEC_FAIL	*The command failed to execute.*
CI_INV_POINTER	*This card detected an invalid pointer.*
CI_TIME_OUT	The Card failed to complete the command.
CI_NULL_PTR	A pointer is NULL.
CI_NO_CARD	The Card was not found.
CI_BAD_CARD	The Card is invalid.
CI_NO_SOCKET	A Socket has not been opened.

Entry State	Exit State
LAW Initialized	LAW Initialized
User Initialized	User Initialized
Ready	Ready

5.35 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.

Use the **CI_Reset** function to log off of the Card.

CI_Reset: [none]
{return value}

int CI_Reset(void)

Parameter	Type	Description
return value	int	The function's completion code.

Value	Meaning
CI_OK	The Card was successfully reset.
CI_NO_SOCKET	A Socket has not been opened.

5.36 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.

Cryptologic Types	Description
CI_DECRYPT_EXT_TYPE	Restore the decryption state from memory buffer.
CI_DECRYPT_INT_TYPE	Restore the decryption state from the Card's storage.
CI_ENCRYPT_EXT_TYPE	Restore the encryption state from memory buffer.
CI_ENCRYPT_INT_TYPE	Restore the encryption state from the Card's storage.
CI_HASH_EXT_TYPE	Restore the hash state from memory buffer.
CI_HASH_INT_TYPE	Restore the hash state from the Card's storage.

CI_Restore: [CryptoType, pData]
{return value}

int CI_Restore(int CryptoType, CI_SAVE_DATA pData)

Parameter	Type	Description
CryptoType	int	The type cryptographic state to save.
pData	**CI_SAVE_DATA**	Pointer to the data to restore.
return value	int	The function's completion code.

Value	Meaning
CI_OK	*The cryptologic state was successfully restored.*
CI_INV_TYPE	*Invalid data type.*
CI_INV_MODE	*Invalid mode.*
CI_INV_STATE	*This function may not be executed in this Card state.*
CI_EXEC_FAIL	*The command failed to execute.*
CI_NO_KEY	*No key.*
CI_NO_SAVE	*No state has been saved for the mode specified.*
CI_INV_POINTER	*This card detected an invalid pointer.*
CI_TIME_OUT	The Card failed to complete the command.
CI_NULL_PTR	A pointer is NULL.
CI_NO_CARD	The Card was not found.
CI_BAD_CARD	The Card is invalid.
CI_NO_SOCKET	A Socket has not been opened.

Entry State	Exit State
Standby	Standby
Ready	Ready

5.37 **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,

a. **CI_Save** (CI_HASH_INT_TYPE, NULL);
b. **CI_Save**(CI_HASH_EXT_TYPE, pSave);
c. **CI_Restore**(CI_HASH_INT_TYPE, NULL);

Step c will restore the state from b, not step a.

Cryptologic Types	Description
CI_DECRYPT_EXT_TYPE	Save the decryption state to a memory buffer.
CI_DECRYPT_INT_TYPE	Save the decryption state to the Crypto-Card's storage.
CI_ENCRYPT_EXT_TYPE	Save the encryption state to a memory buffer.
CI_ENCRYPT_INT_TYPE	Save the encryption state to the Crypto-Card's storage.
CI_HASH_EXT_TYPE	Save the hash state to a memory buffer.
CI_HASH_INT_TYPE	Save the hash state to the Crypto-Card's storage.

CI_Save: [CryptoType]
{pData, return value}

int CI_Save(int CryptoType, CI_SAVE_DATA pData)

Parameter	Type	Description
CryptoType	int	The type of cryptographic state to save.
pData	**CI_SAVE_DATA**	Pointer to the buffer that will receive the Save data.
return value	int	The function's completion code.

Value	Meaning
CI_OK	*The cryptographic state was successfully saved.*
CI_INV_TYPE	*Invalid data type.*
CI_INV_STATE	*This function may not be executed in this Card state.*
CI_NO_KEY	*No key.*
CI_INV_POINTER	*This card detected an invalid pointer.*
CI_TIME_OUT	The Card failed to complete the command.
CI_NULL_PTR	The pointer to the memory buffer is NULL.
CI_NO_CARD	The Card was not found.
CI_BAD_CARD	The Card is invalid.
CI_NO_SOCKET	A Socket has not been opened.

Entry State	Exit State
Standby	Standby
Ready	Ready

5.38 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**.

CI_Select: [SocketIndex]
{return value}

int CI_Select(int SocketIndex)

Parameter	Type	Description
SocketIndex	int	The index of the Socket to select.
return value	int	The function's completion code.

Value	Meaning
CI_OK	The socket was successfully selected.
CI_INV_SOCKET_INDEX	The Socket Index is invalid.

5.39 CI_SetConfiguration

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

Configuration Type	Description
CI_SET_SPEED_TYPE	Sets the speed of the Crypto Engine chip. DataSize must equal the size of an integer: sizeof(int). pData must point to an integer variable and the variable must be set to either: **CI_FAST_MODE** or **CI_SLOW_MODE**.

An example in C to set the Card to slow (low power) mode:

```
int  speed = CI_SLOW_MODE;
CI_SetConfiguration( CI_SET_SPEED_TYPE, sizeof( speed ), (CI_DATA)&speed );
```

CI_SET_TIMING_TYPE	Sets the Worst_Case/ Typical flag on the Card. The DataSize must equal the size of an integer: sizeof(int). pData must point to an integer variable and the variable must be set to either: **CI_WORST_CASE_MODE** or **CI_TYPICAL_CASE_MODE**. Timing to Worst_Case indicates to the Card to run with worst case memory access timing (this condition can be due to extreme environmental conditions).

An example in C to set the Worst-Case condition:

```
int  mode = CI_WORST_CASE_MODE;
CI_SetConfiguration( CI_SET_TIMING_TYPE, sizeof( mode ), (CI_DATA)&mode);
```

CI_SetConfiguration: [Type, DataSize, pData]
{return value}

int CI_SetConfiguration(int Type, unsigned int DataSize, CI_DATA pData)

Parameter	Type	Description
Type	int	The type of configuration setting to make.
DataSize	unsigned int	The number of bytes in the buffer pData.
pData	**CI_DATA**	Points to any configuration data.
return value	int	The function's completion code.

Value	Meaning
CI_OK	The Configuration was successfully set.
CI_INV_TYPE	The Type is invalid.

5.40 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.

CI_SetKey: [RegisterIndex]
{return value}

int CI_SetKey(int RegisterIndex)

Parameter	Type	Description
RegisterIndex	int	Indicates which of the Key Registers to use.
return value	int	The function's completion code.

Value	Meaning
CI_OK	*The Key was successfully set.*
CI_CHECKWORD_FAIL	*The checkword did not match.*
CI_INV_KEY_INDEX	*The Register Index is invalid.*
CI_INV_STATE	*This function may not be executed in this Card state.*
CI_EXEC_FAIL	*The command failed to execute.*
CI_NO_KEY	*No key.*
CI_TIME_OUT	The Card failed to complete the command.
CI_NO_CARD	The Card was not found.
CI_BAD_CARD	The Card is invalid.
CI_NO_SOCKET	A Socket has not been opened.

Entry State	Exit State
Standby	Standby
Ready	Ready

5.41 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.

Cryptologic Types	Description
CI_DECRYPT_TYPE	Set the mode for subsequent decryption processes.
CI_ENCRYPT_TYPE	Set the mode for subsequent encryption processes.

Cryptologic Modes	Description	Block Size Multiple	
CI_ECB64_MODE	64 bit ECB	64 bits	
CI_CBC64_MODE	64 bit CBC	64 bits	(default)
CI_OFB64_MODE	64 bit OFB	64 bits	
CI_CFB64_MODE	64 bit CFB	64 bits	
CI_CFB32_MODE	32 bit CFB	32 bits	
CI_CFB16_MODE	16 bit CFB	32 bits	
CI_CFB8_MODE	8 bit CFB	32 bits	

CI_SetMode: [CryptoType, CryptoMode]
{return value}

int CI_SetMode(int CryptoType, int CryptoMode)

Parameter	Type	Description
CryptoType	int	The type of cryptographic operation to set the mode for.
CryptoMode	int	Indicates which cryptologic mode to set.
return value	int	The function's completion code.

Value	Meaning
CI_OK	*The Cryptologic Mode was successfully set.*
CI_INV_TYPE	*The Cryptologic Type is invalid.*
CI_INV_MODE	*The Cryptologic Mode is invalid.*
CI_INV_STATE	*This function may not be executed in this Card state.*
CI_INV_POINTER	*This card detected an invalid pointer.*
CI_TIME_OUT	The Card failed to complete the command.
CI_NO_CARD	The Card was not found.
CI_BAD_CARD	The Card is invalid.
CI_NO_SOCKET	A Socket has not been opened.

Entry State	Exit State
Standby	Standby
Ready	Ready

5.42 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.

CI_SetPersonality: [CertificateIndex]
{return value}

int CI_SetPersonality(int CertificateIndex)

Parameter	Type	Description
CertificateIndex	int	The Certificate Index to set the Personality to.
return value	int	The function's completion code.

Value	Meaning
CI_OK	*The Personality was successfully set.*
CI_INV_CERT_INDEX	*The Personality Index is invalid.*
CI_INV_STATE	*This function may not be executed in this Card state.*
CI_NO_X	*The Certificate does not have a Private Component.*
CI_INV_POINTER	*This card detected an invalid pointer.*
CI_TIME_OUT	The Card failed to complete the command.
CI_NO_CARD	The Card was not found.
CI_NO_SOCKET	A Socket has not been opened.

Entry State	Exit State
LAW Initialized	LAW Initialized
User Initialized	User Initialized
Standby	Ready
Ready	Ready

5.43 CI_SetTime

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

Note that setting the time to all zero (0) will stop the clock:

```
CI_TIME  sTime;
memset( &sTime, 0, sizeof( sTime ) );
CI_SetTime( &sTime );
```

CI_TIME An array of 16 characters.
The format of the **CI_TIME** data structure is YYYYMMDDhhmmssxx:

YYYY	Four digits of Year.
MM	Two digits of Month, ‘01’ to ‘12’.
DD	Two digits of Day, ‘01’ to ‘07’.
hh	Two digits of Hour, ‘00’ to ‘24’.
mm	Two digits of Minutes, ‘00’ to ‘60’.
ss	Two digits of Seconds, ‘00’ to ‘60’.
xx	Two reserved digits, all ‘00’ (0x3030).

CI_SetTime : [pTime]
{return value}

int CI_SetTime(CI_TIME pTime)

Parameter	Type	Description
pTime	**CI_TIME**	Points to the buffer that contains the new time and date value.

return value	int	The function's completion code.

Value	Meaning
CI_OK	*The Time was successfully set.*
CI_INV_STATE	*This function may not be executed in this Card state.*
CI_EXEC_FAIL	*The command failed to execute.*
CI_INV_POINTER	*This card detected an invalid pointer.*
CI_BAD_CLOCK	*The clock failed.*
CI_TIME_OUT	The Card failed to complete the command.
CI_NULL_PTR	The pointer to the Time 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
Initialized	Initialized
SSO Initialized	SSO Initialized
LAW Initialized	LAW Initialized
User Initialized	User Initialized

5.44 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.

CI_Sign: [pHashValue]
{pSignature, return value}

int CI_Sign(CI_HASHVALUE pHashValue, CI_SIGNATURE pSignature)

Parameter	Type	Description
pHashValue	**CI_HASHVALUE**	Points to the 160 bit (20 byte) Hash Value.
pSignature	**CI_SIGNATURE**	Points to the buffer that will receive 320 bit (40 byte) signature.
return value	int	The function's completion code.

Value	Meaning
CI_OK	*The Hash Value was successfully signed.*
CI_INV_STATE	*This function may not be executed in this Card state.*
CI_EXEC_FAIL	*The command failed to execute.*
CI_NO_X	*There is no X.*
CI_INV_POINTER	*This card detected an invalid pointer.*
CI_TIME_OUT	The Card failed to complete the command.
CI_NULL_PTR	The pointer to Hash Value or Signature is NULL.
CI_NO_CARD	The Card was not found.
CI_NO_SOCKET	A Socket has not been opened.

Entry State	Exit State
Ready	Ready

5.45 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.

Note that C**I_Terminate** calls the **CI_Close** function with the **CI_NULL_FLAG**, which resets the Card, for each open socket.

CI_Terminate: [none]
{return value}

int CI_Terminate(void)

Parameter	Type	Description
return value	int	The function's completion code.

Value	Meaning
CI_OK	The CI Library was successfully terminated.

5.46 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.

CI_TimeStamp: [pHashValue]
{pSignature, pTimeStamp, return value}

int CI_TimeStamp(CI_HASHVALUE pHashValue, CI_SIGNATURE pSignature, CI_TIMESTAMP pTimeStamp)

Parameter	Type	Description
pHashValue	**CI_HASHVALUE**	Points to the 160 bit (20 byte) Hash Value.
pSignature	**CI_SIGNATURE**	Points to the buffer that will receive the time stamped signature.
pTimeStamp	**CI_TIMESTAMP**	Points to the buffer that will receive the time stamp.
return value	int	The function's completion code.

Value	Meaning
CI_OK	*The Time stamp was successfully generated.*
CI_INV_STATE	*This function may not be executed in this Card state.*
CI_EXEC_FAIL	*The command failed to execute.*
CI_INV_POINTER	*This card detected an invalid pointer.*
CI_BAD_CLOCK	*The clock failed.*
CI_TIME_OUT	The Card failed to complete the command.
CI_NULL_PTR	A pointer is NULL.
CI_NO_CARD	The Card was not found.
CI_NO_SOCKET	A Socket has not been opened.

Entry State	Exit State
Standby	Standby
Ready	Ready

5.47 CI_Unlock

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

CI_Unlock: [none]
{return value}

int CI_Unlock(void)

Parameter	Type	Description
return value	int	The function's completion code.

Value	Meaning
CI_OK	The Socket was successfully opened.
CI_SOCKET_IN_USE	This application does not hold the lock to the socket.
CI_NO_SOCKET	A Socket has not been opened.

5.48 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.

Note that the key may not be unwrapped into Key Register zero (0).

CI_UnwrapKey: [UnwrapIndex, KeyIndex, pKey]
{return value}

int CI_UnwrapKey(int UnwrapIndex, int KeyIndex, CI_KEY pKey)

Parameter	Type	Description
UnwrapIndex	int	Indicates which Key Register contains the key to unwrap the wrapped key with.
KeyIndex	int	Indicates which Key Register will hold the unwrapped key.
pKey	**CI_KEY**	Points to a 96 bit (12 byte) wrapped key.
return value	int	The function's completion code.

Value	Meaning
CI_OK	*The Key was successfully unwrapped.*
CI_CHECKWORD_FAIL	*The checkword did not match.*
CI_INV_KEY_INDEX	*The Unwrap Index or Key Index is invalid.*
CI_INV_STATE	*This function may not be executed in this Card state.*
CI_EXEC_FAIL	*The command failed to execute.*
CI_NO_KEY	*There is no key.*
CI_REG_IN_USE	*The Key Register specified by KeyIndex in already in use.*
CI_INV_POINTER	*This card detected an invalid pointer.*
CI_TIME_OUT	The Card failed to complete the command.
CI_NULL_PTR	The pointer to Key is NULL.
CI_NO_CARD	The Card was not found.
CI_BAD_CARD	The Card is invalid.
CI_NO_SOCKET	A Socket has not been opened.

Entry State	Exit State
Standby	Standby
Ready	Ready

5.49 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.

CI_VerifySignature: [pHashValue, YSize, pY, pSignature]
{ return value}

int CI_VerifySignature (CI_HASHVALUE pHashValue, unsigned int YSize, CI_Y pY, CI_SIGNATURE pSignature)

Parameter	Type	Description
pHashValue	**CI_HASHVALUE**	Points to a buffer that contains 160 bit (20 byte) Hash Value.
YSize	unsigned int	The number of bytes in the buffer pY.
pY	**CI_Y**	Points to the buffer that contains the 1024 bit (128 byte) Y.
pSignature	**CI_SIGNATURE**	Points to the buffer that will contains 320 bit (40 byte) Signature.
return value	int	The function's completion code.

Value	Meaning
CI_OK	*The Signature was successfully verified.*
CI_FAIL	*The Signature is invalid.*
CI_INV_STATE	*This function may not be executed in this Card state.*
CI_EXEC_FAIL	*The command failed to execute.*
CI_INV_POINTER	*This card detected an invalid pointer.*
CI_NO_DSA_PARMS	*No DSA parameters (p, q, and g).*
CI_TIME_OUT	The Card failed to complete the command.
CI_NULL_PTR	The pointer to Y or Signature is NULL.
CI_NO_CARD	The Card was not found.
CI_NO_SOCKET	A Socket has not been opened.

Entry State	Exit State
Standby	Standby
Ready	Ready

5.50 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)

Parameter	Type	Description
pHashValue	**CI_HASHVALUE**	Points to the 160 bit (20 byte) Hash Value to time stamp.
pSignature	**CI_SIGNATURE**	Points to the buffer that contains the time stamped Signature.
pTimeStamp	**CI_TIMESTAMP**	Points to the buffer that contains the time stamp.
return value	int	The function's completion code.

Value	Meaning
CI_OK	*The Time stamp was verified.*
CI_FAIL	*The Time stamp did not verify.*
CI_INV_STATE	*This function may not be executed in this Card state.*
CI_EXEC_FAIL	*The command failed to execute.*
CI_INV_POINTER	*This card detected an invalid pointer.*
CI_TIME_OUT	The Card failed to complete the command.
CI_NULL_PTR	A pointer is NULL.
CI_NO_CARD	The Card was not found.
CI_NO_SOCKET	A Socket has not been opened.

Entry State	Exit State
Standby	Standby
Ready	Ready

5.51 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.

Note that Key Register zero (0), the storage key, K_S, of the Card, may not be wrapped.

CI_WrapKey: [WrapIndex, KeyIndex]
{pKey, return value}

int CI_WrapKey(int WrapIndex, int KeyIndex, CI_KEY pKey)

Parameter	Type	Description
WrapIndex	int	Indicates which Key Register contains the key which will be used to wrap the plaintext key.
KeyIndex	int	Indicates which Key Register contains the plaintext key to be wrapped.
pKey	**CI_KEY**	Points to the buffer that will receive the 96 bit (12 byte) wrapped key.
return value	int	The function's completion code.

Value	Meaning
CI_OK	*The Data was successfully wrapped.*
CI_INV_KEY_INDEX	*The Wrap Index or Key Index is invalid.*
CI_INV_STATE	*This function may not be executed in this Card state.*
CI_EXEC_FAIL	*The command failed to execute.*
CI_NO_KEY	*There is no key.*
CI_INV_POINTER	*This card detected an invalid pointer.*
CI_TIME_OUT	The Card failed to complete the command.
CI_NULL_PTR	The pointer to Key is NULL.
CI_NO_CARD	The Card was not found.
CI_BAD_CARD	The Card is invalid.
CI_NO_SOCKET	A Socket has not been opened.

Entry State	Exit State
Standby	Standby
Ready	Ready

5.52 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.

CI_Zeroize: [none]
{return value}

int CI_Zeroize(void)

Parameter	Type	Description
return value	int	The function's completion code.

Value	Meaning
CI_OK	***The Card was successfully zeroized.***
CI_TIME_OUT	The Card failed to complete the command.
CI_NO_CARD	The Card was not found.
CI_NO_SOCKET	A Socket has not been opened.

Entry State	Exit State
Uninitialized	Zeroized
Initialized	Zeroized
SSO Initialized	Zeroized
LAW Initialized	Zeroized
User Initialized	Zeroized
Standby	Zeroized
Ready	Zeroized

6. Parameters

6.1 Constants

Constant	Value in Decimal	Value in Hexadecimal	Description
Sizes			
CI_CERT_SIZE	2048	0800	Size of a Certificate
CI_CERT_NAME_SIZE	32	0020	Size of a Certificate Label without NULL terminator and padding.
CI_NAME_SIZE	32	0020	Size of a name without NULL terminator and padding.
CI_PASSWORD_SIZE	24	0018	Size of a Password without NULL terminator and padding.
CI_PIN_SIZE	12	000C	Size of a PIN phrase without NULL terminator and padding.
Miscellaneous			
CI_NULL_FLAG	0	0000	Flag that does not specify any options.
CI_POWER_DOWN_FLAG	2	0002	Flag to power the Card down.
CI_NO_LOG_OFF_FLAG	4	0004	Flag not to logoff (reset) the Card.
CI_INITIATOR_FLAG	0	0000	Flag to initiate an exchange.
CI_RECIPIENT_FLAG	1	0001	Flag to indicate the recipient of an exchange.
CI_SLOW_MODE	0	0000	The Card uses a slow clock.
CI_FAST_MODE	128	0080	The Card uses a normal (fast) clock.
CI_TYPICAL_CASE_MODE	0	0000	The Card runs in Typical mode.
CI_WORST_CASE_MODE	64	0040	The Card runs in Worst Case mode.
Algorithm Types			
CI_DSA_TYPE	10	000A	Specifies Digital Signature Algorithm (DSA).
CI_KEA_TYPE	5	0005	Specifies Key Exchange Algorithm (KEA).
CI_DSA_KEA_TYPE	15	000F	Specifies both DSA and KEA.
PIN Types			
CI_SSO_PIN	37	0025	Specifies SSO PIN phrase.
CI_USER_PIN	42	002A	Specifies User PIN phrase.
Configuration Types			
CI_SET_SPEED_TYPE	1	0001	Set the speed of the processor.
CI_SET_TIMING_TYPE	2	0002	Set the Worst case/Typical mode
Crypto Types			
CI_DECRYPT_TYPE	1	0001	Specifies Decryption.
CI_ENCRYPT_TYPE	0	0000	Specifies Encryption.
CI_HASH_TYPE	2	0002	Specifies Hash.

Save and Restore Types			
CI_DECRYPT_EXT_TYPE	17	0011	Save or restore decryption state externally.
CI_DECRYPT_INT_TYPE	1	0001	Save or restore decryption state internally.
CI_ENCRYPT_EXT_TYPE	16	0010	Save or restore encryption state externally.
CI_ENCRYPT_INT_TYPE	0	0000	Save or restore encryption state internally.
CI_HASH_EXT_TYPE	18	0012	Save or restore hash state externally.
CI_HASH_INT_TYPE	2	0002	Save or restore hash state internally.

Lock States			
CI_SOCKET_UNLOCKED	0	0000	The socket is not locked.
CI_HOLD_LOCK	3	0003	The application holds the lock to this socket.
CI_SOCKET_LOCKED	1	0001	The socket is locked and therefore unavailable.

Crypto Type Modes			
CI_ECB64_MODE	0	0000	64 bit ECB
CI_CBC64_MODE	1	0001	64 bit Cipher Block Chaining (CBC)
CI_OFB64_MODE	2	0002	64 bit OFB
CI_CFB64_MODE	3	0003	64 bit CFB
CI_CFB32_MODE	4	0004	32 bit CFB
CI_CFB16_MODE	5	0005	16 bit CFB
CI_CFB8_MODE	6	0006	8 bit CFB

Engine States			
CI_POWER_UP	0	0000	The Card is powering up.
CI_UNINITIALIZED	1	0001	The Card is uninitialized.
CI_INITIALIZED	2	0002	The Card is initialized.
CI_SSO_INITIALIZED	3	0003	The Card is SSO initialized.
CI_LAW_INITIALIZED	4	0004	The Card is LAW initialized.
CI_USER_INITIALIZED	5	0005	The Card is User initialized.
CI_STANDBY	6	0006	The Card is waiting for a personality to be selected.
CI_READY	7	0007	The Card is ready for User use.
CI_ZEROIZE	8	0008	The Card has been zeroized.
CI_INTERNAL_FAILURE	-1	FFFF	The Card has detected an internal failure.

Firmware Update Flags:	Hexadecimal	Description
CI_NOT_LAST_BLOCK_FLAG	0000 0000	This is not the last block to be loaded
CI_LAST_BLOCK_FLAG	8000 0000	This is the last block to be loaded
CI_DESTRUCTIVE_FLAG	0000 00FF	Zero non-volatile memory
CI_NONDESTRUCTIVE_FLAG	0000 FF00	Do not zero non-volatile memory

Constant	Value in Decimal	Value in Hexadecimal	Description
Crypto Engine Return Codes			
CI_OK	0	0000	The function successfully completed.
CI_FAIL	1	0001	The function failed.
CI_CHECKWORD_FAIL	2	0002	A Checkword failure occurred.
CI_INV_TYPE	3	0003	An invalid type was specified.
CI_INV_MODE	4	0004	An invalid mode was specified.
CI_INV_KEY_INDEX	5	0005	An invalid Key Register Index was specified.
CI_INV_CERT_INDEX	6	0006	An invalid Certificate Index was specified.
CI_INV_SIZE	7	0007	An invalid Data Size was specified.
CI_INV_HEADER	8	0008	An invalid header was created.
CI_INV_STATE	9	0009	The function may not be performed in this Card State.
CI_EXEC_FAIL	10	000A	The function failed to execute.
CI_NO_KEY	11	000B	A key has not been loaded or generated.
CI_NO_IV	12	000C	An IV has not been loaded or generated.
CI_NO_X	13	000D	An X has not been loaded or generated.
CI_NO_SAVE	15	000F	A Save has not been performed for the Type specified.
CI_REG_IN_USE	16	0010	The Key Register specified is already in use.
CI_INV_COMMAND	17	0011	Invalid command.
CI_INV_POINTER	18	0012	The Card detected an invalid pointer.
CI_BAD_CLOCK	19	0013	The clock failed.
CI_NO_DSA_PARMS	20	0014	There are no DSA parameters (p, q, g).
CI_MAX_ERROR	21	0015	The greatest error code value.
CI Library Errors			
CI_ERROR	-1	FFFF	The CI Library is unstable.
CI_LIB_NOT_INIT	-2	FFFE	The CI Library has not been initialized.
CI_CARD_NOT_READY	-3	FFFD	The Card is not ready.
CI_CARD_IN_USE	-4	FFFC	Another application is using the Card.
CI_TIME_OUT	-5	FFFB	Execution of a command timed out.
CI_OUT_OF_MEMORY	-6	FFFA	Allocation of memory failed, assume out of memory.
CI_NULL_PTR	-7	FFF9	A require pointer is NULL.
CI_BAD_SIZE	-8	FFF8	A Size is invalid, too small, too large, not a multiple of.
CI_NO_DECRYPT	-9	FFF7	The Card failed to decrypt the data.
CI_NO_ENCRYPT	-10	FFF6	The Card failed to encrypt the data.
CI_NO_EXECUTE	-11	FFF5	The Card failed to execute the command.
CI_BAD_PARAMETER	-12	FFF4	A parameter was not valid.
CI_OUT_OF_RESOURCES	-13	FFF3	The CI Library is out of system resources.
CI_NO_CARD	-20	FFEC	The Card is not present.
CI_NO_DRIVER	-21	FFEB	The Cryptologic Engine Driver has not been loaded.
CI_NO_CRDSRV	-22	FFEA	PCMCIA Card Services is not loaded.
CI_NO_SCTSRV	-23	FFE9	PCMCIA Socket Services is not loaded.
CI_BAD_CARD	-30	FFE2	The Card is invalid or malfunctioning.
CI_BAD_IOCTL	-31	FFE1	IOCTL call failed.

CI_BAD_READ	-32	FFE0	Failed to read data from the Card via the driver.
CI_BAD_SEEK	-33	FFDF	Failed to seek to a location on the Card via the driver.
CI_BAD_WRITE	-34	FFDE	Failed to write data to the Card via the driver.
CI_BAD_FLUSH	-35	FFDD	Failed to flush data to the Card via the driver.
CI_BAD_IOSEEK	-36	FFDC	Failed an IOCTL Seek to the Card via the driver.
CI_BAD_ADDR	-37	FFDB	The Card address is invalid
CI_INV_SOCKET_INDEX	-40	FFD8	Invalid Socket Index
CI_SOCKET_IN_USE	-41	FFD7	The specified socket is in use.
CI_NO_SOCKET	-42	FFD6	There are no open Sockets.
CI_SOCKET_NOT_OPENED	-43	FFD5	The Socket failed to open.
CI_BAD_TUPLES	-44	FFD4	The format of the tuples in Card memory is invalid.
CI_NOT_A_CRYPTO_CARD	-45	FFD3	The Card in the Socket is not a Crypto Card.
CI_INVALID_FUNCTION	-50	FFCD	The Card did not recognize the command code.
CI_LIB_ALRDY_INIT	-51	FFCC	The CI Library has already been initialized.
CI_SRVR_ERROR	-52	FFCB	There was an error with the server.
CI_MIN_ERROR	-60	FFC4	The smallest error code value.

6.2 Data Structures

Constant	Data Type
CI_CERTIFICATE	An array of **CI_CERT_SIZE** (2048) characters.
CI_CERT_FLAGS	An array of **CI_CERT_FLAGS_SIZE** (16) characters.
CI_CERT_STR	An array of **CI_CERT_NAME_SIZE** + 4 (36) characters.
CI_CONFIG	A structure containing:

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.
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.

CI_CONFIG_PTR	A pointer to a **CI_CONFIG** structure.
CI_DATA	A pointer to an array of characters.
CI_G	An array of 128 characters.
CI_HASHVALUE	An array of 20 characters.
CI_IV	An array of 24 characters.
CI_KEY	An array of 12 characters.
CI_KS	An array of 10 characters.
CI_P	An array of 128 characters.

Constant	Data Type
CI_PASSWORD	An array of **CI_PASSWORD_SIZE** + 4 (28) characters.
CI_PERSON	A structure containing:

Field Name	Data Type	Description
CertificateIndex	integer	Index of the certificate
CertLabel	**CI_CERT_STR**	Personality string of the certificate

CI_PIN	An array of **CI_PIN_SIZE** + 4 (16) characters.
CI_Q	An array of 20 characters.
CI_RA	An array of 128 characters.
CI_RB	An array of 128 characters.
CI_RANDOM	An array of 20 characters.
CI_RANDSEED	An array of 8 characters.
CI_REG_FLAGS	An array of 4 characters.
CI_SAVE_DATA	An array of 28 characters.
CI_SERIAL_NUMBER	An array of 8 Unsigned characters.
CI_SIGNATURE	An array of 40 characters.
CI_STATE	An unsigned integer value.
CI_STATE_PTR	A pointer to an unsigned integer.
CI_STATUS	A structure containing:

Field Name	Data Type	Description
CurrentSocket	int	The currently selected socket.
LockState	int	The lock status of the current socket.
SerialNumber	**CI_SERIAL_NUMBER**	The serial number of the Crypto Engine Chip.
CurrentState	**CI_STATE**	The state of the Card.
Decryption Mode	int	The mode of the Card.
Encryption Mode	int	The mode of the Card.
CurrentPersonality	int	The index of the current personality.
KeyRegisterCount	int	Count of Key Registers on the Card.
KeyRegisterFlags	**CI_REG_FLAGS**	A set of bit fields indicating register use.
CertificateCount	int	Count of certificates on the Card.
CertificateFlags	**CI_CERT_FLAGS**	A set of bit fields indicating certificate use.
Flags[4]	unsigned char	Flags[0] Bit 4. Clock speed: Fast =1, Slow = 0. Bit 6. Condition: Typical = 0, Worst Case = 1. All other bits and bytes are reserved.

CI_STATUS_PTR	A pointer to a void.
CI_TIME	An array of 16 characters.
CI_TIMESTAMP	An array of 16 characters.
CI_WRAPPED_X	An array of 24 characters.
CI_X	An array of 20 characters.
CI_Y	An array of 128 characters.
NULL	A pointer whose value is 0.