The use of a power analysis for influencing PIN verification on cryptographic smart card

Bachelor thesis

Lukáš Folkman

Brno, spring 2007
Declaration

Hereby I declare, that this paper is my original authorial work, which I have worked out by my own. All sources, references and literature used or excerpted during elaboration of this work are properly cited and listed in complete reference to the due source.

Advisor: Mgr. Petr Švenda
Acknowledgement

I would like to thank Mgr. Petr Švenda, the advisor of my bachelor thesis, for his comments, suggestions and time he spent helping me with this work.
Abstract

This work deals with smart cards, especially PIN verification and problems related to it. The term of a smart card is clarified and components, interfaces and operating systems are introduced. The Java Card operating system is talked into the details. The work describes security threats of timing and power analysis and reveals these terms. The main part includes a description of a cutting off power supply attack using a power analysis and the ways to secure implementation against it. Finally, reverse engineering of a code and a power analysis of a PIN verification process on the GXPLite-Generic smart card is depicted and an attack which might gain access to a PIN to an attacker is proposed.
Keywords

smart card, PIN verification, power analysis, Java Card
# Contents

1 **Smart cards** .................................................. 2  
   1.1 *Types of smart cards* .................................................. 2  
       1.1.1 Division according to components .................................. 3  
       1.1.2 Division according to interface ....................................... 3  
       1.1.3 Division according to operating systems ............................... 3  
   1.2 *Communication protocol* ................................................ 4  
       1.2.1 Application data unit .................................................... 4  
   1.3 *Related standards* ...................................................... 4  
       1.3.1 GlobalPlatform ............................................................ 5  
2 **Java Card** ...................................................... 7  
   2.1 *The split architecture* .................................................... 8  
   2.2 *Security of Java Card* ..................................................... 8  
       2.2.1 Java Card version of Java language .................................... 9  
3 **Timing and power analysis** ............................................ 11  
4 **PIN verification** .................................................. 13  
   4.1 *Three-layer authentication* ................................................ 13  
   4.2 *Implementation of PIN verification* ....................................... 14  
   4.3 *Cutting off power supply attack* ......................................... 14  
       4.3.1 Power break by an accident ............................................. 15  
   4.4 *Encryption and hashing of a PIN value* ................................... 16  
       4.4.1 Encryption of a PIN ...................................................... 16  
       4.4.2 Hashing of a PIN ........................................................ 16  
   4.5 *Further approaches to verification algorithm* ............................. 17  
       4.5.1 Array comparison implementation .................................. 17  
       4.5.2 Hardware security enhancements ................................. 19  
   4.6 *Security evaluation problems* ............................................. 19  
   4.7 *PIN verification on multithreading smart cards* ......................... 19  
5 **Power analysis** .................................................. 22  
   5.1 *Reverse engineering of PIN verification* .................................. 22  
   5.2 *Valid and invalid PINs* .................................................. 23  
   5.3 *PIN Comparison* ......................................................... 23  
   5.4 *Attacking PIN verification* ............................................... 24  
   5.5 *Operations behind writing to memory* .................................... 26  
   5.6 *Overview of PIN verification power traces* ................................ 28  
6 **Conclusions** ..................................................... 31  
Bibliography ........................................................ 32
1 Introduction

Credit cards were first issued in 1950’s, a plastic card was chosen to carry identification information. Then, a magnetic stripe was used in 1960’s for the first time. The idea of bringing a microchip inside a plastic card was patented by Helmut Gröttrup and his colleague Jürgen Dethloff in Germany. In 1974, a patent of smart cards (microprocessor integrated circuit cards) was published by Roland Moreno in France. Smart cards are spreading into various kinds of deployment nowadays and this trend does not seem to be slowing down. One does not even realize how often they use these cards as for example identification cards, debit and credit cards, transit tickets, telephone cards or SIM cards in their cell-phones.

Smart cards offer much more than their predecessors. Smart cards are able to store bigger amount of data, are capable of processing data. Security enhancements can be implemented on a card so sensitive information can be stored and processed inside the plastic case. On the other hand this opens door for new kinds of attacks dedicated to alter or gain access to private and valuable data.

The very first chapter is dedicated to smart cards in generals. It describes the basics and their use and divides them into essential groups according to their components, interface and operating system. Standards related to the topic are mentioned and depicted as well.

The second chapter, called Java Card, goes to the details of this multi-application operating system. It is capable of hosting multiple programs, called applets, which can be written by the third party distributors. The main part of the operating system is Java Card Virtual Machine, which executes applets written in Java Card version of the Java programming language.

Timing and power analysis as the way to reverse engineer a program’s code as well as gain access to sensitive data that are being used while the code execution are described in basics in the third chapter.

PIN verification on the smart cards is the topic of the forth chapter. PIN verification is a process of authenticating a card holder which comes after card and terminal authentication. Power analysis might be used to attack some implementation of PIN verification so countermeasures are suggested.

Power analysis of the GXPLite-Generic card implementation of PIN verification is described in the last, fifth, chapter. It is used to reverse engineer the code of the verification process and propose the attack that might gain access to a PIN value using a cutting off power supply attack. This attack benefits from the fact an attacker may cut off the power supply to the card before information about entering an incorrect PIN is written to a permanent memory.
Chapter 1

Smart cards

A smart card, chip card, or integrated circuit(s) card (ICC) is defined as any pocket-sized card with embedded integrated circuits [14]. There is wide range use of such cards - the most common ones are as credit and debit cards, GSM cards for cellular phones, pre-paid television, phone cards or identification cards. And still they are found to have a new field of application - medical record cards, digital passports or driving licenses.

Smart cards are convenient and important part of security mechanisms used widely nowadays. E-commerce services need to be secured by means of authentication, data integrity, confidentiality and digital signatures. Smart cards are devices capable of storing securely cryptographic keys that are needed by these mechanisms.

The older brother of a smart card is a magnetic stripe card. They are capable of storing information only. The stripe that is running across a card is divided into blocks that are magnetized in particular fashion to represent either 0 bit or 1 bit. This means it is easy to read the data stored on such a card and there are not many problems to alter it as well if suitable equipment is available. That is why for example a PIN (personal identification number) cannot be held on a magnetic stripe card and its verification must be processed online via a network and several problems come with this (encrypting the value, key management, securing the network). Although effortless reading and altering data by unauthorized person is a serious problem, magnetic stripe cards are widely used all over the world. It might be because of investments into technologies for their use and a price of the cards themselves. However, there are much smarter cards at the market nowadays.

1.1 Types of smart cards

When talking about smart cards, we usually refer to these under the standard ISO 7816. Smart cards can be classified in three main categories according to their:

- components
- interface
- operating system
1.1. TYPES OF SMART CARDS

1.1.1 Division according to components

If a card contains as its main part only a non-volatile memory chip, they are said to be memory cards, whereas microprocessor cards contain a memory chip and microprocessor components. The term smart card usually refers to a microprocessor card with various tamper-resistant properties which is able to provide security services. This is a very important feature because all of the processing containing card’s holder secret keys or authentication information (as for example a PIN is) is done on a smart card itself and an owner is not disclosing this sensitive data outside - to environment that is not trusted. On the other hand when there is no microprocessor unit, security logic controls access to a memory and enables to read from and to write to it - this is done by a card reader. It is possible to read from such a card only after providing secret key by a card reader or a card holder themselves.

Memory cards contain an EEPROM (electronically erasable read-only memory) for all of application data. This is usually of size 2 kB to 8 kB. Secondly there is a ROM (read-only memory) that contains such information as a card number, a card’s holder name - the information that does not change through whole card’s life-cycle. Microprocessor cards incorporate a ROM, which holds an operating system; an EEPROM with application programs and application data; a RAM (random access memory) for running particular functions of programs. ROM is usually of size 3 kB to 32 kB, for EEPROM, it is 2 kB to 32 kB. Different situation is with RAM size that is around 256 bytes. Finally, there is an 8-bit CPU (central processing unit) at around 5 MHz clock speed. After reading these numbers, it should be clear that smart cards offer very limited computational power in comprehension to personal computers.

1.1.2 Division according to interface

Depending on an interface, smart cards are contact, contactless or hybrid. In the first case physical contact between a card and a reader must be established. There are six to eight gold-plated contacts on the top of a card, each with particular meaning. A contactless card might be considered more user-friendly as it is enough to pass a card near an antenna to carry out a transaction. However, this is a disadvantage as well. Communication between the two of them might be eavesdropped. Hybrid cards are kind of the most sophisticated solution - easy to use for not much sensitive transactions and secure enough for the other one at the same time. With this comes higher price as for implementation difficulties. When talking about price of smart cards - memory cards can be obtained starting at about 1 $, chip cards starting at 3 $ and prices differ up to 30 $ depending on (mostly security) features they are able to provide.

1.1.3 Division according to operating systems

Nowadays there are many operating systems available for smart cards as Java Card, Multos or Basic Card. Java Card operating system is an open specification allowing third-party applications to be uploaded and is dedicated for applications written in Java. Java Card will be
discussed later in the text to more details. Multos is multi-application smart card operating system as well as the other ones mentioned. Source codes of applications are usually written in C language and then have to be converted to Multos executable language. Microsoft used to prepare its Windows for Smart Cards but original project was cancelled later then.

### 1.2 Communication protocol

As well as physical characteristics of an integrated circuit, sizes, locations and functions of card contacts, a communication protocol and a content of messages, commands and responses are defined in the standard ISO 7816. A basic logical communication datagram is called APDU, which means Application Protocol Data Unit. It allows carrying up to 260 bytes. An APDU contains either a command or a response message. A smart card always waits for a command APDU from a card reader firstly and then executes the action specified in the APDU and replies back with a response APDU.

#### 1.2.1 Application data unit

A header of a command APDU consists of four fields: class (CLA), instruction (INS) and two fields for parameters (P1 and P2). Each field is size of 1 byte. A Class byte usually identifies an application; an instruction byte is used to identify an instruction code. Following a mandatory header, there is a conditional body consisting of a number of bytes in a data field of a command APDU, a data field itself and a number of bytes as the maximum number of bytes expected in a data field of an awaiting response APDU. These numbers are usually referred to as Lc and Le respectively.

A response APDU is composed from a conditional body and a mandatory trailer. The body involves a data field only. The trailer is divided into two parts SW1 and SW2 that denote the processing status of a command APDU.

<table>
<thead>
<tr>
<th>Command APDU</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Header (required)</strong></td>
</tr>
<tr>
<td>CLA</td>
</tr>
</tbody>
</table>

![Figure 1.1: The structure of an application data unit](image)

### 1.3 Related standards

The ISO 7816 standard has already been mentioned. The EMV is another smart card standard based on their interface defined as in the ISO 7816. The EMV stands for Europay, MasterCard and Visa as these corporations are standard’s originators. The EMV is concerned
1.3. RELATED STANDARDS

on an interaction between a smart card and a POS terminal (a point of sale terminal). It defines needs of a secure and authenticated communication. The standard is used widely all over the world for credit and debit card payments as its founders are leading membership organizations for issuers of credit and debit cards.

1.3.1 GlobalPlatform

GlobalPlatform is a publicly available specification for definition and management of smart cards [3]. It defines not only cards themselves but also reading devices and their infrastructure. The ISO 7816-13 Commands for application management in multi-application environment is supposed to integrate the principles from GlobalPlatform as well.

The GlobalPlatform card specification defines an architecture where user’s application together with the GlobalPlatform Card Manager stands on a runtime environment. The Runtime environment consists of an operating system on which a virtual machine with its own API (application programming interface) as well as with the GP Card Manager API is running. The virtual machine API provides native functions that are used by user’s applications, whereas the GP Card Manager API is dedicated to the GP Card Manager. The Card Manager serves as a supervisor over applications and communication on a card.

![GlobalPlatform card architecture](image)

It functions as a command dispatch as more applications can be nested on a card and commands coming from a terminal must be sent to the selected application. It manages a content of a card and gives permission for a specific operation as for example uploading new programs. The Card Manager provides several security functions, both of the participants, card and reader, should be mutually authenticated and a secured encrypted channel should be created for a communication. There is assigned a security domain for each application so an application cannot manipulate with stored data but with the one that it owns.

Finally, besides the Card Manager, there are user applications on a card which perform functionality of a card. There might be many of them (limited by the EEPROM) but only one is selected (running) at the moment.

GlobalPlatform also defines a life cycle of a card and applications. The states of a life cycle are op_ready, initialized, secured, card_locked and terminated. The first two states
are the ones before the time a card leaves its issuer. A card is initialized after uploading differentiation data and specific applications. The state of using of a card is the one called secured as any manipulation is allowed only by a security domain of a card manager. A card might be locked at some time because of a security policy. It can be unlocked as well and go back to the state secured. A card is terminated if a serious security threat occurs or a card just expired. A life cycle of applications is more complex and it differs from an application to an application. But still, all of them involve basic states as loaded, installed, selectable or locked. A loaded application becomes a separate unit just after installing. Once it is installed it can be selected by a card manager. A selected application is the one running and communicating with a card reader.
Chapter 2

Java Card

Java, an object oriented programming language, is popular and widely used by developers nowadays. In 1996, the first specification of the Java Card was announced. The Java Card technology enables to run Java programs on smart cards and other devices with limited computational power. It is a runtime environment that consists from the Java Card Virtual Machine, core classes and supporting files – the Java Card API (application program interface) [4]. The Java Card Virtual Machine runs on a native operating system that is burned on a card by its issuer. The Java Card technology provides an environment for multiple programs to be hosted on a single card. By this mean, it makes a card multifunctional. In this context, programs are referred to as applets (similar to naming of Java Internet applications).

![Java Card architecture](image)

At first glance, Java as a language with an interpreted byte-code might seem too slow to be considered for implementing of applications for devices with limited computational power as smart cards are. On the other hand one of the greatest pros is Java’s portability and security as well. Loads of code, which has already been written by Java’s programmers, might be used for smart cards in altered way. Applets can be uploaded on any card that is
2.1. THE SPLIT ARCHITECTURE

Java Card and it does not matter on native operating system, which may vary from issuer to issuer, important is that native system runs the Java Card Virtual Machine and the runtime environment. Java Card is an open specification and so its API is. By this reason, it can be evaluated by many different experts all over the world what enhances security which is the aim of smart cards themselves. As an object orientated language, Java provides high level of abstraction as well.

2.1 The split architecture

The Java Card Virtual Machine stands on native system of a card. It controls communication among applets and card’s operating system – memory management, input and output, etc. The virtual machine interprets a byte code to a smart card’s native system and controls a flow. The Java Card technology introduces split virtual machine architecture. Applets are written on workstations, although they are run on smart cards. There is an off-card and an on-card part of the virtual machine. The off-card part works as a converter and a verifier. The converter produces a capable file format of an applet (or any Java package) to be loaded on a smart card – on an on-card virtual machine. This format is called CAP file, which means Converted Applet. The converter is supposed to take all the work for linking classes and resolving references. This reduces the load of a work of the on-card virtual machine as it is run on a card. The verifier verifies if CAP to be uploaded is under the Java Card specification. After moving CAP file on a card, it has to be installed. The installing means registering an applet to the on-card virtual machine’s registry table. Finally, the on-card virtual machine executes a code of an applet.

2.2 Security of Java Card

As it was mentioned, Java Card supports multiple applets to be run on a card. At the moment, there is only one of them selected. A card terminal sends a selecting APDU to a card – to a virtual machine and it selects a required applet. With a multifunctional card comes security threat as applets may touch data of other applets. Java Card deals with this problem by introducing an applet firewall. An applet firewall is said to stand between different applets hosted on a single card. It gives applet’s own partition of a memory and controls the use of objects that are stored on a card. Java Card enables sharing of objects of different applets so an applet can invoke methods of the other applets if they share an object encapsulating the method with it. This brings a security problem. It may not be realized that such a situation can happen as the following one. An applet $A$ shares its object $a$ with an applet $B$ which shares an object $b$ with an applet $C$. The object $a$ and $b$ involve methods $\text{methodA}$ and $\text{methodB}$ respectively. $\text{MethodB}$ invokes $\text{methodA}$ in its body. Now, if the applet $C$ invokes $\text{methodB}$, it does invoke $\text{methodA}$ indirectly as well, even though the applet $A$ does not share the object $a$ with it explicitly [7]. This is situation is depicted on the figure 2.2.

The applet firewall is not the only one of security features that the Java Card runtime environment brings. Further, it allows signing applets by their issuer and then verifying
2.2. SECURITY OF JAVA CARD

whether they have not been altered before uploading. The Java Card API incorporates a
wide range of classes for the support of security services as symmetric encryption and de-
cryption algorithms, key interfaces, a signature generation and verification, message digests,
a random data generation and a PIN management. Java Card as an open specification has
an advantage that various security experts can analyze and report its implementation of
security algorithms. Another enhancement is property of transaction atomicity. This means
that either a marked part of code is executed as whole and all of the changes are written
into persistent memory or either none of the changes are saved in case the code has not been
finished until the end of the marked selection (e.g. a card is removed from a card reader, a
power supply is cut off).

2.2.1 Java Card version of Java language

Some of security properties were just inherited by Java Card from the Java programming
language as the fact that it is a strongly typed language – it does not endorse pointers or
it provides a name space management of members of classes to control an access to them
(a member can be set as public, private, protected or package-private). On the other hand
there are several restrictions to the Java language used for Java Card. Some of them even
strengthen security of developed applications and some of them do not. Restrictions are
needed because of limited power of smart cards. The Java Card specification does not sup-
port:

- dynamic class loading
- security manager
- threads and synchronization
- garbage collection and object cloning
- finalization
- large primitive data types (float, double, long and char)
- some classes (most of the java.lang, Object and Throwable in limited form)
2.2. SECURITY OF JAVA CARD

By removing a dynamic class loading it is easier to enforce type safety. Although the security manager is not supported, security policies are included in the virtual machine itself. Lack of threads in Java Card makes it easier to ensure that applets do not collaborate or threaten each other while running simultaneously.

On the other side garbage collection does not have to be implemented on a card. This might be considered as a problem since logic errors in the code of an applet may make a memory of a card full over a time. Further, there might be parts of a memory freed for a new use and still there might be references to the objects that used to be stored at the freed piece of a memory, one may touch the data that does not belong to them using these references.

The Java Card technology tries to build its way on the success of the Java programming language and it adds smart card related features to it.
Chapter 3

Timing and power analysis

Paul Kocher published a paper about a timing analysis in 1996 [6]. A basic idea of the timing analysis is to obtain additional information about card’s internal state from measurements on a time consumption of an execution, because the time consumption depends a lot on input data. The main assumption about a defense against a timing analysis attack is that the length of the execution of an algorithm must be independent on input data. This goes against an optimization of code during a development. Each execution of a function in a program on a smart card must take approximately the same time, in other words, each execution will take the time of the most time-consuming execution possible (depending on processed inputs). Other acceptable defense might be bringing randomness into a computation.

The exponentiation in a naive implementation of the RSA algorithm is a good example of the threat of a timing analysis. $M^d$ is the value needed to be calculated and $d$ is a secret key. The exponentiation algorithm depends on bits of an exponent as it is based on the binary method. Only when the value of a bit is 1, multiplication inside a loop is performed. It means that when there is 0 bit, the loop is finished faster. This implies that Hamming weight of the key can be discovered.

Two years after publishing a timing analysis paper, power analysis was described for the first time by Paul Kocher [5]. It is based on a similar principle. A card’s internal state is estimated from the knowledge of a power consumption of the card. This is able to be measured because a card has no internal power supply and depends on an external source of energy. Simple power analysis (SPA) uses a directly measured power trace to guess the nature of operations and data involved. Differential power analysis (DPA) takes into charge statistical analysis techniques to remove a noise from a signal and to discover less obvious dependency of power consumption on processed data.

The simple power analysis might be good for revealing secret data processed by a smart card, reverse-engineering of a smart card code (estimation of the beginning and the end of a PIN verification procedure, etc.). SPA is often used only to reduce brute-force searched field of possible secret keys. Attacks are based on simple knowledge that for example xor operation consumes more energy the higher hamming distance of operands is. That is because more transitions of transistors from 0 to 1 and vice versa are required.

When implementing any kind of a countermeasure against timing and power analysis attacks, computationally limited power of smart cards must be taken into account. A countermeasure needs to be surely effective but cannot decrease a performance of an execution markedly at the same time. Known countermeasures are of hardware (additional circuit as
a noise generator that consumes a random amount of energy) and software (randomization of the time needed for a computation or masking techniques – applying xor operation on processed data and some random value) kind as well.
Chapter 4

PIN verification

4.1 Three-layer authentication

Let’s have a look at what happens once a card holder obtains their card and wants to use it. A card is plugged into a card reader. Authentication of the card follows. A simple way of authenticating the card might be like this: the card reader sends a random number to the card with request for an authentication. The card encrypts the number using a symmetric key cryptography with the key it shares with the terminal and sends it to the card reader which verifies if obtained value is really the encrypted random number that was sent. This process is called an internal authentication. Afterwards, an authentication of the card reader against the smart card may follow. It is a similar process. As it was told before, a card itself never initializes a communication, it always only responds to requests from a terminal. It means that the card reader sends a request for a random number to be received from the card and then rest of protocol continues as described before.

There might be a secret password chosen by a card owner stored in a secured part of memory on a card that is used for a card holder to be sure that the authentication of a terminal was performed as well. The card doesn’t let the terminal read that memory part until the card reader has authenticated itself. Once the terminal is authenticated as it was described in the paragraph above, it reads the value of the secret password and displays it to the card holder on a display for them to check if it is the right password. This is performed for the user to have some kind of proof that the authentication did go through.

Need for a terminal authentication comes from the existence of dummy terminals. These would just collect PIN values entered and do nothing else. A person deploying such a terminal would then try to steal a card from the user who used the terminal and use the collected PIN to misuse it.

Once both of participants are authenticated a secured channel can be built for their communication so an attacker cannot be overhearing a card’s communication with a card reader. The third part to be verified is a card holder. A PIN verification process follows. The user is asked to enter the PIN on a key pad of a card reader and it sends the value to the card (that is why the secured channel must have been built, because secret sensitive data are about to be sent). PIN is usually a number of length 4 to 8 digits. Only the card owner is supposed to know the value of the PIN. It is transported securely to the hands of the card owner from a card issuer. Either the issuer is not supposed to know actual PIN.
4.2 Implementation of PIN verification

Finding out a PIN value by guessing should not be possible because user has a limited number of tries to enter an invalid PIN. A tries-remaining counter is involved in an algorithm used for verifying a PIN. It is set to a specific number (most often 3) which determines how many times an incorrect PIN can be input. After reaching this limit the card is blocked. If one needs to unblock their card, there is usually PUK for this purpose. PUK stands for Personal Unblocking Key. This key is usually longer than PIN and after inputting it, card is ready to be used again and the tries-remaining counter is set to its maximum value.

A naive implementation of a PIN verification method might seem as the one in the example 4.2.1:

```java
public boolean verifyPIN(byte[] pin, short length){
    if(this.triesRemainingCounter > 0 && length == this.pinLength){
        if(compareArrays(pin, this.cardPin, length) == 0){
            this.triesRemainingCounter = this.triesLimit;
            return true;
        }
    }
    this.triesRemainingCounter--;
    return false;
}
```

Example 4.2.1: Naive implementation of PIN verification

Method takes a PIN to be verified as its argument. The length of an input PIN has to equal the length of a PIN stored on a card (to which it is referred as `this.cardPin`) and value of tries that remains for user has to be greater than zero. If PINs are equal, true is returned. Otherwise, tries-remaining counter is decreased and false value is returned.

4.3 Cutting off power supply attack

There is a possible attack on this implementation of PIN verification [9]. It is assumed that an attacker is able to measure a power consumption of a card and observe and analyze a power trace in real time. What is more, power needed by a card is not constant – a card consumes a diverse amount of power for a diverse kind of processing, but consumption is more or less similar for the same processing. Then, an attacker might be able to differentiate individual operations performed while verification process and finally to differentiate verification of a valid PIN from the other ones. In case an invalid PIN has been input, attacker cuts a power supply to a card off in such a moment it would mean a tries-remaining counter cannot be decreased. Then, such an attacker has conditions to try all possible variations of a PIN until they find valid one. Typical length of a PIN used nowadays is 4 because of a limited ability or want of humans to remember long numbers, so an attacker would have to try no more
than 10,000 variations of PINs.

The method used in this kind of an attack is called a simple power analysis (SPA). SPA is a way for reverse engineering of a code executed on a card. Using syndromes, which are some kind of templates, of power traces for a specific operation (writing into a memory, computing random values, etc.), one is able to analyze what is being performed on a card on the fly. In this case, method shown in the example 4.2.1 is programmed in an improper fashion – with an order of operations which enables an attacker to cut off a power supply in such a moment that they can benefit from not finishing the code of a running method. A countermeasure against such a type of an attack is support of atomic orders by an operation system which means that method is performed entirely or no change is done in the other case. Unfortunately, this does not help with the PIN verification. It is required to decrease the tries-remaining counter even if the operation did not happen to end properly so an attacker cannot profit from that.

The method verifyPIN can be written more sophisticated (meaning against such a type of an attack), depicted in the example 4.3.1:

```java
public boolean verifyPIN(byte[] pin, short length) {
    this.triesRemainingCounter--;
    if (this.triesRemainingCounter > 0 && length == this.pinLength) {
        if (compareArrays(pin, this.cardPin, length) == 0) {
            this.triesCounter = this.triesLimit;
            return true;
        }
    }
    return false;
}
```

Example 4.3.1: Counter-decrease-first implementation of PIN verification

In this case, the tries-remaining counter is decreased at first, before anything else is performed. Then, its value is reset to the maximum value again in case PIN is valid. An attacker cannot benefit from cutting off a power supply at any time of an execution. On the other hand if power to the card is cut off accidentally, the tries-remaining counter has already been decreased and this might be inconvenient for a card holder.

### 4.3.1 Power break by an accident

This reminds of another security problem relating to smart cards – a power break by an accident [11]. Smart cards do not have any internal source of energy. They often operate in an environment in which it is possible to take them out of a card reader while processing. If writing to a memory is being performed at such a moment, a card ends up in an undefined state or it might be even unusable. That is why using a transaction processing when handling sensitive data is important.
Another approach to the problem of the PIN verification is known and was described by Wolfgang Rankl in his paper Overview about attacks on smart cards [9]. The operation that is said to be one of the easiest to recognize from the others when analyzing power trace is writing to a memory. If an attacker tells verification of a valid PIN from invalid one by observing writing to a memory when decreasing a tries-remaining counter, it may be a satisfying countermeasure to write random data to a dummy cell in EEPROM. However, verification of an invalid PIN can be recognized in some cases even sooner during the arrays being compared. Then, the described countermeasure would be useless.

4.4 Encryption and hashing of a PIN value

Inside the PIN verification method (the example 4.3.1) that was shown before, there was a method used to compare two arrays that are filled with PINs. A naive implementation of a comparison algorithm that considers an optimization of a source code during a development would compare values consecutively byte by byte, digit by digit and if it finds out they do not match, it aborts the method immediately with a result that the PIN input is invalid. There might be a significant difference in a time needed to compare values differing in the first digit to these differing in the second digit (for the other digit etc.). An attacker would know at which digit a card’s PIN differs from the one they input. This knowledge decreases the field of PINs that has to be taken into an account by an exhaustive search brute force attack. A solution of the problem might be an algorithm that always compares the whole values of PINs. A design of the comparison method will be discussed later then.

4.4.1 Encryption of a PIN

Actually, a PIN on a smart card is not stored in a plain form because of security reasons [2]. Each card usually has its generic key. This key is often shared among plenty of cards of the same issuer. Then, a PIN is preserved in a memory encrypted with this key by some kind of symmetric key cryptography. This cryptography function is also supported by a card so an entered PIN is encrypted using the same key and encrypted values are compared. Then, even if attacker knows at which position the encrypted PINs differ, the information is useless for them.

4.4.2 Hashing of a PIN

Another issue is to use a hash function instead of an encryption. Since a PIN is usually short number (4 to 8 digits), it is dangerous to use hashing. In case an attacker gains access to a hashed value of a PIN, they do not need much time to provide an exhaustive search brute force attack to retrieve the PIN. A hashing with a salt solves the problem with the length of a PIN. In this context, the salt is a random number long enough to create secure hash from it. The salt is linked with the PIN using a xor function and a result is hashed. When verifying a PIN the same random number must be used to be linked with an entered PIN.
4.5. FURTHER APPROACHES TO VERIFICATION ALGORITHM

With this comes question where and how to store a card generic key and a salt used for the hash. There should be a secured part of a memory where these values can be stored with no access from outside. As this memory has to be highly secured it cannot be big because of an economical reason and so only most important keys are held there.

4.5 Further approaches to verification algorithm

An algorithm that assures a consistency of information stored in a tries-remaining counter and at the same moment does not let a possible attacker to manage to unplug a card from an AC power such a way that they can benefit from is hard to find. If the first condition is held, a tries-remaining counter has to be decreased after a PIN comparison as it was shown in the example 4.2.1. The second method (the example 4.3.1), which decreases the counter prior to the PIN comparison, cannot meet the requirement of a consistency although it is secure. In case of an accident power break the tries-remaining counter is decreased and this restricts a legitimate user.

4.5.1 Array comparison implementation

The third variation of the method was mentioned – the one that decreases counter after the comparison and in case of an invalid PIN some data is written into a dummy EEPROM cell. With this standpoint came the problem that an attacker might be able to recognize verification of an invalid PIN even during the value comparison. Then, a solution could be found in an appropriate design of the comparison method.

Simple implementation of compareArrays looks like the following one in the example 4.5.1:

```java
protected boolean compareArrays(byte[] value1, byte[] value2, short length){
    for(int i = 0; i < length; i++){
        if(value1[i] != value2[i]){ return false; }
    }
    return true;
}
```

Example 4.5.1: Naive implementation of comparison method

As it was told before, if PINs are compared encrypted, information at which digit they differ is useless. Nevertheless, a comparison which is aborted as soon as the first couple of digits are different is still inappropriate. Different length of comparisons might give attacker information whether a correct or incorrect PIN is being processed. That is why a implementation given in the example 4.5.2 is more suitable for smart cards:
4.5. FURTHER APPROACHES TO VERIFICATION ALGORITHM

protected boolean compareArrays(byte[] value1, byte[] value2, short length){
    boolean result = true, dummy = true;
    for(int i = 0; i < length; i++){
        if(value1[i] != value2[i]){
            result = false;
        } else{
            dummy = false;
        }
    }
    return result;
}

Example 4.5.2: Comparison method always going through all the loops

The above algorithm goes always through all the loops. Their count is given by the length of a PIN. Each couple of digits is tested if they match inside the loop. In case they do, local variable dummy is set to false. Otherwise local variable result, which is returned at the end, is set to false. The variable dummy is incorporated just to make each loop look similar in the meaning of operations provided inside.

Still, there is a question if the implementation is secure. It depends on character of an if-then-else instruction inside the loop and a capability to reverse engineer it using a power analysis. Denis Vermoen in his master thesis [13] describes distinguishing of if and else branches in a Java Card applet. It takes Java Card Virtual Machine longer when an else branch is performed. An explanation might be the time needed to compute an address at which a computation continues. A question is if this difference is also significant enough for an attacker to provide the attack successively in real time. An answer will not be clarified. Denis Vermoen created two templates for if-then-else instruction, one for the case an if branch is taken and another for the case an else branch is taken. This shows a theoretical way to attack the implementation. On the other hand he did not try a cut-off power supply attack on any smart card.

However, the implementation of compareArrays can be designed even without the if-then-else instruction. A pseudo-code in the example 4.5.3 illustrates this:

protected boolean compareArrays(byte[] value1, byte[] value2, short length){
    boolean result = true;
    for(int i = 0; i < length; i++){
        result = (value1[i] == value2[i]) && result;
    }
    return result;
}

Example 4.5.3: Comparison method without if-then-else instruction

18
Unfortunately, compareArrays always has to be dependent on the values it is comparing. Satisfying might be to hide this dependency in such a fashion that a successive attack cannot be provided.

4.5.2 Hardware security enhancements

Another way to secure a PIN verification process is out of the method implementation. A noise generator is an additional electronic circuit on a smart card that continuously consumes random amount of energy and so a measured power trace cannot be analyzed. Though, it might be expensive to be implemented because of a limited space for circuits defined in the ISO 7816 and it may be disabled by tampering. A tampering is altering the smart card by making unauthorized changes. A card might be opened and noise generator disconnected from the whole circuit. On the other hand card producers do know countermeasures against a tampering. There are put protective layers, light and passivation sensors around and on microcontrollers. If a tampering is recognized, a card is blocked. Sensitive data are erased or some other reaction is performed depending on a situation. Discovering new attacks goes hand in hand with discovering satisfying countermeasures.

4.6 Security evaluation problems

As it was said, some operating systems are open specifications. Java Card was described into more details. This means that it is practicable to write third-party applications which can be hosted by these operating systems. On the other hand it does not mean that Java Card is an open source. It is usually not possible to read source codes of the Java Card API (which is possible when talking about the API of the standard Java programming language). This phenomenon is called security by non-disclosure and such architectures cannot be fully evaluated and a space for attacks as side channel attacks is created [11].

Each distributor of a smart card operating system or an application provides their own implementation of the PIN verification and it is hard to tell anything about its security. Use of cryptography or other security achievements is not enough to consider that a smart card reaches the status of a reliable device. These security services have to meet the concept of a secure implementation as well. When an implementation is closed to the public, it is difficult to evaluate its properties and an end user can only trust a manufacturer that they provide cards with securely implemented operating systems. This is intended for distributors of smart cards applications too. There comes a context of a legitimate use of a power analysis – reverse engineering of applets, so their design can be evaluated, although, not entirely.

4.7 PIN verification on multithreading smart cards

As the technology of smart cards and their operating systems is getting further, multithreading is about to be supported by future smart cards. Then, multi-application smart cards will become multitasking and this, for sure, brings several problems besides all benefits.
4.7. PIN VERIFICATION ON MULTITHREADING SMART CARDS

When focusing on PIN verification, one has to have on their mind that there is a shared attribute for a remaining-tries counter. If this is forgotten, several unpleasant situations may occur.

Imagine PIN verification is implemented with a counter decreasing prior to a value comparison and a remaining-tries counter is set to the value 1. An applet needs to verify a card holder and executes the verification method. At the very beginning the counter is decreased to the value 0. In this moment another applet needs the card holder to be authenticated and executes the PIN verification method as well. The second applet learns that value stored in the tries-remaining counter is set to 0 and inform the operating system to block an access to the card. Obviously, this is not a suitable behavior of multitasking on smart cards.

Another situation may arise if the verification is implemented with a tries-remaining counter decreasing after a value comparing. It is the situation when the counter is again set to the value 1 and an applet executes the verification method. A card holder enters an incorrect PIN and the entered value is about to be compared with a PIN of the card. At this moment another applet executes the PIN verification and learns that the counter is set to 1, although it should be 0 already. Then the card holder is given one extra try to guess the valid PIN which is against a security policy.

This means that access to the shared attribute tries-remaining counter has to be implemented somehow sophisticated. The Peterson’s algorithm [12] which solves the problem of a mutual exclusion (mutex) can be used here (the example 4.7.1). There is so-called critical section in the algorithm which is a piece of a code that is needed to be performed by just one process at a moment (verification of the PIN).

flag[i] = true;
turn = j;
while(flag[j] && turn == j); //active loop
verifyPIN(pin, length); //critical section
flag[i] = false;

Example 4.7.1: The Peterson’s algorithm

Process i, which wants to enter the critical section, sets its own flag to true. It is not its turn yet, it is the turn of process j. Process i has to be waiting in active loop until the flag of the process j is set to false. Finally, process i enters the critical section and performs the PIN verification and sets its flag to false so the other process can enter.

In this implementation only one process is allowed to be verifying the PIN at a moment and so data stored in the remaining-tries counter is consistent and a changing of it is atomic.

However, this is an old software solution of the mutual exclusion problem which is implemented with a busy waiting. This means that process that wants to access already an occupied critical section is continuously testing the value stored in the variable flag[j] and turn. This is also called a spinning. Nevertheless, a process is consuming processor time in a vain at this moment. That is why other solutions of the mutual exclusion problem were invented. An operating system itself may make a process sleep when about to enter an occu-
4.7. PIN VERIFICATION ON MULTITHREADING SMART CARDS

pied critical section and then wake it up if the process is allowed to enter the critical section.

So called monitors are used for this reason in the standard Java programming language [1]. Threads and synchronization is not supported in Java Card, though. This is because smart cards do not have sufficient computational power nowadays. However, when there are multithreading smart cards one day, they might be more powerful as well and perhaps Java monitors might be used as means of synchronization.
Chapter 5

Power analysis

A power analysis was measured using a special measure device – the board SCSAT02 and as a smart card was used the c card. Measurements were concerned with a PIN verification process. The board is deployed between a card reader and a smart card. In this concept, an inverse smart card reader is used and it is the board which is plugged to an external power supply. The board samples the amount of power needed by the card and it sends this data via LPT port to a computer. The maximum frequency of measuring is 5MHz and the board has memory of size 512 kB. Obtained data are processed with the SCTool software on a computer. SCTool is a program which displays measured values in a visual form as power traces. There are created templates (syndromes) of power traces for specific operations for the GXPLite-Generic card. Finally, measured power traces were analyzed and their pieces were compared to templates for reverse engineering of the code executed while PIN verification to be performed.

5.1 Reverse engineering of PIN verification

Results from bachelor thesis of Jakub Ferenc Odběrová analýza průběhu ověřování PINu na kryptografické čipové kartě [2] are used in this chapter.

The first distinct part on the power trace of PIN verification was recognized as copying of some array. When realizing PIN verification is call of a function with parameters, this might be copying of parameters to local variables. The beginning of the PIN verification power trace, the array copying, is showed in the figure 5.1.

![Figure 5.1: Part of the array-copying power trace](image)

An encryption using the symmetric key cryptography algorithm DES (Data Encryption Standard) most probably follows. It can be assumed that a PIN on the GXPLite-Generic card is not stored in a plain form. It is most probably encrypted using DES and so an entered
5.2 Valid and invalid PINs

In the further text, it is assumed that the hypothesis about the sequence of the operations performed on the GXPLite-Generic card from the previous paragraphs hold true.

A very important fact is that the traces for different valid PINs seem to be the same, in general. The measurements that were taken prove this. The same thing might be said about invalid PINs after analyzing their power traces, although it gets trickier in this case.

5.3 PIN Comparison

Timing analysis of the power traces for a valid and invalid PIN differs significantly in the part between the DES encryption and the writing to a memory. It might mean that the comparison is aborted as soon as the first two digits do not match. Then, there may be such invalid PINs which power traces diverges from each other. However, there is relatively little chance for this to occur. PINs are encrypted using DES when compared. DES is a block cipher with the Feistel network structure and so it produces an indistinguishable output.
from a random sequence for someone who does not know the key. It is unlikely to happen that such an encryption function returns outputs that are identical in the first digit for two different inputs. The probability of happening so for a given input is $1/256$, as the GXPLite-Generic involves an 8-bit processor, which means it most probably compares two numbers on per byte manner.

This hypothesis might be grounded on more detailed observation of the power traces. It is supposed that the comparison fails after the first round of the loop in case of an incorrect PIN. Since DES produces an eight bytes long block, it is expected that the correct PIN comparison part of the power trace is eight times longer. The parts that are identical for both valid and invalid PINs need to be identified so it is possible to calculate times needed for the dissimilar parts. The figure 5.2 depicts the moment when the comparison starts to differ for the first time. The figure 5.3 illustrates the moment when the power traces can be synchronized again. The correct PIN power trace is red color, the incorrect PIN one is colored in blue.

The own part of the correct PIN comparison (red one) lasts from the time 377 to 1135 which gives the duration of 758. The incorrect PIN trace is unparalleled from the time 377 to 470. The duration is 93. It takes 8.15 times longer to compare a correct PIN value to a card PIN value and that is what was predicted. The hypothesis about the PIN comparison implementation most probably holds true.

![Figure 5.2: Synchronized power traces of the PIN comparison at the beginning](image)

Figure 5.2: Synchronized power traces of the PIN comparison at the beginning

![Figure 5.3: Synchronized power traces of the PIN comparison at the end; the place 1135 is the same as 470 (the figure 5.2) in case of the blue trace](image)

Figure 5.3: Synchronized power traces of the PIN comparison at the end; the place 1135 is the same as 470 (the figure 5.2) in case of the blue trace

### 5.4 Attacking PIN verification

The main aim of reverse engineering, timing and power analyses is finding a security hole. The need is to detect a place where one can always differentiate soon enough verification
of a valid and an invalid PIN. Soon enough means that the tries-remaining counter has not been decreased yet. Naturally, such a place must be dependant on the PIN value as the PIN is the only input and its value determines the course of a computation.

After the copying arguments to local variables, the encryption is the next place where PIN value is incorporated in computation. When the power traces of correct and incorrect PINs are studied in this part, they look similar but they diverge in time. Even if the traces of several correct PINs are brought under timing analysis, they do diverge. Nature of the DES implementation causes this behavior. Loops that bring a time consume randomness are involved in the execution so it is hard to reverse engineer. This implies that not much if anything can be deducted from the power traces of the PIN encryption process.

As it was assumed before, after encryption process the PIN comparison follows. A timing analysis shows that the time needed for this operation is strongly dependant on the value of a PIN since matching of PINs digits determines the number of the loops to be taken while comparing.

A possible attacker is able to obtain his own card and learn the time needed for the comparison of the valid PIN using a timing analysis. Then a significant operation that is common for both comparisons of valid and invalid PINs is required to be found. Finally, the attack per se is provided as waiting for this significant operation to emerge and if it occurs in shorter time then it is supposed to in case for the valid PIN, asmart card is unplugged.

Such a significant operation is possible to be recognized on the power traces which were collected from the measurements on the GXPLite-Generic card. It is a sharp short deep fall depicted in the figure 5.4 and 5.5. This is a suitable place to attack PIN verification.

![Figure 5.4: PIN comparison of valid PIN](image)

![Figure 5.5: PIN comparison of invalid PIN; (Sharp short deep fall comes sooner in this case.)](image)

On the other hand an attacker might look for such a place on the traces that is significantly different for valid and invalid PINs. This may be observed coming just after the significant sharp short deep fall which was described before. When examining the power
5.5. OPERATIONS BEHIND WRITING TO MEMORY

traces of valid PINs, two wide falls come one after another just after the mentioned sharp one and then another sharp one comes up (showed in the figure 5.6). The traces of invalid PINs do not involve these (the figure 5.7). However, they are not skipped. Instead of them some kind of a processing is performed. It is hard to tell what is being done on the card at that moment – meaning for both correct and incorrect PINs. According to the templates both of them should not be operations of writing to a memory, this comes just after them, which means that the remaining-tries counter is going to be decreased or reset to the maximum value.

Nevertheless, it is a suitable place to attack the card again. An attacker waits for the described sharp short fall and in case the two wide falls does not appear just after it, they unplug the card before decreasing the counter and try the whole process again with a new value of a PIN. In case two wide falls appears, an attacker gains the valid PIN of the card. In this case an attacker does not have to know the time analysis of the PIN comparison to provide the attack.

5.5 Operations behind writing to memory

Other differentiating operations are found later then. After the writing to a memory operation which is common for processing of both correct and incorrect PINs, a processing similar to the one that was marked as the comparison method in the above text can be found here. This part is longer for invalid PINs than for valid ones.

In case of an invalid PIN, operations exactly similar to these that were already described for the comparison of valid PINs before can be found in this part of the power trace and it sits between two writings to a memory. The first one has already been analyzed as decreasing the counter and its end can be seen in the figure 5.8 as the very first bump. The unknown
5.5. OPERATIONS BEHIND WRITING TO MEMORY

operation discussed in this paragraph follows afterwards. The figure 5.9 depicts how similar the unknown operation for an incorrect PIN is to the operation of the comparison of a correct PIN.

![Figure 5.8: Unknown operation found after writing to memory for invalid PIN](image)

Even with a piece of knowledge that the power trace is strongly comparable to the one that were depicted sooner as the valid PIN comparison it is hard to tell what is being computed at this moment. Accordingly to the mentioned theory, PIN has already been compared to the value on the card and the tries-remaining counter has been rewritten.

In case of a valid PIN, an operation which follows the setting of the counter (the figure 5.10) is different from all the others previously discussed. However, it is similar in some parts (the figure 5.11). The meaning is not known again. The operation is followed by the operation similar to the syndrome of writing to a memory but some differences can be found. All other discovered writings to a memory were exactly similar to the template.

![Figure 5.10: Unknown operation found after writing to memory for valid PIN](image)

Nevertheless, there were places to attack described in the previous chapter and an important thing is that these places come before the operation which meaning is unknown. It is a reason why the success of the described attacks should not be compromised.
5.6 Overview of PIN verification power traces

This is an overview about the thoughts that were put together in the whole chapter 5, called Power analysis. It summarizes the comparison of the correct and the incorrect PIN power traces. Time values that are stated in this chapter are taken from the SCTool data files which involve complete average power traces for correct and incorrect PINs verification.

PIN verification on the GXPLite-Generic begins with the copying of an array in case of both valid and invalid PINs. This is the part from 0 to 900 approximately and it is depicted in the figure 5.12.

![Figure 5.12: Array copying, traces are identical](image)

The DES encryption of a PIN value follows from around the time 1700 to 6000. The first two significant parts are known as a key initialization. The second one is desynchronized a little. It was said that a DES processing distinguishes in time as a result of a card defense mechanisms. The figures 5.13 and 5.14 illustrate traces for both valid and invalid PINs.

![Figure 5.13: DES encryption of the supplied PIN, part 1](image)

The part which is with the highest probability PIN comparison is showed in the figures 5.15 and 5.16. Power traces differ from each other the most in this part. The traces in the figure 5.15 are synchronized at the beginning of the comparison, whereas they are synchronized at the end in the figure 5.16. Valid PIN is red color, invalid one blue. The PIN comparison goes from about the time 7100 to 8500 in case of the correct PIN power trace,
5.6. OVERVIEW OF PIN VERIFICATION POWER TRACES

although the incorrect PIN trace is dedicated for the comparison from the position 6800 to 7600, which is noticeable shorter.

The writing to a memory is performed as the next. It may be setting the value of the tries_remaining counter. It is a long operation which equates for a correct and an incorrect PIN in general. The figures 5.17, 5.18 and 5.19 depict it. In case of a valid PIN writing to memory goes from the time 8500 to 31300 (as it was said it is a longtime operation). The duration for an invalid PIN is from 7600 to 30400.

The operation which meaning was not discovered in this work follows the memory writing. It is similar to the valid PIN comparison in its final part but this cannot be said about the whole operation, though. It is also the second operation which differs from each other for correct and incorrect PINs. This is depicted in the figure 5.20. Again, red one is correct PIN power trace from the time 31400 to 32100. Blue incorrect PIN trace extends from 30400 to 31200.

In case of invalid PIN (the figures 5.21 and 5.22) one more writing to a memory is performed from the position 31200 to 46800. The valid PIN power trace from the position 32100 to 45800 contains operation similar to the memory writing but it differs in some aspects.
5.6. OVERVIEW OF PIN VERIFICATION POWER TRACES

Figure 5.17: Memory writing, part 1

Figure 5.18: Memory writing, part 2

Figure 5.19: Memory writing, part 3

Figure 5.20: Unknown operation behind the first memory writing

Figure 5.21: The last operation of the PIN verification, part 1

Figure 5.22: The last operation of the PIN verification, part 2
Chapter 6

Conclusions

The basics about smart cards were discussed in this thesis. Java Card as a modern multi-application operating system was described. Security aspects of this system were mentioned shortly. The main part of the work touches PIN verification on a cryptographic smart card. Advantages and disadvantages of possible implementations of this operation were analyzed with emphasis on protection against a power analysis and possible cutting off power supply attack. A development of programs for smart cards should not forget possibility of this attack to occur. It is not an easy task to hide a code dependency on input values. Smart cards technology is in progress and there might be multithreading supported by them in a short time. This theme was touched in brief. Unknown implementation of PIN verification on the GXPLite-Generic card was analyzed practically. Most of the operations that can be observed on the power traces of verification method were described and theoretical knowledge could be substantiated by them. The power traces were measured on a measure device SCSAT02. Two, but very similar, possible ways how to attack the PIN verification method were declared. If these attacks are realized, an attacker gains unlimited count of tries to verify an incorrect PIN value.

Smart cards are smart devices which became one of the things people are used to carrying every day in recent years. They operate with sensitive and private data mostly so a stress is put on their security features and tamper resistance. Power analysis is a manner how to both evaluate and attack these properties.
Bibliography


[10] Ruuskanen, J.: Javacard, <http://citeseer.ist.psu.edu/486867.html>. 4.3.1, 4.6


