

Writing a Smart Card Application

Siddhant Bhansali
Kritikal Solutions Private Limited

1. What is a Smart Card?

A smart card is a credit card sized piece of plastic that contains a tiny microprocessor and some memory. Having these two components allows the Smart Card to store and process information, unlike credit cards and magnetic strip cards which can only store data.

There are two ways by which a Smart Card can transfer data to and from the world. Contact cards interact via gold plated points and Contactless cards interact via radio frequencies embedded in the card itself.

In either case, a special reader is required to read (or write) to a card. For contact cards, the card has to be actually inserted into the reader for it to exchange data properly. For contactless cards, merely bringing them close to the reader is sufficient. In case of special reader hardware which is placed outside doorways in order to grant access to the building, the results of the transaction is displayed via LED lights or a simple LCD display.

Smart Cards, like magnetic strip cards do not require an internal battery. The energy is provided via the reader when it establishes contact. In contactless cards, the radio waves serve as a power source.

For all practical purposes, a smartcard is like a mini-harddisk. It contains an Electrical Erasable Programmable Read Only Memory. There are some cards which contain only memory, known as "memory cards". But Smart Cards are defined as those with microprocessors and memory in them.

The basic idea behind a smart card is to provide security (of certain information) while enabling mobility of it as well. Imagine carrying your Driver's License, Bus Pass, Bank ATM Card, Company ID Card, Phone Card, and special club membership cards all rolled into one "credit card" sized device. That is the power of a Smart Card!

A smart card is considered to be quite secure, as it is very difficult to forge the card or to access protected data on it. Such security features enable it to serve as an ideal medium for applications such as an "electronic purse".

2. Why and where can we use a Smart Card?

Among some of the applications already in use worldwide:

- Access control to buildings or restricted rooms
- Automatic fare collection in buses, trains and airline travel
- Electronic purse
- Financial transactions

- Loyalty and customer programs
- Phone SIM cards
- Warehouse and inventory control

Some of the advantages of using a smart card are:

- It is more reliable than a magnetic stripe card
- It can store more information than a magnetic stripe card
- It is more difficult to tamper with
- It can be made disposable or reusable
- It can perform multiple functions in a wide range of applications
- It is compatible with portable electronic devices such as phones, personal digital assistants (PDAs), and PCs

3. Types of Smart Cards

There are a variety of Smart Cards for different applications and purposes. Some of these are listed below:

1. Memory or Microprocessor cards
2. Contact or Contactless cards
3. Hybrid cards – which possess both a microprocessor and memory but those two are not inter-connected
4. Dual Interface cards – which have a contact and a contactless surface. For e.g., to loading an e-purse is done via the contact side only, but spending of small amounts can be done contactlessly.
5. Optical cards – can be used to store large amounts of data, like X-ray images of a patient.

4. Different types of Smart Card Interfaces

PC/SC

This standard was developed by Microsoft and several other companies. It is an application interface for communicating with smart cards from Windows-based platforms. It is possible to work with SmartCards in a Linux environment using the MUSCLE API.

<http://www.pcscworkgroup.com/>

OpenCard Framework

The OpenCard Framework provides a common interface for both the smart card reader and the application on the card based on a Java® architecture, allowing portability and interoperability. It also enables interaction with existing Personal Computer/Smart Card (PC/SC) 1.0 supported reader devices. The OpenCard Framework is supported by an Industry consortium that provides for inter-operable smart cards solutions across many hardware and software platforms. The OpenCard Framework is an open standard providing an architecture and a set of APIs that enable application developers and service providers to build and deploy smart card aware solutions in any OpenCard-compliant environment.

<http://www.opencard.org/>

JavaCard

The Java Card specifications enable Java technology to run on smart cards. Applets developed with Java Card technology will run on any Java Card technology-based smart card, independently of the card vendor and underlying hardware. There are many unique features to a JavaCard, such as: secure execution environment, dynamic installation of new applications, and compatibility with international standards for smart cards such as ISO7816.

<http://java.sun.com/products/javacard/>

5. What is needed to build a Smart Card Application?

Not much! All you need is a smart card reader, drivers to install it on your system, test software from the manufacturer and a few smart cards to play with. Having a development environment to program in wouldn't hurt either! We used Microsoft Visual C++ for this article, since it had several built-in functions which we could call directly. You can also use Linux along with the MUSCLE API. We've also provided a class which provides an easier interface to the Smart Card functions present in Visual C++ and MUSCLE API.

Smart card reader

The reader is an external hardware device that plugs into your PC's serial or USB port. There are separate reader units such as those which are meant to be located on walls next to doorways. But we have used a SCM SCR301 Smart Card Reader which can attach to the USB port of a PC. For development purposes, you would want to start off in a similar manner, and later on creating the application in custom hardware.

Once proper communication is established between the computer and the Reader, interfacing with the smart card should be a breeze. Although each Reader manufacturer builds a unique interface for his device, communicating with the card itself is based on a well established standard – the ISO 7816 specifications.

The drivers for your Smart Card Reader and the diagnostic programs should be included with the device or may be available from the manufacturer's website.

Smart Cards

Some of the more well known smart card manufacturers are: Gemplus and Schlumberger. Visit their websites or contact their Indian agents to place an order. Their addresses are listed at the end of this article.

Microsoft Visual C++ 6

VC++ has some built-in functions which make it easier to develop Smart Card Applications. However, you can also use a Linux environment with the MUSCLE API. Some modifications to the PCSC_WRAPPER class may be required.

The PCSC_WRAPPER Class

We've built a class in C++ which interfaces with the standard libraries in Visual C++ and the MUSCLE API, taking out much of the complexities and hassles involved. Although we do not cover all possible scenarios where the smart card can be used – it should serve well as a template for basic applications, while allowing further customization by the reader.

6. The Smart Card

Before we go any further, we have to understand a few unique aspects of Smart Cards which enable them to be such useful devices. These concepts are:

- The Directory Structure
- SmartCard Authentication
 - Internal Authentication
 - External Authentication
- Files on a SmartCard
 - File Types
 - Structure of files on a SmartCard
 - Identification of files on a SmartCard
- PV Key file
- Calculation Key file
- File Permissions
 - Possible Options for an "Action"
 - "Actions"
 - Binary/Transparent Files
 - Linear Record Files
 - Cyclic Record Files
 - Protection/Verification Key File
 - Structure of a PV Key
 - Calculation Key File
 - Structure of a Calculation Key
 - Dedicated File
- File Creation

The Directory Structure

The data stored in the memory of a smart card is laid out in a similar manner as the file structure on a PC – with directories, and files within it. At the top of this tree is a "Master File" (MF) – equivalent to the "root" directory. This file is mandatory and is present on all smart cards. Contained within the Master File, we have Dedicated Files (DF) (which can be considered as "application directories") and Elementary Files (EF) (which are used to hold information such as data and also secret keys and passwords).

Dedicated Files serve to isolate each application's data from one another. They contain other DF files or a set of EF files. Dedicated Files also serve as a security mechanism by (optionally) requiring key based verification before access is granted to the files contained below them.

It is possible to have up to 3 layers of DFs, but most applications, including the one we build here will not require that feature. We shall build our application in a DF right below the MF.

SmartCard Authentication

The concept of 'authentication' is very important in understanding how a Smart Card works. There are two kinds of authentication – Internal and External Authentication.

Internal Authentication

Refers to the mechanism by which the Application verifies the authenticity of the SmartCard, ie it checks whether the card inserted in it is a valid card or has it been tampered with. The Application sends a 'random number' to the SmartCard. The SmartCard then uses its 'Calculation Key' to encrypt that number and sends it back to the Application. The Application then encrypts the number with it's own key and compares it to what was received from the SC. If the two match, then the SmartCard shares the same 'Calculation Key' and therefore is authentic.

External Authentication

The 'world outside the SmartCard' is initially un-trusted and therefore must undergo a process to 'authenticate' itself to the SmartCard. This process is initiated when the Application sends a "get_challenge()" request to the SmartCard. Basically, it is asking the SmartCard to send it a 'random number'. The Application gets the 'number' and then applies the secret (DES) key on it, obtaining another number from it, equal in length. This number is then sent back to the SmartCard. After receiving this new number, the SmartCard applies the DES key stored within it to the number which it had initially sent out. If the two results match, then it knows that the Application has the valid DES key, and therefore can be 'trusted'.

Files on a SmartCard

File Types

Certain EF files can be categorized as "internal" files – those which are required by the Smart Card itself, in order to provide security or protection to your application's data and the rest which comprise of your data.

There are several kinds of "internal EF" files:

- Protection/Verification Key File
- Calculation Key File
- PIN File

*There may be additional "system files" used by card manufacturers specific to their implementation.

Structure of files on a SmartCard

Data itself can be organized according to the following file structures:

1. Transparent – this is equivalent to a "binary file" or to "no structure at all". Data is just stored as a sequence of bytes.

2. Record

- i. Linear Fixed – a fixed number of records exist in the file, decided on creation.
- ii. Linear Variable – a fixed number of records, but with variable lengths.
- iii. Cyclic Fixed – a fixed number of records, but the system cycles back to the first record after crossing the last one.

All files, including the “internal EF” files have one of the above layouts.

Identification of files on a SmartCard

All files are identified by an unique 2 byte hexadecimal code. The Master File has a predetermined file ID, “3F00”, defined as per ISO standards. This implies that the application/“reader device” must know in advance what file IDs are being used. Since you are most likely designing both – this shouldn’t be a problem.

If you are planning on making a heavy-duty smart card application with “dumb terminals” installed at different places, you need to plan and finalize the “structure” of the Smart Card before you start manufacturing the terminal hardware.

PV Key file

The Protection/Verification Key file contains a set of record(s) which are the keys required for “External Authentication”. Other data files (your Application’s data files) make a reference to one of the records within this file. There can be a maximum of 16 records in this file (0 .. 15).

Calculation Key file

The Calculation Key file contains a set of record(s) which are the keys required for “Internal Authentication”. Otherwise, it is similar to the PV Key file.

File Permissions

Permissions on certain types of files can vary from manufacturer to manufacturer. We have used Schlumberger’s Payflex type of cards, so the following may be specific to those only. This information can be obtained from looking at the card manufacturer’s datasheets.

It is important to note that the Smart Card OS keeps track of the “transaction” occurring with the outside world. So, if you authenticate yourself and are granted permission to access a certain file, you are granted that permission for the rest of the transaction. Having said that, it doesn’t hurt to authenticate yourself whenever you need to obtain specific permission for a file.

Access Conditions or, Possible Options for an "Action"

| | |
|------------------|--|
| Always | Always permit the action. |
| Never | Never permit the action. (i.e., never allow the outside world to perform this action. The Smart Card OS 'trusts' itself so it can perform its internal operations regardless.) |
| Protection | Require external authentication with specified PV Key before permitting this action.* |
| PIN | Require external authentication with specified PIN before permitting this action.* |
| PIN & Protection | Require external authentication with specified PIN and PV Key before permitting this action.* |

*Note that the PV Key/PIN itself is not specified here. Only an indication is made of what type of permission is required. A reference to the actual PV Key/PIN's location is made in another field.

"Actions"

| | |
|-------------|---|
| Read | This field specifies the conditions which have to be satisfied before the external world can read the file. |
| Write | Conditions to be satisfied before the external world can write to the file. |
| Update | Conditions to be satisfied before the external world can update the file. |
| Unblock | If a number of incorrect attempts are made, the card blocks the file in question which can only be "unblocked" by a special code. |
| Create File | This field determines the conditions which have to be satisfied before the DF can allow creation of new files within it. |

Note that not all of the above "actions" are permitted on all types of files. Certain combinations do not make sense and are not permitted. Also, there may be additional "actions" for certain types of files.

At file creation time, access conditions and the associated key numbers have to be specified.

Two bytes **AC1** and **AC2** are used to indicate which **access condition** is related to a "Action". Each file type can have a maximum of four types of "Actions" associated with it.

Two bytes **KN1** and **KN2** are used to indicate which **key** (0 to 15) is related to each "Action". As you can see, not all "Actions" require a key number, so you should send a 0x00 as a placeholder for those "Actions". These "Key Number" bytes have a meaning only if the corresponding "Action" is – Protection, PIN or "PIN & Protection".

Binary/Transparent Files

A transparent file has no internal structure and is used to store information such as binary data. "Actions" possible on this type of file

are READ_BINARY and UPDATE_BINARY. The data is accessed using an offset relative to the start of the file.

| [Byte] | [Bits: 0-3] | [Bits: 4-7] |
|--------|-------------|--------------------|
| AC1 | READ_BINARY | UPDATE_BINARY |
| AC2 | Reserved | LOCK_UPDATE_BINARY |

| [Byte] | [Bits: 0-3] | [Bits: 4-7] |
|--------|----------------------------|-----------------------------------|
| KN1 | Key Number for READ_BINARY | Key Number for UPDATE_BINARY |
| KN2 | Reserved | Key Number for LOCK_UPDATE_BINARY |

Linear Record Files

These files contain a set of records of fixed length. The first record of the structure for ordinary Linear Record files is accessed as record #1, but for PV Key or Calculation Key files, the first record is accessed as record #0. A dynamic pointer is automatically managed by the Payflex operating system to locate a record in reading and writing operations. You can use WRITE_RECORD to store several pieces of information on consecutive records, as it moves to the next record after writing to the current one. After it has written to the last one, the next call to it will fail. Note that WRITE_RECORD works only on "empty" locations; you must use UPDATE_RECORD to change the content of a record.

| [Byte] | [Bits: 0-3] | [Bits: 4-7] |
|--------|--------------|---------------|
| AC1 | READ_RECORD | UPDATE_RECORD |
| AC2 | WRITE_RECORD | Reserved |

| [Byte] | [Bits: 0-3] | [Bits: 4-7] |
|--------|-----------------------------|------------------------------|
| KN1 | Key Number for READ_RECORD | Key Number for UPDATE_RECORD |
| KN2 | Key Number for WRITE_RECORD | Reserved |

Cyclic Record Files

This type of file is similar to a linear record file, but with the records linked together as a ring. The last written record is record number 1. The oldest written record has the highest record number. The physical location of the newest/oldest record is managed by the operating system. This is useful when you have a fixed number of records you want to store, for example – the last 5 bank transactions.

| [Byte] | [Bits: 0-3] | [Bits: 4-7] |
|--------|-------------|---------------|
| AC1 | READ_RECORD | UPDATE_RECORD |
| AC2 | Reserved | Reserved |

| [Byte] | [Bits: 0-3] | [Bits: 4-7] |
|--------|-------------------------------|---------------------------------|
| KN1 | Key Number for READ_RECORD | Key Number for UPDATE_RECORD |
| KN2 | Reserved | Reserved |

Protection/Verification Key File

This is similar to a Linear Record File, except that the data contained within it is used as keys for “external authentication”. The first record is accessed as record #0, allowing for a maximum of 16 keys. The READ_RECORD access condition should always be set to NEVER, so that the ‘outside world’ can never read your secret keys. As you can see there is no possible way to “update” a key in this type of card, so once you create your SmartCards, and your keys are compromised – you have no choice but to reissue new cards. So it is pretty important that the security of the keys be maintained at all times.

If you are creating all the files which you want on the SmartCard at the same time, you can set WRITE_RECORD to ALWAYS. The user inserts a blank smart card into the reader device, then your application gains access to the card and writes everything all at once. Since this process is unlikely to be interrupted and hardly takes a second or two, having an ALWAYS permission on WRITE_RECORD is not a security issue. However if you have a situation where you need to write a PV Key to the file right now, and plan to put another later on, you need to establish security for the WRITE_RECORD action.

When creating a PV Key file, you have to specify a “Status byte” and a “Maximum counter” at the beginning and end of your key. See accompanying box titled “Structure of a PV Key” for details.

| [Byte] | [Bits: 0-3] | [Bits: 4-7] |
|--------|--------------|-------------|
| AC1 | READ_RECORD | Reserved |
| AC2 | WRITE_RECORD | UNBLOCK* |

| [Byte] | [Bits: 0-3] | [Bits: 4-7] |
|--------|--------------------------------|----------------------------|
| KN1 | Key Number for READ_RECORD | Reserved |
| KN2 | Key Number for WRITE_RECORD | Key Number for UNBLOCK* |

*Note: More information on the UNBLOCK feature can be obtained from the datasheet.

Calculation Key File

Similar to PV Key file except the data within it is used as keys for “internal authentication”. The first record is also accessed as record #0, allowing for a maximum of 16 keys. See accompanying box titled “Structure of a Calculation Key” for details.

| [Byte] | [Bits: 0-3] | [Bits: 4-7] |
|--------|--------------|-------------|
| AC1 | READ_RECORD | Reserved |
| AC2 | WRITE_RECORD | Reserved |

| [Byte] | [Bits: 0-3] | [Bits: 4-7] |
|--------|--------------------------------|-------------|
| KN1 | Key Number for READ_RECORD | Reserved |
| KN2 | Key Number for WRITE_RECORD | Reserved |

Structure of a PV Key

| Byte: | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |
|----------|----|-----|-----|-----|-----|-----|-----|-----|-----|----|
| Example: | 0F | ... | ... | ... | ... | ... | ... | ... | ... | 0F |

Byte 0: "Status byte"

| b7 | b6 | b5 ... b4 | b3 ... b0 |
|----------------------|--------------------------|----------------|----------------------------|
| Activation Indicator | Algorithm Mode Indicator | Version Number | Remaining Attempts Counter |

Activation Indicator:

"0" if activated, "1" if not.

Algorithm Mode Indicator:

"0" for DES. "1" for 3DES. Several consecutive keys in the PV Key file have to be declared in Mode "1" for 3DES to work. Most situations will not require this level of security.

Version Number:

Key version number.

Remaining Attempts Counter:

The number of consecutive attempts still possible.

Byte 1-8:

The Actual Key Value

Byte 9: "Maximum counter"

The maximum number of consecutive incorrect key attempts before the system blocks the key.

Note: If you happen to block your key, you are allowed ONE try of the Unblock command. If that fails, the Activation indicator is switched to 1 and the key is definitively blocked.

Structure of a Calculation Key

| | | | | | | | | | | |
|--------------|----------|----------|----------|----------|----------|----------|----------|----------|----------|----------|
| Byte: | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |
| Example: | 00 | ... | ... | ... | ... | ... | ... | ... | ... | 01 |

Byte 0: "Status byte"

| | | |
|----------------------|--------------------------|----------------|
| b7 | b6 | b5 ... b0 |
| Activation Indicator | Algorithm Mode Indicator | Version Number |

Activation Indicator:

"0" if activated, "1" if not.

Algorithm Mode Indicator:

"0" for DES. "1" for 3DES. Several consecutive keys in the PV Key file have to be declared in Mode "1" for 3DES to work. Most situations will not require this level of security.

Version Number:

Key version number.

Byte 1-8:

The Actual Key Value

Byte 9: "Key Type"

| Command type | Key type value |
|---------------------------------------|----------------|
| (for all calculation functionalities) | 00h |
| Internal_Authentication | 01h |
| Certificate Debit* | 02h |
| Certificate Credit* | 03h |

*See the PayFlex datasheet for additional details.

Dedicated File

There is only one possible "action" possible for a DF – CREATE_FILE. If you are creating all the files at the same time, you can set CREATE_FILE to ALWAYS. But if you are not, you should have some security precautions in place.

| [Byte] | [Bits: 0-3] | [Bits: 4-7] |
|--------|-------------|-------------|
| AC1 | Reserved | CREATE_FILE |
| AC2 | Reserved | Reserved |

| [Byte] | [Bits: 0-3] | [Bits: 4-7] |
|--------|-------------|-------------------------------|
| KN1 | Reserved | Key Number for CREATE_FILE |
| KN2 | Reserved | Reserved |

File Creation

All file sizes and permissions are set when the application 'framework' is written onto the card. It is not possible either to change permissions on a file, or to add/reduce file sizes at a later time. Trying to "overwrite" a file will also not work. Your application can under utilize the data space it was allocated, but cannot use more. Although there are parallels between the file structure on a Smart Card and an ordinary PC, don't assume everything will work as in a PC. Think of a DF as more like a partition, which once put in place, cannot change its size.

You have to begin writing your application's file structure on the card with the DF. Once you create it with a specified amount of memory, you can use the space within it as you like. The DF may be the parent of elementary files or other dedicated files. The number of bytes reserved for the DF must include the size of your data and the operating system's overhead for each file (this information is available in the datasheet). A dedicated file is generally used as an application identifier. The card we use can have three levels of Dedicated Files under the Master File. Each level may have its own keys and access rights. Multiple levels of DFs should not be required for most applications.

Once you have the file sizes, access conditions and the key numbers/references organized, the actual call to create a file is a simple matter.

7. Building a Smart Card Application

Central to developing any application is how you expect the user to use the product itself. Therefore, the first step in developing a Smart Card Application is to clearly organize the various scenarios which the user will encounter while using the product and how the product itself should respond to those situations.

In this article we shall provide the basic framework needed to create a simple Smart Card Application which permits or denies entry into a secure building or room in your office. Although we assume the presence of a centrally networked "gate keeper" device, we can use the computer and the attached Smart Card Reader as its equivalent. It can be assumed that there is a physical mechanism within control of

the “gate keeper” that allows or denies entry into the secure area. Additional features such as – maintaining a record of successful or failed attempts can easily be implemented by the reader.

1. All users are issued SmartCards containing their unique and secret User ID.
2. When attempting to enter a secure area, they have to insert their card into a “gate keeper” device and wait for entry permission.
3. The device should verify that the Smart Card presented to it is indeed a genuine one and not been tampered with.
4. It should read the User ID and check against a list of predetermined users to determine if the user is on the “allowed” list.
5. If the User ID is a valid one, it should permit entry. If not, it should alert the central server and prevent the door from opening.

With these points in mind, we can design the ‘file structure’ on the Smart Card which enables us to implement the above.

8. File Structure for Sample Application

| File | Description | Bytes required by data | Bytes required by system |
|-------------|---|------------------------|--------------------------|
| UserIDFile | An unique number to distinguish the User’s identity | 4 | 12 |
| PVKeyFile | File containing the Protection/Verification Key | 10 | 12 |
| CalcKeyFile | File containing the Calculation Key | 10 | 12 |
| | | | = 24 + 36 = 60 |
| DF Total | The DF needs to contain all other files | 60 | 16 |
| Total | | | = 60 + 16 = 76 |

While declaring the DF file, we will have to specify 60 bytes as the amount of memory we require. Note that the DF file itself has an overhead of 16 bytes. So this application will occupy a total of 76 bytes on the card.

File Permissions

| | Create_ File | Read_ Record | Write_ Record | Update_ Record | Unblock |
|--------------------|--------------|--------------|---------------|----------------|---------|
| DF | Always | N/A | N/A | N/A | N/A |
| UserIDFile | N/A | PV[0] | Always | PV[0] | N/A |
| PVKeyFile | N/A | Never | Always | N/A | No |
| CalcKeyFile | N/A | Never | Always | N/A | N/A |

| | | | | | |
|-----------------------|-----|--------|-------|-------|-----|
| BookRecordFile | N/A | Always | PV[0] | PV[0] | N/A |
|-----------------------|-----|--------|-------|-------|-----|

Note: PV[0] indicates that we use the 0th Protection/Verification Key (ie, the first one).

9. The PCSC_WRAPPER Class

The PCSC_WRAPPER Class contains a number of functions, the most important of which are listed below.

| Function | Description |
|--|---|
| int establish_pcsc_connection(); | Connect to the PCSC framework. |
| int connect_reader(const char *reader_name); | Use the specified reader as the default reader. E.g.: "SCM Microsystems SCR301 0" |
| int wait_for_card_ins(); | Wait indefinitely for a smart card to be inserted. |
| int wait_for_card_ins_once(); | Just check once if the card has been inserted. This function will not "hang" the system in Windows, if used properly with other messaging mechanisms. |
| int connect_card(); | Connect & send a "Reset" to the smart card. |
| int begin_transaction(); | Mark the beginning of a transaction. |
| int select_file(unsigned short id); | "Select" the file given its file ID as the argument and perform all operations on this file until another file is chosen . Use this command to select DFs also. |
| int get_challenge(unsigned char *chlng); | Get the "challenge" from the smart card for external authentication. The argument is the 8 byte array – to store the number received. |
| int external_authenticate(unsigned char *data, unsigned char key_num); | Do the external authentication*. "data" is the encrypted result generated response to the previous call to get_challenge. "key_num" is the key number to authenticate with. |
| int internal_authentication(unsigned char *data, unsigned char key_num); | Internal authentication*. "data" – the challenge to be passed on to the smart card. "key_num" - keynumber to authenticate with. |
| int get_response(unsigned char *data, unsigned char len); | Get response to challenge for internal authentication. "data" – the "encrypted" data is returned in this. "len" is the expected |

| | |
|---|--|
| | length. |
| int read_record(unsigned char rec_no, unsigned char *data, unsigned char length); | Read record from a file – where “rec_no” is the record number to be read. “data” is where the data should be stored. “length” is the length of the record to be read. |
| int write_record(unsigned char *data, unsigned char length); | Writes a record in the next vacant place in the file. “data” - data to be written. “length” - length of the data. |
| int update_record(unsigned char rec_no, unsigned char *data, unsigned char length); | Update a record at location “rec_no”. “data” is what is to be written, “length” is the length of data. |
| int read_binary(unsigned short offset, unsigned char *data, unsigned char length); | Read Binary data. “offset” - offset to read from. “data” – where the data should be stored. “length” - length of data to be read. |
| int update_binary(unsigned short offset, unsigned char *data, unsigned char length); | Update Binary data. “offset” - offset to write to. “data” - data to be written. “length” - length of data to be written. |
| int create_file(unsigned char ftype, unsigned short fid, unsigned short bsize, unsigned short ac, unsigned short kn, unsigned char rl); | <p>Create a file.</p> <ul style="list-style-type: none"> • “ftype” – the file type to be created (Choose from macros in pcsc_wrapper_defs.h). • “fid” – the unique file id to be assigned to this file. • “bsize” – the total file size (including operating system overheads). • “ac” - access conditions (Select from macros defined in pcsc_wrapper_defs.h). • “kn” - key numbers references (Select from macros defined in pcsc_wrapper_defs.h). • “rl” - record length (for record files only). |
| int end_transaction(); | Mark the end of a transaction. |
| int release_card(); | Release the handle associated with the smart card. |
| int wait_for_card_rem(); | Wait for user to remove the smartcard. |
| int disconnect_reader(); | Disconnect from the reader. |
| int disconnect_pcsc(); | Disconnect from the PCSC framework. |
| int trans_status(); | Return the status of the transaction. |
| | |

| | |
|--|--|
| Other functions | |
| int search_readers(char *reader_name[], int *alloc_space); | Detects all readers connected to the system. |
| int card_status(); | Returns the current status of the card. |
| int select_root(); | Select the Master File. |
| int verify(unsigned char *pin, unsigned char kn); | Verify with the "pin" the key number "kn". |
| int unblock(unsigned char kn, unsigned char *pin, unsigned char mode); | Unblock a blocked key. Apply against "kn" key number, the "pin" (Pin/Key Value) with the command UNBLOCK_PIN or UNBLOCK_PUK. |

* External functions which do DES encryption must be called before this function.

10. PCSC_WRAPPER Macros

A number of macros are already defined for you in the file: pcsc_wrapper_defs.h. Some are listed below for your convenience.

File Types

| | |
|-----------------------|------|
| DEDICATED_FILE | 0x38 |
| LINEAR_ELEM_FILE | 0x02 |
| CYCLIC_ELEM_FILE | 0x06 |
| CALC_KEY_FILE | 0x12 |
| VER_KEY_FILE | 0x32 |
| PIN_KEY_FILE | 0x22 |
| ELECTRONIC_PURSE_FILE | 0x1e |
| TRANSPARENT_ELEM_FILE | 0x01 |

Access Conditions

| | |
|--------------------|----|
| CREATE_FILE | 0 |
| UPDATE_RECORD | 0 |
| READ_RECORD | 4 |
| WRITE_RECORD | 12 |
| UPDATE_BINARY | 0 |
| READ_BINARY | 4 |
| LOCK_UPDATE_BINARY | 8 |
| UNBLOCK | 8 |

Access Conditions

| | |
|-------------|--------------|
| ALWAYS(id) | (0x0 << id) |
| NEVER(id) | (0x0f << id) |
| AUT(id) | (0x02 << id) |
| PIN(id) | (0x01 << id) |
| PIN_AUT(id) | (0x03 << id) |

E.g.: Suppose we want to create an elementary file where

1. READ_RECORD is always allowed

2. UPDATE_RECORD is based on PIN
3. WRITE_RECORD is through external authentication

The corresponding access condition would be

ac = ALWAYS(READ_RECORD) | PIN(UPDATE_RECORD) |
AUTH(WRITE_RECORD)

Macros to ease creation of key numbers

| | |
|------------------|-------------|
| KEYGEN(oper,num) | (num<<oper) |
|------------------|-------------|

E.g.: Suppose we want to create an elementary file where

1. READ_RECORD has to be authenticated by key number 0 in key file
2. WRITE_RECORD has to be authenticated by key number 3 in key file

The key statement would be

key= KEYGEN(READ_RECORD,0) | KEYGEN(WRITE_RECORD,3)

Unblock Macros

| | |
|-------------|------|
| UNBLOCK_PIN | 0x00 |
| UNBLOCK_PUK | 0x01 |

For use with the unblock() command. We can unblock a locked PIN value using the locked PIN value or using the PIN Unblock Key value.

11.Code for writing to the SmartCard

[See listing of file: create_card.cpp]

We used the "FreeSec: libcrypt for NetBSD" by David Burren for DES encryption/decryption. You can use the built-in functions in Linux or download one of the many sources available on the net.

12.Pseudo-code for "gate reader"

The "gate reader" has to function in a manner similar to the following:

1. Wait for card to be inserted
2. Once card is inserted,
 - a. Begin transaction
 - b. Select Application Directory, fail if not found (goto g).
 - c. Do External Authentication,
 - i. Send request for 'random number' from Smart Card (for External Authentication)
 - ii. Receive 'random number'
 - iii. Apply PV Key to 'random number'
 - iv. Send number back to Smart Card
 - v. Receive confirmation of success, else fail (goto g).
 - d. Do Internal Authentication,

- i. Send a 'random number' to the Smart Card
 - ii. Wait for response from Smart Card
 - iii. Receive 'number' from Smart Card
 - iv. Apply Calculation Key to the 'random number' which was sent
 - v. Compare this result with 'number' received
 - vi. If they match, continue, else fail (goto g).
 - e. Read User ID from the Smart Card
 - f. If User ID is allowed access to secure area,
 - i. Open door
 - g. End transaction
3. Wait for card to be removed

13.For further information/Resources

Smart Card Manufacturers

GemPlus

<http://www.gemplus.com/>

5/7, 2nd Floor, Vasant Vihar,
New Delhi - 110 057

Phone: +91-11-6153533/34/37/38

vijay.parthasarathy@gemplus.com

27, 80 Feet road, HAL III Stage, Indiranagar,
Bangalore - 560 075, Karnataka

Phone: +91-80-5288247/57, +91-80-5213536/37

Schlumberger

<http://www.slb.com/smartcards/>

The Capital Court, 4th Floor

LSC Phase – III, Olof Palme Marg, Munirka

New Delhi - 110067

Phone: +91 11 610 6277 / 6711 / 6765

Email: indiamarketing@new-delhi.tt.slb.com

Others:

http://directory.google.com/Top/Computers/Hardware/Systems/Smart_cards/Manufacturers/

Smart Card Reader Manufacturers

Gemplus (<http://www.gemplus.com/>)

SSP Solutions/Litronic (<http://www.sspsolutions.com/>)

SCM Microsystems (<http://www.scmmicro.com>)

Others:

http://directory.google.com/Top/Computers/Hardware/Systems/Smart_cards/Readers/

Other Links:

alt.technology.smartcards FAQ (contains details on General Info, Standards, Hardware Manufacturers, Operating Systems, Types of Cards, and Management System)
<http://www.scdk.com/atsfaq.htm>

Movement for the Use of Smart Cards in a Linux Environment (MUSCLE) <http://linuxnet.com/>

Dallas Semiconductor's "Ibuttons"
<http://www.ibutton.com>

[http://directory.google.com/Top/Computers/Hardware/Systems/Smart cards/](http://directory.google.com/Top/Computers/Hardware/Systems/Smart_cards/)

References:

1. "Smart Cards: A Case Study", Jorge Ferrari Robert Mackinnon, Susan Poh Lakshman Yatawara, IBM (<http://www.redbooks.ibm.com>)
2. "Smart cards: A primer", by Rinaldo Di Giorgio (<http://www.javaworld.com/javaworld/jw-12-1997/jw-12-javadev.html>)
3. MicroPayflex / Payflex 1K Reference Manual, Schlumberger Inc.

(Siddhant Bhansali works with Kritikal Solutions Private Limited, an Embedded Systems startup funded by the Indian Institute of Technology, New Delhi. This article was written as part of his work there on SmartCards.)