Some security problems raised by open multiapplication smart cards

Serge Chaumette Damien Sauveron[‡]

LaBRI, Laboratoire Bordelais de Recherche en Informatique
UMR 5800 – Universite Bordeaux 1
351 cours de la Liberation, 33405 Talence CEDEX, FRANCE.
{serge.chaumette,damien.sauveron}@labri.fr, http://www.labri.fr/

Abstract

Multiapplication smart cards now make it possible to have several applications sharing the same physical piece of plastic.This raises new security problems by creating additional ways to attack them. This is particularly true with future ***open multiapplication cards*** that allow anybody to load his/her own application. For example an attacker can load her own programs to determine the physical signature of instructions using measurements on side channels (electromagnetic or power signal) and build a database of the whole set of instructions in order to reverse-engineer an application already installed on the card. The vulnerabilities of these cards are the topic of this paper. The attacks are described for open multiapplication cards in general and illustrated by means of code samples for Java Cards.

KEYWORDS: *Smart Cards, Java Cards, Open Multiapplication Cards, Security, Attacks.*

1 Introduction

Even though multiapplication smart cards allow several applications to share the same medium, dynamically loading a new application requires the knowledge of authorization keys. With the open multiapplication smart cards this constraint will not exist anymore.

The aim of this paper is to describe new security problems that arise when dealing with these **open** multiapplication smart cards. First we will present the common way to explore and attack classical smart cards and then we will clearly define what are closed multiapplication smart cards and open multiapplication smart cards. In section 4 we will show the general attacks against these new cards. Then, in the context of the open multiapplication smart cards, we will expose in section 5 some physical characterization methods and we will investigate in section 6 how to use them in order to achieve effective attacks on these cards. Finally, we will present our experiments, the related work and we will conclude. The contributions of this paper are the description of the physical characterization methods to characterize a platform and the explanation of the new threats of these methods in the context of open multiapplication smart cards.

2 Exploring and attacking closed smart cards

Even though there are not many ways to explore a closed card (*i.e.* a card that does not allow codes to be uploaded after it has been issued) before trying to attack it, there are at least two possibilities that remain:

[*] LaBRI (Talence, FRANCE) and LMSI (Limoges, FRANCE).

[†] This work was partly supported by a doctoral grant from the french ministry of research and SERMA Technologies.

- Software approach. Since loading code on a closed card is impossible, the only accessible information are its external interfaces. More precisely, the only visible external interface of a card is its communication port. The communication architecture between a smart card and a reader connected to a host can be seen as a protocol stack, from the physical layer to the application layer (*see* Fig. 1). It is described in the ISO7816-3 [1] and ISO7816-4 [2] standards. One common

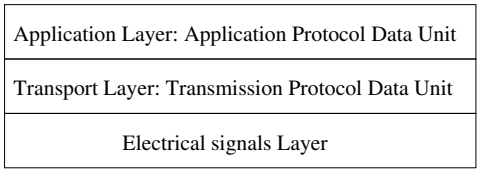

Figure 1: Stack of communication protocols.

possibility to set up an attack at the level of the communication layer is to send all the possible APDU commands to the card, in order to identify all the services available from the outside and to perform fuzzing attacks[1]. But these kinds of attacks are quite inefficient because it is easy to block the unwanted APDUs.

- Hardware approach. Based on the results obtained by Kocher, the main path of attack to explore a card without damaging it, is to observe side channels [3] such as the physical emanations from the chip or the time consumptions [4]. The power analysis (PA) [5, 6] or electromagnetic analysis (EMA) [7, 8] methods make it possible to identify patterns corresponding to operations achieved by the card. These attacks are carried out as a blind man, or a semi-blind man if the specifications of the card operations are known.

  The non-invasive attacks cited above are fundamental in order to succeed to build the dictionary that we propose in section 6.1. Besides, smart cards are prone to many other invasive (e.g. microprobing) and non-invasive (e.g. glitches[2] on the different pads of the card or fault-injection with laser) attacks [9, 10, 11, 12] which are out of the scope of this paper (but they can be used as described in sections 4.3 and 6.2).

3 Two types of multiapplication smart cards

A multiapplication card is able to embed several applications. Today, these applications do not run simultaneously because common OS of smart cards have a single thread. The two main standards of multiapplication cards are Java Card [13, 14] and MULTOS [15]. Recently, two new technologies appeared with the birth of Smartcard.NET [16, 17] by Hive-Minded and of the MultiApplication BasicCard ZC6.5 [18] by ZeitControl. Windows for Smart Cards by Microsoft can also be cited even though it does not seem to be active any longer. Java Card was the first to appear but all these technologies share many concepts anyway. The architecture of multiapplication smart cards (*see* Fig. 2) consists of:

- an embedded Operating System that supports the loading and the execution of several applications. The OS is a runtime environment with a virtual machine (VM) that provides security features (e.g. a firewall between applications).
- the applications that are interpreted by the VM.

3.1 The closed multiapplication smart cards

Until now no multiapplication technology was completely proven secure by formal methods to reach the high level of security required to allow it to run uncertified applications that may embed malicious code to attack the assets of the platform and those of the other applications. To prevent these problems,

[1] Fuzzing attacks consist in giving all the possible values to the parameters of a service.
[2] A glitch is a sudden change in voltage in an electrical current.

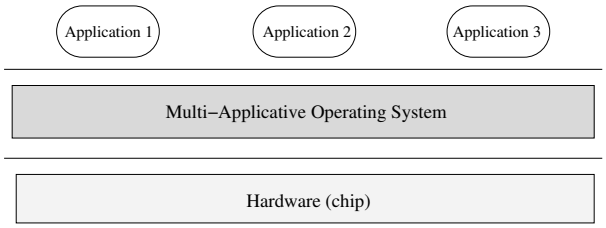

Figure 2: Architecture of a multiapplication smart card.

the smart card issuers began to support standards such as GlobalPlatform [19] which specifies how to securely load, install and manage applications on a card. Indeed this standard is used to easily set up and use cryptographic mechanisms to authorize or prevent the loading of an application on the card. For example the application can be digitally signed off-card by a trusted party, e.g. the card issuer. Once loaded, the card can check the signature and accept or reject the application. The major drawback of this solution is its centralized model because of the trusted third party required to sign the application; it thus decreases flexibility.

For short, if the user does not own the authentication keys, she can only use the card as a closed platform; else, if she owns the keys she will be able to load her code and to apply the internal attacks presented in section 4. She will also be able to use the physical characterization methods shown in section 5 to set up attacks as described in section 6. All these methods are intended to the ITSEF[3] or card manufacturers (who can access the keys) to test the security.

Note that in our paper, GlobalPlatform compliant cards are considered as closed cards since knowledge of keys are needed to load a new application.

3.2 The open multiapplication smart cards

Although Java Card designers first thought it was impossible to embed a verifier due to the resource constraints of the smart cards, on-card verifiers have been developed [20, 21, 22, 23, 24, 25, 26]. Among the different solutions that have been proposed, three of them (*i.e.* the defensive VM, the verifier based on code transformation and the stand-alone verifier) allow to bypass the signature step of the application without jeopardizing the card security and thus allow everybody to freely load his/her own application – *i.e.* anyone can still load a rogue application but it will eventually be rejected by the verifier or blocked by the defensive VM. Note that the verifier based on code transformation [22, 23] requires an off-card program to normalize the application whereas the two others (*i.e.* the defensive VM [24] and the stand-alone verifier [26]) are stand-alone. We only consider as ***open multiapplication cards*** the multiapplication cards based on one of these two stand-alone solutions. Both are equivalent [25] but the defensive VM dynamically checks each executed bytecode whereas the stand-alone verifier statically checks the application once at load time and it is associated with an offensive VM that does not check the bytecode at execution time.

In the past, the FIPR (Foundation for Information Policy Research) thought that the multiapplication cards were a bad idea, except in limited applications such as GSM SIM cards [27]. Nevertheless they are now a reality but they are only available as closed cards. However, even if these open multiapplication smart cards are not yet available on the market, they most probably are the future of smart cards since all the research projects presented above will contribute to make them as secure as required in a near future.

4 Internal attacks

Obviously a common problem of the open multiapplication cards is the possibility to load a malicious application to create internal attacks. Such an application may:

[3]Information Technology Security Evaluation Facility.

- identify available services on the card, collect information and then try to deduce the possible behavior of an official[4] application (since it cannot use unavailable services!);
- directly attack the VM and the firewall.

4.1 Identification of services and collection of information

The preliminary fundamental steps required to attack a card are the identification of the services that it offers and the collection of information regarding its OS and the loaded applications. There are several ways to achieve these operations.

First, to identify the services offered by the card, the documentation may be helpful. If it is not accessible, an attacker can try to load an application on the card to test every service defined in the specifications of the technologies supported by the card. For example, to identify the cryptographic services of a Java Card, she can load an application that uses the `Cipher.getInstance` method in a proper way[5] with all the valid parameters.

Second, if the card supports special mechanisms similar to the GlobalPlatform management functions (e.g. the `GET STATUS` command) the simplest way to collect information is to send the proper APDU commands. Else the attacker can also load an application on the card to achieve this task. For example, to detect all the available applications on a Java Card and to work out if they offer services (through the `Shareable` interface), she can use the `JCSystem.getAppletShareableInterfaceObject` method and fuzz it. Note that this particular method may return false negative results due to access control rules defined by the targeted applications to share their services. If it succeeds and the attacker obtains `Shareable` interfaces, she can then try to apply the illegal reference casting method cited in [28].

4.2 Attacks against the VM and the firewall

There exist many attacks directed against the VM and the firewall but we do not detail them here because they have already been explained in the literature: for example two attacks against the firewall mechanism (AID impersonation and illegal reference casting that provides access to all interface methods of a class) are described in [28]; based on type confusion, attacks against the VM are shown in [29]; there also exist attacks coming from problems in the specifications such as those presented in [30].

A bad specification or a bad implementation can also lead to attacks against the VM. For example the program of listing 1 that forges a `reference` would normally be rejected by an open multiapplication card: at loading time for those based on the stand-alone verifier; at execution time for those based on the defensive VM. But an attacker should always try to load such malicious codes to test if there are implementation problems or not.

The piece of code shown listing 1 (provided in Java Card Virtual Machine bytecode language) forges a `reference` (since the `saload` bytecode reads a `short` at an index in an array and it normally takes two arguments, one of `reference` type – *i.e.* an object, the array to read – and one of `short` type – *i.e.* the index –, but it gets here two arguments of `short` type) and tries to read one `short` of the object by considering it as an array. This function enables (if the call succeeds) to traverse all the allocation table (using all the possible values of `Address`) and to read the contents of all objects as if they were `short` arrays.

Note if the `reference` type is represented as a memory address (*i.e.* the equivalent of a pointer) in the VM implementation this code allows to traverse all the memory.

4.3 Mixed hardware and software attacks

To improve the efficiency of the attacks on the card, it is possible to elaborate simultaneous hardware and software attacks. Using hardware attacks makes it possible to create faults at software level [10, 11, 12] (e.g. to bypass verification). These modifications of the normal behaviour can be exploited by the loaded application to access unauthorized services or information. To perform hardware attacks, it is still possible to use equipments such as a laser, source of heat, etc.

[4] An application installed by some trustworthy organization, e.g. a banking application.

[5] If the card supports the garbage collection mechanism, the method can be called directly anywhere in the code, else, the call should be inserted in the constructor, the application should be installed on the card, it should test if the service is available and it should be deleted to avoid to overflow the heap with multiple instances of the `Cipher` class.

Listing 1: Get short value at the specified address

```
short readShortAtAddress(short Address) {
  sload_1   // Get the Address local variable. We hope this short will be considered as a reference
  sconst_0  // Represent the index 0 in the false array starting at Address
  saload    // Try to get one short and push it on the stack. If this operation succeeds the stack
            // is not well implemented since the saload bytecode should only accept that the first
            // argument is of reference type (and not a short type as Address) and that the
            // object referenced is also a short array
  sreturn
}
```

One problem of these kinds of attacks is the precision required (in space but especially in time) to induce usable faults. To improve it, we need to have a better localization of where the trigger should be placed in the software execution. To achieve this task, we can use the physical characterization methods described in section 5.

For example it is possible to try to bypass the firewall checks with attacks described in section 6.2. A very good paper [31] also illustrates how to achieve mixed hardware and software attacks on the VM of a PC using a lamp as source of heat.

5 Physical characterization methods

Once the services have been identified, an interesting possibility to set up attacks against these services and the applications which use them, consists in observing their physical signatures using measurements on side channels. For example an attacker may load her own programs on an open multiapplication card to determine the physical signature of basic instructions in order to build a database of the whole set of instructions to reverse-engineer (from its execution trace) an official application already installed on the card. In this section we present the applicable physical characterization methods that we identified in the context of open multiapplication smart cards and in the following section we will show how they can be exploited to set up real attacks.

The main physical signals that can be observed come from side channels such as the power consumption, the electromagnetic emissions or the execution time. For instance it is possible to use a cryptographic service and determine the characteristics of the physical signals emitted during the execution of this service (duration in time, location of the best electromagnetic emissions, power consumption, etc.). This characterization can target the use of a whole service or an elementary operation, e.g. the interpretation of a single bytecode or a sequence of bytecodes. Note that in the remainder of this paper, the term "signature" will refer to the physical signature and not to the digital signature discussed section 3.

The difficulty to isolate a pattern from all those composing the trace leads to set up reliable characterization methods to determine the physical signature of the interesting pattern. For example, Fig. 3 shows the difficult task to identify the pattern to observe in the normal execution trace of an applet[6].

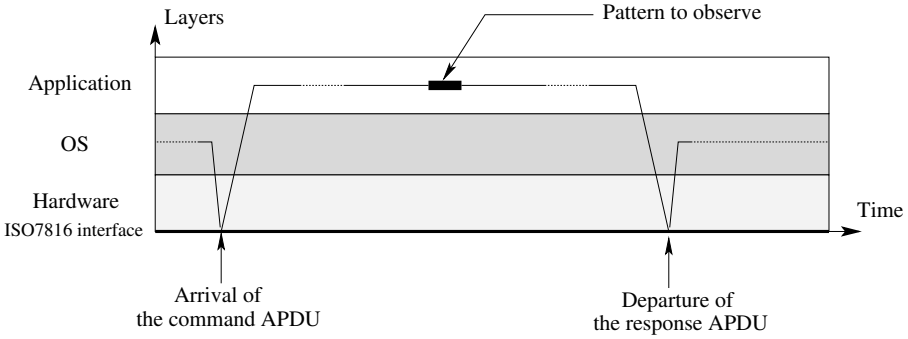

Figure 3: Trace of a normal execution in the layers.

[6]Java Card applications are also called *applets*.

We identified two main ways to easily, efficiently and quickly determine the physical signatures. The first which is also the simplest is achieved by using glitches on the I/O channel of the card, *i.e.* sending output data from the card to the reader. The second consists in causing the pattern to observe to be repeated. In the following sections we present these two approaches and discuss their advantages and drawbacks. We eventually propose a hybrid method that overcomes the problems and takes the best of the two previous methods.

Note that if the physical characterization methods explained below can be applicable on a closed card, it is always in a special context (e.g. the user who wants to attack the card owns the key to load her program). But with open multiapplication smart cards these methods can be applicable without restriction.

5.1 The glitches based method

This method consists in enclosing the targeted pattern with events that are visible from outside the smart card (*i.e.* for the attacker). The only visible event to an external observer that the card can produce is the emission of bytes on the communication channel. Fig. 4 shows that it is easier to locate a pattern in the trace if the execution is glitched using this event.

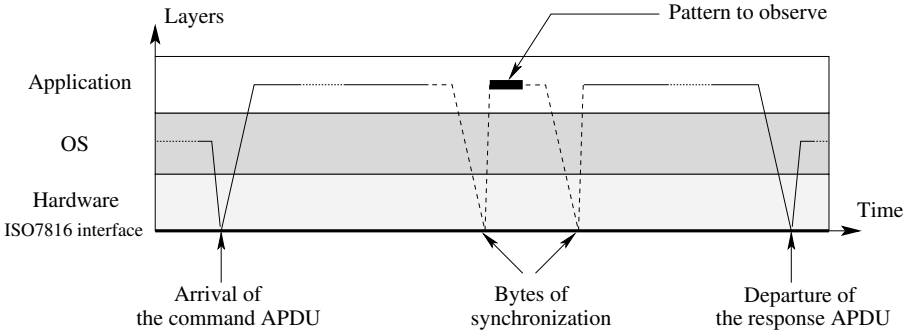

Figure 4: Trace of a glitched execution in the layers.

The code inserted to trigger the glitches causes an overhead and thus the pattern to observe does not appear at the same instant in the first trace and in the second trace (*i.e.* the pattern in the second trace is time shifted).

Another advantage of this approach is that the trace to save is shorter (because the area to observe is smaller – *i.e.* only the part of the signal between the glitches) and thus it is possible to get a better sampling of the signal, still producing the same amount of data.

For more details, the reader is refered to one of our previous papers [32]. It presents the methodology to easily set up test suites with the tools provided by the JCatools suite [33, 34]. The JCatools suite was developed during the Java Card Security project between the LaBRI and the ITSEF of SERMA Technologies. The paper also describes how to modify the CAP[7] file by working at bytecode level to improve the precision of the observation.

5.1.1 Preventing this characterization method from being used

Some solutions to prevent this characterization method from being used can be implemented. For instance, it is possible:

- to introduce a random delay on the I/Os to block these attacks. But it is a bad solution because with many tries it should still be possible to cope with this randomness.
- to store all the data in a buffer until the end of the execution before sending the buffer on the I/O. This would most likely require a second buffer much bigger than what remains acceptable on a card.

[7]The standard binary file format of the Java Card platform.

- to store the data in a buffer until their size reaches a fixed threshold before sending them on the I/O. In this case the hacker may instrument her applet to generate the exact amount of data needed to obtain a response on the I/O and to work out the beginning of the pattern to observe.
- to store the data as previously in a buffer but with a random threshold. A hacker could still bypass this security using a probabilistic technique.

For all these reasons we believe that mixing a random delay and a random threshold storage of the data in a buffer seems to be a solution that may be worth considering.

Note that even if the manufacturers find a better other, effective, countermeasure, it will still be possible to locate the pattern by enclosing it with something that induces high power consumption (e.g. a cryptographic mechanism) to produce an event visible from the outside.

5.2 Repeated pattern based method

Since the manufacturers can add the countermeasures proposed above, we consider another method that still makes it possible to capture the signature of the operations achieved on the card. This solution consists in repeating many times an identical sequence so that it is easier to locate the area to study and then to identify the elementary pattern.

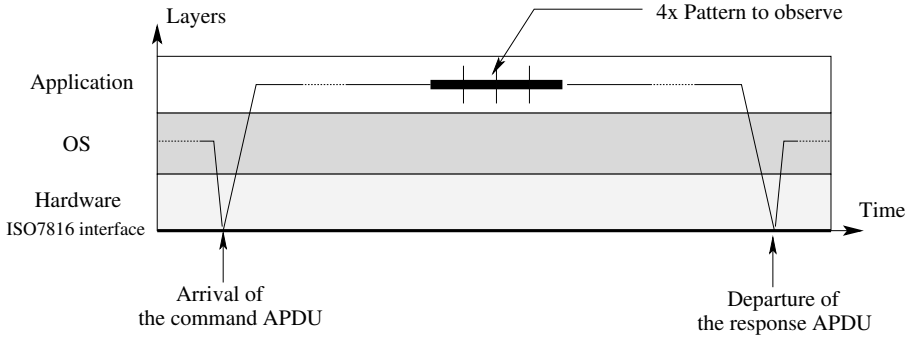

Figure 5: Trace of a multiple patterns execution in the layers.

For instance inserting the code shown listing 2 in a Java Card applet makes it easier to find the `dup` elementary pattern. The physical emanations corresponding to the execution of the above program will

Listing 2: Patterns method

```
...
// assume that there is at least one element on the stack
dup     // pattern to observe
dup     // pattern to observe
dup     // pattern to observe
dup     // pattern to observe
...
```

contain four times the pattern corresponding to the `dup` operation.

5.2.1 Preventing this characterization method from being used

The first solution to prevent such a characterization method from being used is to put constraints on the on-card verifier to check for instance that the loaded code does not contain a sequence of two `dup` bytecodes, *i.e.* `dup dup`. Indeed the converter will produce a `dup2` and not this sequence. But it is not possible to do such verifications for all the possible bytecodes due to the fact that it would require a verifier capable of understanding the semantic of the sequence of bytecodes at application level (*i.e.* what the application wants to do). The verifier can only understand the semantic of the sequence of

bytecodes at VM level (*i.e.* if the chain of bytecodes is valid). A defensive VM can do the same checks but the solution remains the same.

A better solution would be to use hardware countermeasures (e.g. a random internal clock or asynchronous processors [35, 36]) to better jam the physical emanations.

5.3 The hybrid method using both glitches and patterns

This method minimizes the overhead caused by the transitions between the application layer, the OS layer and the chip layer (only two glitches for the sequence) due to the insertion of glitches. It allows to easily locate the sequence of identical patterns in the trace and then it is easy to find the elementary pattern in the located sequence. An example of this hybrid method to observe `sxor` is shown listing 3.

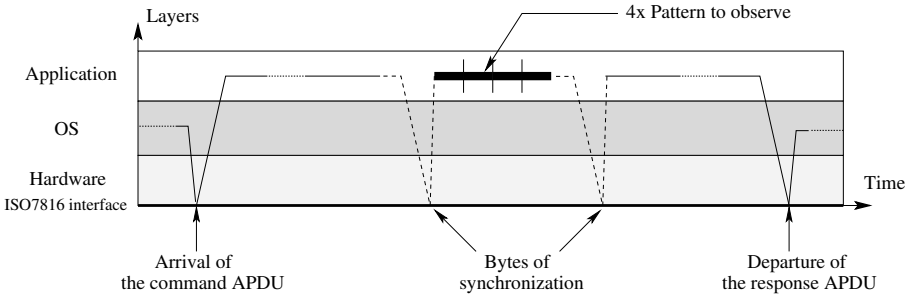

Figure 6: Trace of a glitched multiple patterns execution in the layers.

Listing 3: Sample code used by the hybrid method

```
...
aload_1     //
sconst_0    // Here are
sconst_m1   // several arguments
sconst_m1   // needed to the
sconst_m1   // execution of
sconst_m1   // the patterns
sconst_1    // to observe and
aload_1     // the glitches
sconst_0    //
sconst_1    //
invokevirtual    0x0 0xc // glitch 1
sxor     // pattern to observe
sxor     // pattern to observe
sxor     // pattern to observe
sxor     // pattern to observe
invokevirtual    0x0 0xc // glitch 2
...
```

6 Applications of the physical characterization methods

Based on the physical characterization methods explained above we describe in this section two possible applications:

- the matching attack that allows to reverse-engineer an official applet;
- the physical attack that allows to quickly work out the feasibility of an attack against an official applet.

Note that even though an attacker has access to blank smart cards of the same model as that of the targeted open multiapplication card and she develops and loads her own OS on it, the physical characterization methods would be useless to compare the two implementations since the OS would be different and so would be the physical signatures of all the operations.

6.1 Matching attack

This attack consists in matching the signal identified for the pattern in the malicious applet with the signal of an official applet to reverse-engineer it. The first step required to set up this attack is to build a dictionary matching pattern signals with bytecode instructions. The second step is to identify patterns of the dictionary in the execution trace of the official applet to discover the operations effectively performed. It is a difficult task but there are research projects in progress to automatically recognize patterns in a signal [37]. If this attack succeeds, it jeopardizes the confidentiality of the code of the official applet (*i.e.* the code is known) but it does not imply that an attack will be possible against the execution of this official applet. Nevertheless in the Common Criteria[8] [38] evaluation method, the code of an official applet is often an asset to protect and such an attack is then considered a problem. Moreover, the knowledge of the code enables the use of analysis tools to find breaches of security. Finally, it is also possible to combine the knowledge of the code with the attacks described below.

6.2 Physical attacks

Thanks to the physical characterization methods presented above it is possible to quickly evaluate the feasibility of an attack against an official applet. Using the mixed hardware and software attacks presented in section 4.3, a hacker is in the best experimental position to easily and quickly attack a card implementation (e.g. to try to bypass the access conditions or perturb a cryptographic algorithm) without needing to perform many useless tests, thus saving a lot of time. If her attacks succeed then she can attack an official applet or the platform. For instance if she knows a way to attack a cryptographic algorithm that the official applet uses, she can try to set up the attack against this algorithm by calling it from her own applet and enclosing it by glitches to quickly test if its implementation is secure. It is also possible to add a glitch just before accessing the data of the targeted applet so as to synchronize a physical attack to try to bypass the checks of the firewall thanks to the disturbance of the hardware component.

7 Experiments

A few years ago, Sebastien Garcia[9] carried out experiments based on identical techniques to achieve matching attacks with the power and electromagnetic emanations for the ITSEF of SERMA Technologies. He worked at the assembly language level and he obtained interesting results but with the product that he used it was very hard to get a good dictionary because for some instructions it was even difficult to differentiate the signals. We are in a different situation. A bytecode is in fact a sequence of assembly language instructions that represents a bigger pattern than a single assembler instruction, therefore we believe that it may be possible to effectively build such a dictionary of Java Card bytecodes. Furthermore, since each bytecode corresponds to a really different sequence of assembler instructions, it is most likely that their signature will also be significantly different.

We have carried out some experiments on Java Cards at the ITSEF of SERMA Technologies using the physical characterization methods described in this paper in order to develop a methodology to reverse engineer official applications. We succeeded to build our characterization tests with the JCatools software environment [33, 34] that we have developed and we have encountered no problem to load them on all the Java Cards (since the sequence of operations is legal from the bytecode verification point of view). Nevertheless these tests were performed unsuccessfully on Java Cards of the GemXpresso Pro family already certified at level EAL5+ of the Common Criteria. Based on our expertise, we now are sure this was not a good idea to begin with these products since we were only given two full days to make our experiments. Moreover we wanted to work on the most interesting – but also the most difficult – tests: the physical identification of bytecodes (we had already successfully tested the attacks described in section 6.2 during real evaluations of other products). We focused our efforts on the electromagnetic emanations (since this domain was the most interesting for SERMA Technologies and since most of the

[8] The Common Criteria for Information Technology Security Evaluation, abbreviated "CC", defines a language for defining and evaluating information technology security systems and products. The framework provided by the CC allows a manufacturer to define a set of security functional and assurance requirements for his product. The CC also provides evaluation laboratories with procedures for evaluating the products or systems against the specified requirements.

[9] IXL laboratory of the University Bordeaux 1.

time the integrated circuit chips of the Java Cards already have very good countermeasures against the power analysis approach) but we have failed to establish a simple dictionary. This is mainly due to the many expensive techniques required by the recent chips to really perform a systematic and complete search (removing the shield against the electromagnetic emanations, etc.). Moreover in our case the measurement of the electromagnetic emanations is specific for a given sample of a specific model of an integrated circuit chip because of the precision required for the antennae. However with a better equipment it may be possible to have the same measurements for all the samples of a chip model. So it should also be noted that if someone succeeded to establish such a dictionary it should be significant only for an individual chip type (perhaps a family of chips that integrate the same technology).

Some attacks presented in this paper or some improved versions are used by the ITSEF of SERMA Technologies to test and attack Java Card platforms, applets and their services for instance within the framework of a Common Criteria evaluation. We succeeded to load applications using a code similar to the one shown listing 1 and to execute them on real "old" Java Cards. Besides, we often used the physical characterization methods to be in the best experimental positions in order to perform attacks against cryptographic algorithm implementations (*see* Section 6.2). Most of the time these attacks are used with the theoretical assumption that the attacker could load her code (cards evaluated are not yet open). Obviously this assumption is taken into account in the quotation of the attacks to know if they exceed the desired evaluation assurance level.

8 Related work

Some similar projects on multiapplication smart cards [39, 40, 41] have been carried out in 1999 by the Gemplus teams. They addressed many issues related to code loading, the virtual machine, object sharing, the information flow between the applets, but they did not identify the attacks based on the physical characterization methods that we have described in section 6. More recent work focused on different topics: the project presented in [42] enables a smart card issuer to verify that a new applet securely interacts with already loaded applets; in [43] a generic security model for operating systems of multiapplication smart cards is presented, that formalizes the main security aspects of secrecy, integrity, secure communication between applications and secure loading of new applications.

Our approach is thus original: we have not found any related work based on the physical methods that we propose to use to characterize a platform. Perhaps they have already been used for power analysis or electromagnetic analysis of closed smart cards. Nevertheless such an approach has never been presented as a killer method to characterize any product because if it were used it could only be in a restricted context (*i.e.* with many assumptions: e.g. "code loading is forbidden on this closed smart card but using it anyway saves a lot of time for the identification of the interesting patterns; if this method were not used it would still be possible to achieve the same goal – although this is not obvious"). The contribution of this paper is to clearly introduce these methods to characterize open multiapplication cards. In this context, these methods really threaten the security because it is not possible to prevent code that uses them from being loaded since this is a valid operation.

9 Conclusion and future work

This paper describes what open multiapplication smart cards are and it surveys the problems that they raise. The majority of the problems and attacks presented in this paper (*i.e.* sections 5 and 6) were unpublished till now and we hope that sharing our experience with others interested in the area will help to secure the future open smart cards. Even if the problems raised and the countermeasures proposed may sometimes seem obvious, we have already used them successfully during a real product evaluation to quickly set up attacks. It should be noted that people at the Smart Card Centre of the Royal Holloway (University of London) seem to begin working in the same direction [44]. In the near future, at the XLIM[10] laboratory of the University of Limoges we will continue our experiments so as to get a bytecode dictionary.

[10]The laboratory will be created in 2006 and will gather in particular people of physics specialized in the antennas – old IRCOM laboratory –, people of mathematics specialized in the cryptographic – old LACO laboratory – and people of computer science specialized in the information security – old LMSI laboratory.

References

[1] International Organization for Standardization: Information technology – Identification cards – Integrated circuit(s) cards with contacts – Part 3: Electronic signals and transmission protocols. ISO (1997)

[2] International Organization for Standardization: Identification cards – Integrated circuit cards – Part 4: Organization, security and commands for interchange. ISO (2005)

[3] Muir, J.A.: Techniques of Side Channel Cryptanalysis. Master's thesis, University of Waterloo, Ontario, Canada (2001) Master of Mathematics in Combinatorics and Optimization.

[4] Kocher, P.: Timing attacks on implementations of diffie-hellman, rsa, dss, and other systems. In: Proceedings of the 16th Annual International Cryptology Conference on Advances in Cryptology, Springer-Verlag (1996) 104–113

[5] Kocher, P., Jaffe, J., Jun, B.: Differential Power Analysis. In: Proceedings of the 19th Annual International Cryptology Conference on Advances in Cryptology, Springer-Verlag (1999) 388–397

[6] Coron, J.S., Kocher, P., Naccache, D.: Statistics and Secret Leakage. In: Proceedings of Financial Cryptography (FC2000). Volume 1962 of Lecture Notes in Computer Science., Springer-Verlag (2001) 157–173

[7] Quisquater, J.J., Samyde, D.: ElectroMagnetic Analysis (EMA): Measures and Countermeasures for Smart Cards. In: Proceedings of E-smart 2001. Volume 2140 of Lecture Notes in Computer Science., Springer-Verlag (2001) 200–210

[8] Gandol, K., Mourtel, C., Olivier, F.: ElectroMagnetic Analysis: Concrete Results. In: Proceedings of CHES'2001. Volume 2162 of Lecture Notes in Computer Science., Springer-Verlag (2001) 251–261

[9] Kömmerling, O., Kuhn, M.G.: Design Principles for Tamper-Resistant Smartcard Processors. In: Proceedings of the USENIX Workshop on Smartcard Technology (Smartcard '99), Chicago, Illinois, USA (1999) 9–20

[10] Bar-El, H., Choukri, H., Naccache, D., Tunstall, M., Whelan, C.: The Sorcerer's Apprentice Guide to Fault Attacks. In: Proceedings of Workshop on Fault Detection and Tolerance in Cryptography, Italy (2004)

[11] Giraud, C., Thiebeauld, H.: A survey on fault attacks. In: Proceedings of CARDIS'04, Smart Card Research and Advanced Applications VI, Toulouse, France, Kluwer academic publisher (2004) 159–176

[12] Skorobogatov, S., Anderson, R.: Optical Fault Induction Attacks. In: Proceedings of Workshop on Cryptographic Hardware and Embedded Systmes (CHES 2002), San Francisco Bay (Redwood City), USA (2002)

[13] Chen, Z.: Java Card™ Technology for Smart Cards: Architecture and Programmer's Guide. The Java™ Series. Addison-Wesley (2000)

[14] Sun microsystems: Java Card™ 2.2.1 Specifications. Sun microsystems (2003)

[15] MULTOS: The MULTOS™ Specification. (`http://www.multos.com/`)

[16] Hive-Minded: Smartcard.NET. (`http://www.hiveminded.com/`)

[17] SmartCards Trends: .NET Brings Web Services to Smart cards. SmartCards Trends **1** (2004) 12

[18] ZeitControl: BasicCard. (`http://www.basiccard.com/`)

[19] GlobalPlatform: GlobalPlatform. (`http://www.globalplatform.org/`)

[20] Rose, E., Rose, K.: Lightweight bytecode verification. In: In Workshop on Fundamental Underpinnings of Java, OOPSLA '98 Workshop., Vancouver, Canada (1998)

[21] Casset, L., Burdy, L., Requet, A.: Formal Development of an embedded verifier for Java Card Byte Code. In: Proceedings of the IEEE International Conference on Dependable Systems & Networks, Washington, D.C., USA (2002)

[22] Leroy, X.: On-Card Bytecode Verification for Java Card. In: Proceedings of the International Conference on Research in Smart Cards, E-Smart 2001, Springer-Verlag (2001) 150–164

[23] Leroy, X.: Bytecode verification on Java smart cards. Software-Practice & Experience **32** (2002) 319–340

[24] Cohen, R.M.: Defensive Java Virtual Machine Version 0.5 alpha. (1997)

[25] Barthe, G., Dufay, G., Jakubiec, L., Melo de Sousa, S.: A Formal Correspondence between Offensive and Defensive JavaCard Virtual Machines. In: Proceedings of VMCAI'02. Volume 2294 of Lecture Notes in Computer Science., Venice, Italy, Springer-Verlag (2002) 32–45

[26] Deville, D., Grimaud, G.: Building an "impossible" verifier on a Java Card. In: 2nd USENIX Workshop on Industrial Experiences with Systems Software, Boston, USA (2002)

[27] Foundation for Information Policy Research: Framework for Smart Card Use in Government – Consultation Response. http://www.cl.cam.ac.uk/users/rja14/cards.html (1999)

[28] Montgomery, M., Krishna, K.: Secure Object Sharing in Java Card. In: Proceedings of the USENIX Workshop on Smartcard Technology (Smartcard '99), Chicago, Illinois, USA (1999)

[29] Witteman, M.: Java Card Security. Information Security Bulletin **8** (2003) 291–298

[30] Betarte, G., Giménez, E., Chetali, B., Loiseaux, C.: FORMAVIE: Formal Modelling and Verification of Java Card 2.1.1 Security Architecture. In: Proceedings of E-Smart 2002, Nice, France (2002) 215–229

[31] Govindavajhala, S., Appel, A.: Using Memory Errors to Attack a Virtual Machine. In: Proceedings of IEEE Symposium on Security and Privacy. (2003)

[32] Chaumette, S., Sauveron, D.: An efficient and simple way to test the security of Java Cards. In: Proceedings of 3rd International Workshop on Security In Information Systems : WOSIS 2005, (Miami, Florida, USA)

[33] Chaumette, S., Hatchondo, I., Sauveron, D.: JCAT: An environment for attack and test on Java Card. In: Proceedings of CCCT'03 and 9th ISAS'03. Volume 1., Orlando, FL, USA (2003) 270–275

[34] Hatchondo, I., Sauveron, D.: The JCatools website. (http://sourceforge.net/projects/jcatools/)

[35] Moore, S., Anderson, R., Cunningham, P., Mullins, R., Taylor, G.: Improving Smart Card Security using Self-timed Circuits. In: Proceedings of ASYNC'02. (2002) 211–218

[36] Jacques J.A. Fournier, S.M., Li, H., Mullins, R., Taylor, G.: Security Evaluation of Asynchronous Circuits. In: Proceedings of CHES'2003. Volume 2779 of Lecture Notes in Computer Science. (2003) 137–151

[37] Quisquater, J.J., Samyde, D.: Automatic Code Recognition for smart cards using a Kohonen neural network. In: Proceedings of the 5th Smart Card Research and Advanced Application Conference (CARDIS'02). (2002)

[38] CCIMB: International Common Criteria home page. (http://www.commoncriteriaportal.org/)

[39] Girard, P., Lanet, J.L.: Java Card or How to Cope with the New Security Issues Raised by Open Cards? In: Proceedings of Gemplus Developer Conference, Paris, France (1999)

[40] Girard, P., Lanet, J.L.: New Security Issues raised by Open Cards. In: Information Security Technical Report. Volume 4. (1999) 19–27

[41] Girard, P.: Which security policy for multiapplication smart cards? In: Proceedings of the USENIX Workshop on Smartcard Technology (Smartcard '99), Chicago, Illinois, USA (1999) 21–28

[42] Bieber, P., Cazin, J., Girard, P., Lanet, J.L., Wiels, V., Zanon, G.: Checking Secure Interactions of Smart Card: Applets Extended Version. Computer Security **10** (2002) 369–398 Special issue on ESORICS 2000.

[43] Schellhorn, G., Reif, W., Schairer, A., Karger, P., Austel, V., Toll, D.: Verification of a Formal Security Model for Multiapplicative Smart Cards. In: Proceedings of ESORICS 2000. Volume 1895 of Lecture Notes in Computer Science. (2000) 17–36

[44] Smart Card Centre of the Royal Holloway (University of London). (http://www.scc.rhul.ac.uk/projects.php)