libSIAEcard: specifiche libreria e ambiente di collaudo

Author: Giuseppe Amato

Version: 1.1

Indice

1 Premessa

1.1 Acronimi

AC	Access Conditions
AID	Application Identifier
APDU	Application Protocol Data Unit
ASN	Abstract Syntax Notation
ATR	Answer-To-Reset
BER	Basic Encoding Rules
BS	Base Security
BSO	Base Security Object
CC	Common Criteria Version 2.1
CGA	Certification Generation Application
CSE	Current Security Environment
DES	Data Encryption Standard
DF	Directory File
DIR	Directory

DS	Digital Signature
EAL	Evaluation Assurance Level
EF	Elementary File
ETU	Elementary Time Unit
FCI	File Control Information
FID	File ID
HI	Human Interface
HW	Hardware
ICC	Integrated Circuit Card
ID Card	Identity Card
I/O	Input/Output
IT	Information Technology
lsb	Last Significant Bit (b0)
LSB	Last Significant Byte
MAC	Message Authentication Code
MF	Master File
msb	Most Significant Bit (b7)
MSB	Most Significant Byte
MSE	Manage Security Environment (command)
MTSC	Manufacturer Transport Secure Code
OS	Operating System
OCI	Object Control Information
PDA	Personal Digital Assistant
PDC	Patient Data Card
PIN	Personal Identification Number
PP	Protection Profile
PPSC	Personalization Secure Code
PSO	Perform Security Operation (command)
RFU	Reserved for Future Use

RSA	Rivest, Shamir, Adleman
SCA	Signature-Creation Application
SCD	Signature-Creation Data
SD	Signatory's data
SDO	Signed Data Object
SE	Security Environment
SECI	Security Environment Control Information
SEO	Security Environment Object
SF	Security Function
SFI	Short File Identifier
SHA	Secure Hash Algorithm
SFP	Security Function Policy
SM	Secure Messaging
SOF	Strength of Function
SSCD	Secure Signature-Creation Device
ST	Security Target
SVD	Signature-Verification Data
TLV	Tag Length Value
TOE	Target of Evaluation
TSC	TSF Scope of Control
TSF	TOE Security Functions
TSFI	TSF Interface
TSP	TOE Security Policy

1.2 Introduzione

1.2.1 Architettura PC/SC

"*libSIAEcard*" è stata implementata tenendo conto dello standard PC/SC (rif. http://www.pcscworkgro
In tale ottica, la libreria costituisce uno strato middleware che consente ad applicazioni con un livello di astrazione più alto, di comunicare, in maniera del tutto trasparente con

la smart card, ovvero un ICC Service Provider (come è possibile evincere dallo schema dell'architettura PC/SC riportato di seguito). La scelta dell'architettura standard PC/SC è giustificata dal fatto che, essa rappresenta, di fatto, uno standard di riferimento per lo sviluppo di applicazioni su smart card in tutti gli ambienti software al momento più diffusi.

1.3 Descrizione dei file della libreria e note sulla compilazione.

La libreria "libSIAE" fornita insieme al presente documento, è scritta interamente in linguaggio "ANSI C".

I nomi dei file che compongono la libreria e la descrizione del loro contenuto sono riportati di seguito:

libSIAECardT.h	Definizione delle costanti e dei tipi utilizzati.
libSIAECard.h	Prototipi delle funzioni di uso generale della libreria.
libSIAECard.c	Implementazione delle funzioni di uso generale della libreria.
scardHAL.h	Header delle funzioni platform dependent (HAL=Hardware Abstraction Layer)
scardHAL.c	Implementazione delle funzioni dell'HAL. L'implementazione riportata nel fiel è valida sia sotto piattaforma Linux sia sotto piattaforma Win32. La portatilità delle funzioni sotto sistemi di natura diversa è fortemente dipendente dalla presenza sul sistema di uno strato PC/SC.
internals.h	Contiene la definizione di costanti utilizzate internamente.
sha1.h md5.h global.h sha1.c md5.c	File contenenti header e implementazione delle funzioni per il calcolo dell'hashing.

1.3.1 Note per la compilazione

Di default, le funzioni vengono compilate con le calling convention tipiche del linguaggio C e, pertanto, possono essere richiamate solo da tale linguaggio. E' possibile, altresì settare la convenzione delle chiamate alla modalità standard (stdcall) inserendo la riga *#define USE_STDCALL* nel proprio codice sorgente. In questo modo, la libreria risul-

tante potrà essere utilizzata anche da linguaggi di programmazione di natura diversa dal C.

1.4 Descrizione delle API implementate

API per la gestione del collegamento con la smart card.

1.4.1 isCardIn

La funzione isCardIn rileva la presenza della smart card nel lettore individuato da nSlot.

Prototipo:

```
int isCardIn(int Slot)
```

Argomenti: Slot indica il numero dello slot nel quale è inserita la carta con la quale si vuole stabilire il collegamento PC/SC.

Descrizione codici di ritorno: La funzione restituisce un intero diverso da 0 (zero) se nel lettore nSlot è presente una smart card. In caso contrario, la risposta è 0 (zero).

1.4.2 Initialize

La funzione Initialize crea ed inizializza il collegamento con la smart card presente in un lettore.

Prototipo:

```
int Initialize(int Slot)
```

Argomenti: Slot indica il numero dello slot nel quale è inserita la carta con la quale si vuole stabilire il collegamento PC/SC.

N.B.: la funzione Initialize deve essere sempre chiamata prima di ogni altra funzione della libreria.

Descrizione codici di ritorno:

C_OK	Operazione conclusa con successo.

C_CONTEXT_ERROR	Questo errore si verifica quando la libreria non riesce a stabilire un contesto PC/SC dell'applicazione. Attenzione! E' un errore estremamente grave e non recuperabile dal programmatore. Il modulo PC/SC potrebbe non essere stato installato correttamente. Verificare la corretta installazione del software e/o contattare l'amministratore di sistema.
C_READER_UNKNOWN	Il valore Slot passato alla libreria non è riconosciuto come valido. Una possibile causa di ciò potrebbe essere una cattiva installazione dei driver del lettore di smart card. Contattare l'amministratore di sistema per una verifica della corretta installazione dei driver.
C_NO_CARD	Non è presente nessuna carta nel lettore.
C_UNKNOWN_CARD	La carta presente nel lettore non è una S.I.A.E. card o c'è qualche problema sul file system.
C_ALREADY_INITIALIZED	La libreria è già stata inizializzata.
C_GENERIC_ERROR	Si è verificato un errore durante l'operazione.

1.4.3 Finalize

Conclude la comunicazione con la carta.

Prototipo:

```
int Finalize()
```

Descrizione codici di ritorno:

C_OK	Operazione conclusa con successo.
C_NOT_INITIALIZED	Non è stato inizializzato alcun canale di comunicazione.

Esempio di utilizzo delle funzioni di gestione della comunicazione con le smart card:

```
...
/* Dichiarazioni ed inizializzazioni varie */
int rc=C_OK;
...
```

```
/*   Inizializzazione comunicazione con la carta
 presente nel primo lettore (0)    */
rc=Initialize(0) ;
...
// Operazioni sulla carta
...
rc=Finalize() ;

...
```

1.4.4 API per la gestione dei file.

La "S.I.A.E. Card" è inizializzata in modo tale da consentire la gestione di un numero massimo di 256 file compatibilmente con la quantità di memoria disponibile. I file possono essere di diverse tipologie:

- EF binary: cioè file che vengono visti, all'interfaccia, come sequenze di byte ai quali si può accedere in maniera causale.
- File a record: tali file possono essere visti come sequenze di record identificabili individualmente. A loro volta, i file a record possono essere di diverso tipo:
  - EF Linear Fixed: sono sequenze di record aventi tutti la stessa dimensione;
  - EF Linear Variable: le dimensioni dei singoli record sono variabili;
  - EF Cyclic: sono file con record di lunghezza fissa nei quali l'accesso avviene in maniera circolare, ovvero, dopo aver letto l'ultimo record, si ricomincia a leggere il primo.

1.4.5 Select

La funzione Select, selezione un file sulla carta, preparando la stessa, nel caso in cui il file sia di tipo elementare, ad una operazione di lettura.

Prototipo:

```
int Select(int fid)
```

Parametri:

fid	è l'indice del file da selezionare.

Descrizione codici di ritorno:

C_OK	Operazione conclusa con successo.
C_NOT_INITIALIZED:	Non è stato inizializzato alcun canale di comunicazione.
C_FILE_NOT_FOUND:	Il file che si sta cercando di leggere non è presente sulla smart card.

1.4.6 ReadBinary

La funzione ReadBinary consente la lettura del contenuto di un file di tipo binario.

Prototipo:

```
int ReadBinary(WORD Offset, BYTE* Buffer, int *Len)
```

Parametri:

Offset	è il byte del file a partire dal quale si vuole cominciare la lettura.
Buffer	è il puntatore alla zona di memoria destinata a contenere il risultato.
Len	è il numero di byte che si desidera leggere. Qualora tale valore sia superiore ai byte effettivamente presenti nel file, all'uscita della funzione,in Len sarà contenuto il numero di byte effettivamente letti.

Descrizione codici di ritorno:

C_OK	Operazione conclusa con successo.
C_NOT_INITIALIZED	Non è stato inizializzato alcun canale di comunicazione.

C_FILE_NOT_FOUND	Il file che si sta cercando di leggere non è presente sulla smart card.
C_NOT_AUTHORIZED	Non è stato possibile leggere dal file richiesto in quanto non si possiedono i diritti per questa operazione sul file.
C_WRONG_TYPE	Il file scelto è incompatibile con la funzione ReadBinary; è possibile, ad esempio che si stia cercando di effettuare una lettura di tipo binario su di un file con struttura a record.
C_WRONG_LENGTH	Non è stato possibile leggere tutti i dati richiesti. Il numero di byte effettivamente letti è contenuto il Len.

Esempio:

Il seguente esempio mostra come leggere dal file "5F0A", 20 byte a partire dal 30°.

```
...
BYTE Buff[20];
int rc=C_OK;
int Len=20;
rc=Select(0x5F0A);
rc=ReadBinary(30, Buff, &Len);

...
```

1.4.7 ReadRecord

La funzione ReadRecord estrae un record da un file avente una struttura a record.

Prototipo:

```
int ReadRecord( int nRec, BYTE* Buffer, int *Len)
```

Parametri:

nRec	è l'indice del record che si vuole estrarre.
Buffer	è il puntatore alla zona di memoria nella quale deve essere estratto il record.
Len	è la lunghezza del record che si vuole estrarre. Qualora la lunghezza del record da estrarre non fosse nota a priori, è possibile ricavarla effettuando la ReadRecord in due passi come indicato nell'esempio che segue.

Esempio:

```
...
int rc=C_OK;
int Len;
rc=Select(0x3F00);
rc=Select(0x0000);
rc=Select(0x1111);
rc=Select(0x5F02);
/*   Passando NULL come terzo parametro la funzione
ReadRecord ricava la lunghezza del record selezionato
e la restituisce in Len           */
rc=ReadRecord(1, NULL, &Len);
BYTE *Buff;
Buff=(BYTE)malloc(Len);
rc= ReadRecord (1, Buff, Len);

...
```

1.4.8 GetSN

La funzione GetSN legge il numero di serie della smart card.

Prototipo:

```
int GetSN(BYTE serial[8])
```

**Parametri: **

Inline strong start-string without end-string.

serial	è la stringa di byte destinata a contenere il numero di serie della smart card (8 byte).

Esempio: :: ... int rc=C_OK; BYTE sn[16]; rc=GetSN(sn); ...

Descrizione codici di ritorno:

C_OK	Operazione conclusa con successo.
C_NOT_INITIALIZED	Non è stato inizializzato alcun canale di comunicazione.
C_FILE_NOT_FOUND	Il file EF.GDO (contenente il serial number) non è presente sulla smart card.
C_GENERIC_ERROR	Non è stato possibile leggere il serial number.

1.4.9 VerifyPIN

La funzione VerifyPIN verifica in PIN consente di effettuare una verifica del PIN sulla smart card. La carta riceve in ingresso il valore del pin, lo confronta con il corrispettivo valore in essa memorizzato e, nel caso in cui tali valori coincidessero, abilita tutte le operazioni del caso. Il PIN deve essere verificato a partire dal S.I.A.E. Application Domain (DF 0000) o da uno dei DF al suo interno.

Attenzione! Dopo 3 volte che la verifica di un PIN fallisce, tale PIN risulta bloccato.

Prototipo:

```
int VerifyPIN(int nPIN, char *pin)
```

Parametri:

nPIN	è il numero del PIN da verificare.
pin	è una stringa NULL terminated contenente il valore da verificare.

Descrizione codici di ritorno:

C_OK	Operazione conclusa con successo.
C_NOT_INITIALIZED	Non è stato inizializzato alcun canale di comunicazione.
C_PIN_BLOCKED	Il PIN che si sta cercando di verificare è bloccato.
0x63Cx	La verifica del PIN non è andata a buon fine, restano ancora x tentativi.

1.4.10 ChangePIN

La funzione ChangePIN consente di cambiare il valore di un PIN.

Prototipo:

```
int ChangePIN(int nPIN, char *Oldpin, char *Newpin)
```

Parametri:

nPIN	è il numero del PIN da verificare.
Oldpin	è una stringa NULL terminated contenente il vecchio valore del PIN.
Newpin	è una stringa NULL terminated contenente il nuovo valore del PIN.

Descrizione codici di ritorno:

C_OK	Operazione conclusa con successo.
C_NOT_INITIALIZED	Non è stato inizializzato alcun canale di comunicazione.
C_PIN_BLOCKED	Il PIN che si sta cercando di cambiare è bloccato.
0x63Cx	La verifica del vecchio PIN non è andata a buon fine, restano ancora x tentativi.

1.4.11 UnblockPIN

La funzione UnblockPIN consente di sbloccare il valore di un PIN previa verifica del PUK ad esso associato.

Prototipo:

```
int UnblockPIN(int nPIN, char *Puk, char *Newpin)
```

Parametri:

nPIN	è il numero del PIN da sbloccare.
Puk	è una stringa NULL terminated contenente il PUK da verificare.
Newpin	è una stringa NULL terminated contenente il valore da assegnare al PIN una volta sbloccato.

Descrizione codici di ritorno:

C_OK	Operazione conclusa con successo.
C_NOT_INITIALIZED	Non è stato inizializzato alcun canale di comunicazione.
0x63Cx	La verifica del PUK non è andata a buon fine, restano ancora x tentativi.

1.5 Funzioni per la gestione del contatore di emissioni

1.5.1 ReadCounter

Legge il valore corrente del contatore di emissioni (EF_CNT).

Prototipo:

```
int ReadCounter(DWORD *value)
```

Parametri:

value	è il puntatore alla zona di memoria destinata a contenere il valore attuale del contatore.

Descrizione codici di ritorno:

C_OK	Operazione conclusa con successo.
C_NOT_INITIALIZED	Non è stato inizializzato alcun canale di comunicazione.
C_NOT_AUTHORIZED	Condizioni di accesso non soddisfatte per questa operazione.

1.5.2 ReadBalance

Legge il valore corrente del contatore di bilancio (EF_CNT_BALANCE).

Prototipo:

```
int ReadBalance(DWORD *value)
```

Parametri:

value	è il puntatore alla zona di memoria destinata a contenere il valore attuale del “Balance Counter”.

Descrizione codici di ritorno:

C_OK	Operazione conclusa con successo.
C_NOT_INITIALIZED	Non è stato inizializzato alcun canale di comunicazione.
C_NOT_AUTHORIZED	Condizioni di accesso non soddisfatte per questa operazione.

1.5.3 ComputeSigillo

Incrementa il valore del contatore di emissioni di una unità e ne restituisce il MAC calcolato in base al nuovo valore del contatore, della data e dell’ora dell’incremento, del prezzo del biglietto emesso e del numero di serie della smart card. Prima di effettuare il calcolo del Sigillo Fiscale, è necessario aver verificato il PIN (vd. VerifyPIN).

Prototipo:

```
int ComputeSigillo(BYTE *Data_Ora,DWORD Prezzo,BYTE *SN,
            BYTE *mac,DWORD *cnt)
```

Parametri:

Data_Ora	è una sequenza di 8 byte che contiene l’ora del conteggio.
Prezzo	è il prezzo del biglietto emesso.
SN	è una sequenza di byte contenente il numero di serie della smart card.
mac	è il puntatore alla zona di memoria nella quale la funzione restituisce il MAC calcolato sull’attuale valore di conteggio. Il MAC è costituito da 8 byte.
cnt	è la variabile di output che contiene il valore di conteggio dopo l’incremento.

Descrizione codici di ritorno:

C_OK	Operazione conclusa con successo.
C_NOT_INITIALIZED	Non è stato inizializzato alcun canale di comunicazione.
C_NOT_AUTHORIZED	Condizioni di accesso non soddisfatte per questa operazione.

1.5.4 ComputeSigilloEx

E' funzionalmente identica alla funzione ComputeSigillo con la differenza che legge automaticamente il serial number della carta. Prima di effettuare il calcolo del Sigillo Fiscale, è necessario aver verificato il PIN (vd. VerifyPIN).

Prototipo:

```
int ComputeSigilloEx(BYTE *Data_Ora,DWORD Prezzo,
   BYTE *mac,DWORD *cnt)
```

Parametri:

Data_Ora	è una sequenza di 8 byte che contiene l'ora del conteggio.
Prezzo	è il prezzo del biglietto emesso.
mac	è il puntatore alla zona di memoria nella quale la funzione restituisce il MAC calcolato sull'attuale valore di conteggio. Il MAC è costituito da 8 byte.
cnt	è la variabile di output che contiene il valore di conteggio dopo l'incremento.

Descrizione codici di ritorno:

C_OK	Operazione conclusa con successo.
C_NOT_INITIALIZED	Non è stato inizializzato alcun canale di comunicazione.
C_NOT_AUTHORIZED	Condizioni di accesso non soddisfatte per questa operazione.

1.5.5 ComputeSigilloFast

La funzione ComputeSigilloFast è fondamentalmente analoga alla funzione ComputeSigillo ma, al fine di rendere più snello e veloce il calcolo iterato di sigilli fiscali, non effettua le operazioni di selezione file preventive. La funzione ComputeSigilloFast, pertanto presuppone che l'utente si sia posizionato preventivamente sul file contatore (EF_CNT) ed ottenuto i permessi necessari per il calcolo del sigillo (attraverso il comando VerifyPIN).

Prototipo:

```
int ComputeSigilloFast(BYTE *Data_Ora,DWORD Prezzo,BYTE *SN,
            BYTE *mac,DWORD *cnt)
```

Parametri:

Data_Ora	è una sequenza di 8 byte che contiene l'ora del conteggio.
Prezzo	è il prezzo del biglietto emesso.
SN	è una sequenza di byte contenente il numero di serie della smart card.
mac	è il puntatore alla zona di memoria nella quale la funzione restituisce il MAC calcolato sull'attuale valore di conteggio. Il MAC è costituito da 8 byte.
cnt	è la variabile di output che contiene il valore di conteggio dopo l'incremento.

Descrizione codici di ritorno:

C_OK	Operazione conclusa con successo.
C_NOT_INITIALIZED	Non è stato inizializzato alcun canale di comunicazione.
C_NOT_AUTHORIZED	Condizioni di accesso non soddisfatte per questa operazione.

1.6 Funzioni Crittografiche

1.6.1 GetKeyID

La funzione GetKeyID restiruisce il key identifier della chiave disponibile per la firma.

Prototipo:

```
BYTE GetKeyID()
```

Il valore restituito dalla funzione rappresenta l'identificativo univoco della chiave di firma attiva sulla carta. In caso di errore, la funzione ritorna 0.

1.6.2 GetCertificate

La funzione GetCertificate, legge il certificato dalla smart card e lo restituisce in un buffer.

Prototipo:

```
int GetCertificate(BYTE *cert, int *dim)
```

Parametri:

cert	è il puntatore al buffer destinato a contenere il certificato letto;
dim	è il puntatore all'intero che rappresenta la dimensione (in byte) del certificato.

Suggerimento: Per una corretta allocazione della memoria necessaria a contenere il certificato, si consiglia di chiamare due volte la funzione GetCertificate come mostrato nell'esempio che segue:

```
...
int dim=0;
int rv=GetCertificate(NULL,&dim);
/*
La funzione restituisce in dim la dimensione
del certificato.
*/
...
BYTE *cert=(BYTE)malloc(dim);
rv=GetCertificate(cert,&dim);

...
```

1.6.3 Sign

La funzione Sign effettua la firma di un blocco di byte (la cui dimensione deve essere di 128 byte).

Prototipo:

```
int Sign(BYTE kx, BYTE *toSign,  BYTE *Signed)
```

Parametri:

kx	è l'indice della chiave privata da utilizzare per la firma. E' possibile ottenere tale indice chiamando la finzione GetKeyID.
toSign	è il puntatore al buffer da firmare la cui dimensione deve essere di 128 byte. Signed è il puntatore ad un buffer dalle dimensioni di 128 byte (precedentemente allocato) destinato a contenere il risultato della firma.

Descrizione codici di ritorno:

C_OK	Operazione conclusa con successo.
C_NOT_INITIALIZED	Non è stato inizializzato alcun canale di comunicazione.
C_NOT_AUTHORIZED	Condizioni di accesso non soddisfatte per questa operazione.

1.6.4 GetCACertificate

La funzione GetCACertificate, legge il certificato della Certification Authority che ha emesso il certificato utente dalla smart card e lo restituisce in un buffer.

Prototipo:

```
int GetCACertificate(BYTE *cert, int *dim)
```

Parametri:

cert	è il puntatore al buffer destinato a contenere il certificato letto;
dim	è il puntatore all'intero che rappresenta la dimensione (in byte) del certificato.

1.6.5 GetSIAECertificate

La funzione GetSIAECertificate, legge il certificato SIAE dalla smart card e lo restituisce in un buffer.

Prototipo:

```
int GetSIAECertificate(BYTE *cert, int *dim)
```

Parametri:

cert	è il puntatore al buffer destinato a contenere il certificato letto;
dim	è il puntatore all'intero che rappresenta la dimensione (in byte) del certificato.

1.6.6 Hash

Prototipo:

```
int Hash(int mec,BYTE *toHash, int Len, BYTE *Signed)
```

Parametri:

mec	è il meccanismo che si vuole utilizzare per effettuare l'hash (vd. Appendice A).
toHash	è il puntatore al buffer del quale si desidera calcolare l'hash.
Len	è la dimensione del buffer del quale si desidera calcolare l'hash.

Hashed	è il puntatore ad un buffer destinato a contenere l'hash del buffer toHash (le dimensioni di tale buffer sono di 20 byte nel caso di hash SHA1 e 16 nel caso di MD5).

Descrizione codici di ritorno:

C_OK	Operazione conclusa con successo.
C_GENERIC_ERROR	Si è verificato un errore durante l'operazione.

1.6.7 Padding

Prototipo:

```
int Padding(BYTE *toPad, int Len, BYTE *Padded)
```

Parametri:

toPad	è il puntatore al buffer del quale si desidera ottenere il Padding.
Len	è la dimensione del buffer del quale si desidera ottenere il Padding.
Padded	è il puntatore ad un buffer destinato a contenere il padding del buffer toPad (le dimensioni di tale buffer sono di 128 byte).

Descrizione codici di ritorno:

C_OK	Operazione conclusa con successo.
C_GENERIC_ERROR	Si è verificato un errore durante l'operazione.

Esempio di utilizzo delle funzioni crittografiche:

L'esempio che segue, mostra in che modo è possibile utilizzare le funzioni crittografiche messe a disposizione per firmare l'hash SHA1 di un buffer ("12345678"):

```
...
BYTE toSign[8]=‘‘12345678’’;
BYTE Padded[128];   /* Buffer che conterrà il contenuto da firmare */
BYTE Hashed[20];    /* Buffer che conterrà l’Hash  da firmare */
BYTE Signed[128];
int rc=C_OK;
int kid=0;
...
kid=GetKeyID();
rc=Hash(HASH_SHA1,toSign,8,Hashed);
rc=Padding(Hashed,20,Padded);
rc=Sign(kid ,Padded,128,Signed);

...
```

1.6.8 BeginTransaction

La funzione BeginTransaction inizia una transazione PCSC, acquisendo il lettore per l’uso esclusivo. NOTA: la transazione deve durare il tempo strettamente necessario ad effettuare le operazioni richieste.

Prototipo:

```
int BeginTransaction()
```

1.6.9 EndTransaction

La funzione EndTransaction termina una transazione PCSC iniziata da BeginTransaction.

Prototipo:

```
int EndTransaction()
```

1.7 Estensione del set di funzioni

Di tutte le funzioni che effettuano operazioni sulla smart card, sono presenti delle versioni estese. In particolare, tali “estensioni”, prendono come ulteriore parametro di ingresso il lettore sul quale devono essere indirizzate.

Nella seguente tabella sono riportate, sulla prima colonna le funzioni originarie della libreria e sulla seconda, l’estensione di ciascuna di esse.

Finalize	FinalizeML
Select	SelectML

ReadBinary	ReadBinaryML
ReadRecord	ReadRecordML
GetSN	GetSNML
VerifyPIN	VerifyPINML
ChangePIN	ChangePINML
UnblockPIN	UnblockPINML
ReadCounter	ReadCounterML
ReadBalance	ReadBalanceML
ComputeSigillo	ComputeSigilloML
ComputeSigilloEx	ComputeSigilloExML
ComputeSigilloFast	ComputeSigilloFastML
Sign	SignML
GetKeyID	GetKeyIDML
GetCertificate	GetCertificateML
BeginTransaction	BeginTransactionML
EndTransaction	EndTransactionML

Il prototipo di ciascuna delle funzioni estese è identico a quello della relativa funzione originaria ma presenta, come parametro aggiuntivo, in coda, il numero dello slot sul quale si intende lavorare. Di seguito sono riportati, a titolo esplicativo, i prototipi della funzione Select originaria, e quello della relativa funzione estesa (SelectML).

```
int Select(WORD fid); à  int SelectML(WORD fid, int nSlot);
```

1.8 Appendice A: Costanti definite nella libreria

ACCESS CONDITIONS	
AC_NEVER	00
AC_PIN1	01
AC_PIN2	02
…	…

AC_ALWAYS	15
FILE TYPE	
EF_BINARY	01
EF_LINEAR_FIXED	02
EF_LINEAR_FIXED_TLV	03
EF_LINEAR_VARIABLE	04
EF_LINEAR_VARIABLE_TLV	05
EF_CYCLIC	06
EF_CYCLIC_TLV	07
HASHING MECHANISMS	
HASH_SHA1	01
HASH_MD5	02

1.9 Appendice B: Tipi definiti nella libreria

La libreria definisce ed utilizza i seguenti tipi:

DWORD: unsigned long;

WORD: unsigned short;

BYTE: unsigned char;

AccessConditions:

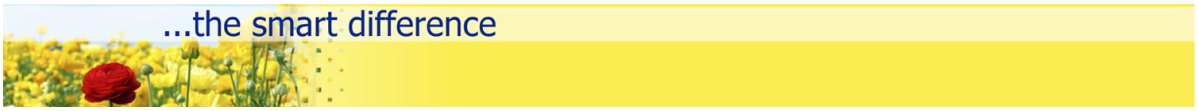
```
typedef struct {
 int  ACRead;
int  ACUpdate;
int ACAppend;
  int ACDelete;
  int ACAdmin;
  int ACIncrease;
  int ACDecrease;

} AccessConditions;
```

Tali definizioni si trovano all'interno del file "libSIAEcard.h"

1.10 Appendice C: Definizione e descrizione degli errori

Tutte le funzioni implementate ritornano il valore C_OK se concluse con successo. In caso di errore, fare riferimento alla seguente tabella.


...the smart difference

Risposta	Valore
C_OK	0000
C_CONTEXT_ERROR	0001
C_NOT_INITIALIZED	0002
C_ALREADY_INITIALIZED	0003
C_NO_CARD	0004
C_UNKNOWN_CARD	0005
C_COUNTER_ALREADY_INITIALIZED	0010
C_WRONG_LENGTH	6282
	63Cx
C_WRONG_TYPE	6981
C_NOT_AUTHORIZED	6982
C_PIN_BLOCKED	6983
C_WRONG_DATA	6A80
C_FILE_NOT_FOUND	6A82
C_RECORD_NOT_FOUND	6A83
C_WRONG_LEN	6A85
C_ALREADY_EXISTS	6A89
C_GENERIC_ERROR	FFFF