

# Communication with a Smart Card using Perl and Mathematica: Application to Testing in a Public Key Context

EXTENDED ABSTRACT

Gemplus, Product Security Group,  
B.P. 100, 13881  
Gémenos Cedex, France  
{Laurent.Gauteron, Pierre.Girard, Jean-Luc.Giraud,  
Ludovic.Rousseau}@gemplus.com

Laurent Gauteron      Pierre Girard      Jean-Luc Giraud  
Ludovic Rousseau

September 28, 2000

## Abstract

In this paper we introduce a Perl module and a Mathematica package to communicate with smart cards. We compare these tools with existing solutions in other languages and demonstrate their usage to write a test program of a public key card. We also give a design overview of our solution including the underlying library dealing with the TLP protocol, currently used for the communications between the computer and the card reader.

## 1 Introduction

Applications communicating with smart cards are now written and deployed routinely in the framework of large projects thanks to development kits and libraries for different cards and readers.

But when developers face the problem to communicate with a smart card for testing or debugging purpose there is a lack of flexible and powerful tools. Interpreted languages such Perl or Mathematica have demonstrated to be powerful environments to develop quickly small pieces of code for diagnostic, test or prototype purposes. They are still good candidates for such tasks in the smart card field. In this paper we present a Perl module and a Mathematica package and we motivate their usage for testing a public key enabled smart card.

We first give some basics on public key cryptography and on GPK8000, Gemplus' generic public key card. In the next section we review the classical solutions that are available to test public key based cards. Then we introduce our solution, beginning with the C library currently implementing the TLP protocol. Finally we successively explain the API, how to use it and the design of the Perl module and Mathematica package.

## 2 Public key cryptography and GPK8000

Public key cryptography deals with algorithms designed to use two different keys for the ciphering (or signature verification) and deciphering (or signature) operations. The two keys are related but it is not possible to deduce one from the other in a reasonable time. The ciphering (or signature verification) key is made public (and called the public key) so that the problem of secret key exchange is avoided. The deciphering (signature) key must be kept secret and is called the private key.

The most widely used public key algorithm is probably RSA [9] which has been introduced more than twenty years ago. It use two large prime numbers  $p$  and  $q$  that must be kept secret. The public key is made of the product  $n = p \times q$  and another number (denoted  $e$  for encryption purposes and  $v$  for signature verification purposes) relatively prime to  $(p - 1)(q - 1)$ . Generally  $e$  (resp.  $v$ ) is chosen to be a small number like 3, 17 or  $2^{16} + 1$ . The private key (denoted  $d$  for deciphering and  $s$  for signature) is computed as  $d = e^{-1} \bmod (p - 1)(q - 1)$ .

The encryption (signature verification) of a message  $M$  is computed as follows:  $C = M^e \bmod n$ .

The decryption (signature) of a cipher text  $C$  is computer as follows:  $M = C^d \bmod n$ .

Usually, for a signature operation the message  $M$  to be signed is not the entire data or file but a digest of it computed with an algorithm such as SHA-1 or MD5. The format of an RSA signature of a message digest has been standardised in PKCS#1[10].

Private keys are usually extremely valuable as they are used to generate signature which now have a legal value in some countries[3]. To protect those keys, it is strongly advised, and, in some cases, required by the law to store them inside a

tamper resistant device like a smart card. The device should be used, not only to store the keys, but also to compute the signatures so that the private key is never sent outside the card. In this case, a smart card which is able to compute an RSA signature is required. The Gemplus GPK product range (including the latest card in this range, the GPK8000) allows to compute such signatures and to store a few private and public keys.

In this paper we suppose that a private/public key pair for RSA is generated outside the card and is loaded on the GPK8000 by a dedicated program. We demonstrate how to use our API in Perl or Mathematica to test if a card performs a signature correctly, i.e. if the key generation, the key loading and the key usage are correct.

If we suppose that a key has been loaded on a GPK8000, a signature can be obtained by performing the following commands:

1. Select the master file of the card or the DF where the key file is located (SELECTFILE command);
2. Unlock the private key file by submitting the PIN code to the card (VERIFY command);
3. Select a cryptographic mode, that is a signature algorithm (RSA, DSA or DES), a message digest algorithm (MD5 or SHA-1) and a key file; in our case, we will use RSA signature with MD5;
4. Send the data to be signed to the card (PUTCRYPTODATA command);
5. Ask the card to compute and send the signature (PK\_SIGN command).

The test will consist in performing the above sequence of commands and compare the signature computed by the card with the signature computed on the same data with the same key by the test program.

In the next section, we give an overview of usual ways to implement such a test with classical programming languages.

### **3 Classical solutions in C, C++ and Java**

In C and C++, developer willing to deal with public enable smart card would have to use a large integer library able to compute modular exponentiation on numbers up to thousands bits long such as MIRACL[11] or LiDIA[5]. These libraries have very good performances, however, they are overly complete for our purposes and need a heavier programming style including memory management.

In Java, the OpenCard Framework[2] allows developers to communicate easily with smart cards and the standard Java2 class `java.math.BigInteger` permit modular exponentiation on large numbers. However, we are still in a compile and run programming style and powerful features such pattern matching or regular expression still lack.

## **4 A portable C library to communicate with readers**

The library described in this chapter is *not* the Gemplus standard library given with development kits. Ours is much less powerful but also much more simple to understand and enrich. We put the emphasis on simplicity to enable any developer to easily access smart card maybe without giving him access to all functionalities of the reader. This library is then more suited for prototyping and validation of smart card applications than for the development of full fledged solutions.

### **4.1 Why C ?**

The first building block of the system presented in this paper is a C library. C was chosen because we wanted to have a very small and easy to use library. Furthermore, C is well suited to develop low-level libraries and to debug serial IOs.

One of our goals was also to give the opportunity to developers to write their own version of the library in their favourite language. All developers probably know enough C to understand the library and then rewrite it. On top of this, most languages/programs allow the use of libraries developed in C as plug-ins to add new APIs. This is at least the case for Mathematica and Perl.

Communications with smart cards are byte-oriented. It is therefore very easy to write a portable C library. We tested it on a Unix platform (Linux x386) as well as Windows NT.

### **4.2 Structure**

The library is written in an object oriented fashion to ease the process of adding support for new readers. Explanations and schematics describing the structure of this library will be added in the final version of the paper.

### **4.3 API**

The API of the library is small and simple to allow maximum usability. There are 3 main families of functions : functions to control the communication (opening of a channel), binary commands that use binary values as input (unsigned char, int,...

) and ASCII commands which have strings as parameters. The API will be fully described in the final version of the paper.

## 4.4 Supported readers

The library currently supports readers implementing the TLP protocol. This protocol is different from the T=0 protocol used to communicate between the card and the reader. The TLP protocol is very simple and lightweight but only works for serial readers. We will describe it in the full version of the paper. The TLP protocol is implemented by a lot of smart card readers for backward compatibility. The following readers have been tested on Intel platforms (Linux and Windows): GCR 200, GCR400, GemPC 410, GCR 500 and GemSelf 800 (contactless).

## 4.5 Limitations

The library supports TLP and TLP-like readers (like the GemSelf 800). Some compatibility issues might appear with readers we did not test because they might implement commands that differ from the one used in Gemplus readers. These incompatibilities should not be difficult to correct with a little documentation on the target reader.

Readers using other communication protocols are not supported but we would be really pleased to receive contribution code to add to the distribution of the library to support a broader range of readers. Please e-mail potential contributions to [Laurent.Gauteron@gemplus.com](mailto:Laurent.Gauteron@gemplus.com).

The current implementation does not support any speed of communication other than 9600 baud. Cards communicating with the T=1 protocol have not been tested.

# 5 Mathematica

## 5.1 Why Mathematica ?

Mathematica is a widely used program in the community of mathematicians for its powerful symbolic math capabilities. It is also used by other scientists for its advanced built-in pattern matching and for its functional programming language. It is indeed a real swiss-army knife for scientists. A really interesting feature for smart card testing, is that it is naturally able to compute operations with integers of arbitrary length. It is therefore really useful for cryptographers working on public key cryptosystems. With the development of public-key enabled smart cards, verification and testing of programs has become more difficult than before. Lots of

people using these cards for evaluation purposes either have to develop their own code (possibly C) to compute and verify public key signatures or have to copy and paste results given by the card in a Mathematica notebook. Having access to smart cards directly through the Mathematica kernel would help developers greatly and ease the prototyping and debugging of a smart card aware application. This was the main motivation for the bringing together of a Mathlink package to communicate with cards.

On top of this, Mathematica is available for a wide range of platforms, from Windows to Macintosh, including most UNIX systems (Linux, Sparc, Digital Unix, HP-UX, RS/6000, IRIX,...).

## 5.2 API

The Mathematica package provides functions that are very similar the C library API.

- `Integer InitSerial[string Device]`
- `Integer SetReaderType[Integer Handle,Integer Reader-Type]`
- `Integer PowerDown[Integer Handle]`
- `String ResetCardAscii[Integer Handle]`
- `List_Number ResetCardBinary[Integer Handle]`
- `String SendIncomingAPDUAscii[Integer Handle, String APDU]`
- `String SendOutgoingAPDUAscii[Integer Handle, String APDU]`
- `String SendIncomingOutgoingAPDUAscii[Integer Handle, String APDU]`
- `List_Number SendIncomingAPDUBinary[Integer Handle,Integer CLA, Integer INS, Integer P1,Integer P2, Integer L,List_Number Data]`
- `List SendOutgoingAPDUBinary[Integer Handle,Integer CLA, Integer INS, Integer P1,Integer P2,Integer L]`

- `List SendIncomingOutgoingAPDUBinary[Integer Handle,Integer CLA,  
Integer INS,  
Integer P1,Integer P2,  
Integer L,  
ListNumber Data_in]`
- `Integer GetRerrno[ ]`

The first functions deal with the establishment and closing of the communication channel. The next ones are used to exchange commands with the card for incoming (i.e. with data sent to the card) or outgoing (i.e. with data received from the card) data and with both incoming and outgoing data (the `GET_RESPONSE` command is automatically issued by the library if necessary).

### 5.3 Example

In the final version of this paper, we will include an example and commentary on how to use of the Mathematica package to verify a GPK signature.

### 5.4 Design

The MathLink package is a wrapper around the C library. The wrapper has three parts : a TM file and two C files. The TM file describes the link between the name of the function of the C library (which has to be Mathematica aware, as far as Data format is concerned) and the name of the functions that will be available in Mathematica. The `mprep` command applied on the TM file will generate a C file to compile and link with the add-on library and Mathematica libraries. In our implementation, the `MultiReader.c` file implements functions to allocate and de-allocate some cardreader structures. These operations are implemented in the Mathlink package. The `mathreader.c` contains the body of the Mathematica functions corresponding to the declaration made in the TM file.

## 6 Perl

### 6.1 Why Perl?

Perl [7] is a free software distributed under the GNU General Public Licence [6] and the Artistic Licence [4]. Perl exists for many operating systems (Unix, Windows, MacOS and more than 60 others [8]).

The advantages of using Perl are many. Perl is an interpreted and high level language. You can easily create complex data structures like lists and associative tables. Perl is often used as a glue between different programs or computing resources. Perl can be extended using modules. Modules can be seen as extensions to the language to provide new functionalities. Many Perl modules exist to communicate with programs and external services. You can find modules to communicate with an LDAP (Lightweight Directory Access Protocol) server or do authentication with PAM (Pluggable Authentication Module), use SSL (Secure Socket Layer) as a client or server, etc.

The Perl language is then well suited for rapid development of (small) applications like the ones used to debug or test a smart card functionality.

The advantage of Perl over C is a transparent allocation/deallocation mechanism. You don't need to bother with malloc or free. Memory and other resources are automatically released when the object is destroyed.

## 6.2 API

As described in the manual page (`cardreader(3pm)`) the methods exported by the `cardreader` module are:

- `$desc = new cardreader($type, $device);`
- `$atr = $desc -> ResetCard;`
- `$sw = $desc -> SendIncomingAPDU($apdu);`
- `($res, $sw) = $desc -> SendOutgoingAPDU($apdu);`
- `($res, $sw) = $desc -> SendIncomingOutgoingAPDU($apdu);`
- `$desc -> PowerDown;`
- `$error = ISO7816_ErrorStr($sw);`

The main differences with the C library are:

- the Perl module only uses ASCII strings. For instance, byte value `0x42` is coded by the text string `"42"`.
- the methods `SendOutgoingAPDU` and `SendIncomingOutgoingAPDU` return a list containing two scalar elements: the result and the status word.
- the serial port allocation and release is transparent. The port is allocated (opened) when a new `cardreader` object is created. The port is closed when the object is dereferenced.

- the method `ISO7816Error` return the error message in text corresponding to the status word passed in parameter.

### 6.3 Example

```

1
2  #!/usr/bin/perl -w
3  use strict;
4  use cardreader;
5
6  my ($scr, $sw, $cmd);
7
8  ##### new #####
9  # allocate a descriptor
10 # the default device is /dev/screader
11 $scr = new cardreader or die "Can't allocate reader: $!\n";
12
13 ##### ResetCard #####
14 print "ATR: ", $scr -> ResetCard, "\n";
15
16 ##### SendIncomingAPDU #####
17 # CLA = 0
18 # INS = A4 SELECT FILE
19 # P1 = 01
20 # P2 = 00
21 # Lc = 02
22 # Data = 0100
23 $cmd = "00"."A4"."01"."00"."02"."0100";
24 print "=>$cmd\n";
25 $sw = $scr -> SendIncomingAPDU($cmd) or print "Send-
IncomingAPDU: $!\n";
26 print "sw $sw, ", $scr -> ISO7816_ErrorStr($sw), "\n";
27
28 ##### SendOutgoingAPDU #####
29 # CLA = 0
30 # INS = C0 GET RESPONSE
31 # P1 = 00
32 # P2 = 00
33 # Lc = depends
34 # Data =
35 $cmd = "00"."C0"."00"."00".(substr $sw, 2);

```

```

36 print "=>$cmd\n";
37 ($data, $sw) = $scr -> SendOutgoingAPDU($cmd) or print
38 "SendOutgoingAPDU: $!\n";
39 print "sw $sw, ", $scr -> ISO7816_ErrorStr($sw), "\n";
40 print "Data $data\n";
41
42 #####          PowerDown          #####
43 $scr -> PowerDown or die "PowerDown: $!\n";
44
45 undef $scr;

```

The example is very simple. Line 4 includes the module. Line 11 creates an instance of a new cardreader object. `$scr` is the reference to this object. Line 14 reset the card and prints the ATR returned. Line 23 creates a string variable containing the command. The variable is used to ease reading. Also each ISO 7816-4 fields is clearly separated. A4 is the instruction for SELECT FILE. The file ID selected is 3F00. We then need to retrieve the information returned using a GET RESULT command. The value of `Le` is the number of bytes to read returned by the SELECT FILE command. Line 43 powerdown the card. The object referenced by `$scr` is implicitly destroyed when the script exists. We could use `undef $scr` to explicitly destroy it and close the allocated (serial) port.

With the Gemplus badge the output of this script is:

```

ATR: 3B241100000080729443
=>00A40100020100
sw 611A, 611A: 26 bytes of response still available.
=>00C000001A
sw 9000, 9000: Normal processing
Data 8510C101010038000006010001000000006C840653595354454D

```

The source code archive contains many other examples: a text interface to interactively send commands, a GPK signature test, a Web interface to communicate with a smart card using a Web browser talking to an ASP (Active Server Page) script using the Apache Web server.

## 6.4 Design

The `cardreader` Perl module is a wrapper around the C library. The wrapper has two parts: an XS file and a Perl file. XS is a language used to create an extension interface between Perl and a C library.

The C library described in § 4 uses C structures to store information about the reader. The C structures are seen by Perl as a typed pointer passed to C functions. Perl does not know what is inside the C structure.

The XS file (`cardreader.xs`) calls the low-level C functions and converts the arguments and results in the right Perl format. This file also sets the Perl error variable (`$!` or `$ERRNO`) to the correct numerical and string value (for example “card absent” or “wrong number of parameters”). This file also contains the XS code called when a `cardreader` object is dereferenced (destroy method).

The `.pm` file (`cardreader.pm`) is the core of the Perl module. It contains the new method and wrappers to low level functions provided by the XS file. This file also contains the documentation in POD (plain old documentation) format. The documentation can be converted to text, html or manpage format.

## 7 Resources

The final version of the paper will detail the different resources (modules, packages, libraries, manuals, examples and source codes) which will be released on the Gemplus developer site before the conference.

You can find sources (C, Perl and Mathematica) and binaries (for Windows) on [http://www.gemplus.fr/developers/technologies/tlp\\_drivers/](http://www.gemplus.fr/developers/technologies/tlp_drivers/).

## 8 Conclusion

We presented in this paper a small and easy to use C library that has been used to enrich powerful prototyping and development tools like Perl or Mathematica. An example illustrates how these tools ease the management of public key cards. Source code of all libraries will be available on the Gemplus developer’s site for developers to patch and upgrade according to their needs. We hope that they will send us their patches so that we include them in the distribution.

## References

- [1] *gp - PARI calculator*. <ftp://megrez.math.u-bordeaux.fr/pub/pari>.
- [2] *OpenCard Framework*. <http://www.opencard.org>.
- [3] *Directive 1999/93/EC Of The European Parliament on a Community framework for electronic signatures*, December 1999.
- [4] *The “Artistic License”*. <http://www.perl.com/language/misc/Artistic.html>.

- [5] Darmstadt University of Technology. *LiDIA, a C++ Library for Computational Number Theory, version 2.0*. <http://www.informatik.th-darmstadt.de/TI/LiDIA/welcome.html>.
- [6] Free Software Foundation. *GNU General Public License*, June 2 1991. <http://www.fsf.org/copyleft/gpl.html>.
- [7] *The Source for Perl*. <http://www.perl.com/>.
- [8] *CPAN: Perl Ports (Binary Distributions)*. <http://www.cpan.org/ports/index.html>.
- [9] R. L. Rivest, A. Shamir, and L. Adleman. A Method for Obtaining Digital Signatures and Public-Key Cryptosystems. *Communications of the ACM*, 21(2):120–126, February 1978.
- [10] RSA Laboratories. *PKCS#1 v2.0: RSA Cryptography Standard*, September 1998. <http://www.rsasecurity.com/rsalabs/pkcs/pkcs-1/index.html>.
- [11] Shamus Software Ltd. *MIRACL Library, version 4.24*. <http://indigo.ie/~homedirmscott>.