Stefan Mangard (Ed.)

LNCS 7771

Smart Card Research and Advanced Applications

11th International Conference, CARDIS 2012
Graz, Austria, November 2012
Revised Selected Papers

Springer

Stefan Mangard (Ed.)

Smart Card Research and Advanced Applications

11th International Conference, CARDIS 2012
Graz, Austria, November 28-30, 2012
Revised Selected Papers

Springer

Volume Editor

Stefan Mangard
Infineon Technologies AG
Am Campeon 1-12, 85579, Neubiberg, Germany
E-mail: stefan.mangard@infineon.com

ISSN 0302-9743 e-ISSN 1611-3349
ISBN 978-3-642-37287-2 e-ISBN 978-3-642-37288-9
DOI 10.1007/978-3-642-37288-9
Springer Heidelberg Dordrecht London New York

Library of Congress Control Number: 2013933707

CR Subject Classification (1998): E.3, C.2, K.6.5, D.4.6, C.3, D.2

LNCS Sublibrary: SL 4 – Security and Cryptology

Typesetting: Camera-ready by author, data conversion by Scientific Publishing Services, Chennai, India

Printed on acid-free paper

Springer is part of Springer Science+Business Media (www.springer.com)

Preface

This volume contains the papers that were selected for the 11th Conference on Smart Card Research and Applications (CARDIS 2012). Since 1994, CARDIS has been the foremost international conference dedicated to the security of smart cards and embedded systems. The conference was held every two years until 2010. Since then, the conference has been taking place every year owing to the fast evolution of the security research on smart cards and embedded systems. The conference brings together people from academia and industry to present and discuss new results on a wide range of security topics that include hardware architectures, operating systems, cryptography, application development as well as all kinds of physical attacks and corresponding countermeasures.

This year, CARDIS was held during November 28-30, 2012, in Graz, Austria. It was organized by the Institute for Applied Information Processing and Communications (IAIK) of Graz University of Technology. The Program Committee with its 30 members selected 18 papers out of 48 submissions for presentation at the conference and for inclusion in these proceedings. Each submission was reviewed by at least three reviewers.

In addition to the presentations of the papers, there were two invited talks at the conference. The first talk was given by N. Asokan and was entitled "Mobile Platform Security." The talk compared different approaches for platform security architectures as they are used in today's smart phones. The second talk, "Defensive Leakage Camouflage," was given by David Naccache. A paper version of this talk is also part of this volume.

Many people deserve credit for spending their time and energy to make this conference such a successful event. I would first like to thank all the members of the Program Committee and the 74 external reviewers for their hard work in evaluating and discussing the submissions. A big thank-you also goes to Jörn-Marc Schmidt and his team at IAIK, who did a great job in organizing the conference. Furthermore, I am very grateful to the CARDIS Steering Committee for their support and for giving me the opportunity to serve as Program Chair at such a recognized conference. The financial support of the gold sponsor Infineon Technologies, the silver sponsor NXP Semiconductors, the city of Graz, and the province of Styria was highly appreciated and allowed us to provide student stipends. Finally and most importantly, my thanks go to all the authors who submitted their work to CARDIS 2012 as well as to the invited speakers.

December 2012 Stefan Mangard

Organization

CARDIS 2012 was organized by Graz University of Technology, Austria.

Executive Committee

General Chair

Jörn-Marc Schmidt	Graz University of Technology, Austria

Program Chair

Stefan Mangard	Infineon Technologies, Germany

Publicity Chair

Jean-Jacques Quisquater	Université catholique de Louvain, Belgium

Program Committee

N. Asokan	University of Helsinki, Finland
Gildas Avoine	Université catholique de Louvain, Belgium
Chetali Boutheina	Trusted Labs, France
Josep Domingo-Ferrer	Rovari i Virgili University, Catalonia
Hermann Drexler	Giesecke & Devrient, Germany
Martin Feldhofer	NXP Semiconductors, Austria
Berndt Gammel	Infineon Technologies, Germany
Tim Güneysu	Ruhr-Universität Bochum, Germany
Helena Handschuh	Cryptography Research, USA and KU Leuven, Belgium
Michael Hutter	Graz University of Technology, Austria
Marc Joye	Technicolor, France
Ioannis Krontiris	Göthe Universität Frankfurt, Germany
Jean-Louis Lanet	Université de Limoges, France
Konstantinos Markantonakis	Royal Holloway, UK
Andrew Martin	University of Oxford, UK
David Naccache	ENS, France
Elisabeth Oswald	University of Bristol, UK
Catuscia Palamidessi	INRIA, France
Eric Peeters	Texas Instruments, USA
Erik Poll	RU Nijmegen, The Netherlands

Bart Preneel	KU Leuven, Belgium
Emmanuel Prouff	ANSSI, France
Matthieu Rivain	CryptoExperts, France
Pankaj Rohatgi	Cryptography Research, USA
Ahmad-Reza Sadeghi	TU Darmstadt, Germany
Jean-Pierre Seifert	TU Berlin, Germany
Sergei Skorobogatov	Cambridge University, UK
François-Xavier Standaert	Université catholique de Louvain, Belgium
Frederic Stumpf	Fraunhofer AISEC, Germany
Marc Witteman	Riscure, The Netherlands

External Reviewers

Guido Bertoni
Guillaume Bouffard
Cees-Bart Breunesse
Ileana Buhan-Dulman
Sébastien Canard
Xavier Carpent
Pierre-Louis Cayrel
Jean-Sébastien Coron
Joeri De Ruiter
Fabrizio De Santis
Cécile Delerablée
Alexandra Dmitrienko
François Durvaux
Ehab Elsalamouny
Junfeng Fan
Oriol Farràs
Matthieu Finiasz
Wieland Fischer
Hamza Fraz
Alpar Gergely
Karin Greimel
Vincent Grosso
Sardaouna Hamadou
Benedikt Heinz
Jens Hermans
Stephan Heuser
Stefan Heyse
Johann Heyszl
Jin-Meng Ho
Lars Hoffmann
Eliane Jaulmes
Elif Bilge Kavun
Justin King-Lacroix
Mario Kirschbaum
Ilya Kizhvatov
Miroslav Knezevic
François Koeune
Thomas Korak
Juliane Krämer
Mario Lamberger
Marc Le Guin
John Lyle
Benjamin Martin
Tania Martin
Dominik Merli
Oliver Mischke
Bruce Murray
Dmitry Nedospasov
Ventzi Nikov
Michael Østergaard Pedersen
Andrew Paverd
Roel Peeters
Pedro Peris-Lopez
Thomas Plos
Jürgen Pulkus
Thomas Pöppelmann
Prithvi Rao
Francesco Regazzoni
Alfredo Rial
Thomas Roche
Sami Saab
Ahmad Sabouri
Martin Schläffer
Jörn-Marc Schmidt
Thomas Schneider
Steffen Schulz
Raphael Spreitzer
Rolando Trujillo-Rasua
Michael Tunstall
Fabian Van Den Broek
Ingo von Maurich
Christian Wachsmann
Erich Wenger
Carolyn Whitnall

Table of Contents

Java Card Security

Protocols

Side-Channel Attacks I

Implementations

Implementations for Resource-Constrained Devices

Side-Channel Attacks II

Invited Talk

Towards the Hardware Accelerated Defensive Virtual Machine – Type and Bound Protection

Michael Lackner[1], Reinhard Berlach[1], Johannes Loinig[2],
Reinhold Weiss[1], and Christian Steger[1]

[1] Institute for Technical Informatics,
Graz University of Technology, Graz, Austria
{michael.lackner,reinhard.berlach,rweiss,steger}@tugraz.at
[2] NXP Semiconductors Austria GmbH, Gratkorn, Austria
johannes.loinig@nxp.com

Abstract. Currently, security checks on Java Card applets are performed by a static verification process before executing an applet. A verified and later unmodified applet is not able to break the Java Card sand-box model. Unfortunately, this static verification process is not a countermeasure against physical run-time attacks corrupting the control or data flow of an applet. In this piece of work, designs for Java Card Virtual Machines are investigated in relation to their ability to perform run-time security checks. These security checks are accelerated by hardware units and performed in parallel to CPU instructions that are executing concurrently. Attacks on the Java operand stack and local variables, which are elementary components for the Virtual Machine, are thwarted by type and bound protection. To enable these hardware checks, different designs of a defensive Java Card Virtual Machine are compared to their overheads on a prototype platform.

Keywords: Java Card, Defensive Virtual Machine, Hardware Countermeasure, Fault Attack, Logical Attack.

1 Introduction

Current applied static verification of Java Card applets provides insufficient security protection against run-time Fault Attacks. This is especially a problem in the field of multi-application Java Cards. In this field, cards are used in a wide range of applications (e.g., passport, e-money) and have the ability to perform post-issuance loading of new applets. An adversary provoking a Logical Attack by changing the bytecode or internal representation of an uploaded Java applet can get access to security related data from other applets or the Java Card Virtual Machine (JCVM) [12]. To thwart Logical Attacks, verification of applets is performed either off-card or on-card. This verification procedure is currently a static process performed once before an applet is executed. One of the most time and memory consuming checks performed is the bytecode verification [9,15].

Java Card applets are stored into non-volatile memories such as EEPROM. With the help of physical Fault Attacks it is possible for a JCVM to read out

S. Mangard (Ed.): CARDIS 2012, LNCS 7771, pp. 1–15, 2013.

incorrect values from these memories or skip CPU instructions. Therefore, it is possible to change the bytecode of stored applets to execute ill-formed Java instructions. Knowledge about these attack possibilities is used in [3] to create a new class of attacks called Combined Attacks. To perform Combined Attacks, applets which pass the verification process are used and become malicious in combination with a Fault Attack. Combined Attacks are used to bypass the Java Card sand-box, mounted by the static verification process, JCVM and Java Card Runtime Environment [11].

To guard the Java Card against run-time attacks, a so called defensive JCVM is needed [4]. This defensive JCVM can be reached by performing all checks done by the static verification process during run-time. However, this is currently not achievable because of the constrained hardware resources of today's Java Cards. In this work specific security checks, extracted from the Java Card specification [12], are performed on the executing bytecode during run-time. In this specification the data flow is exactly defined for every bytecode with some additional constraints which must be fulfilled.

To speed up bytecode checking during run-time, new hardware protection units are introduced in this work to speed up the checks performed on every bytecode. These hardware checks can be performed in parallel while the CPU performs its operations. Therefore hardware checks are a good solution for run-time checks that are performed very often, in contrast to software checks. Software checks slow down the whole system if they are performed on the same CPU that the standard operations are performed on. Beside this benefit, hardware checks also have the advantage of being more immune against additional fault injects onto the Java Card. This is due to the fact that software checks [13,5,2] are vulnerable to skipping them by additional Fault Attacks. This threat of skipping software security operations leads Vertanen in [16] to the conclusion that hardware assisted run-time checks are mandatory for enhancing run-time security for Java Cards.

This work introduces a hardware accelerated defensive JCVM which performs selected security checks on the executing bytecodes by hardware with a low computational overhead. These checks are also part of the verification process which is done statically before executing a Java applet. As far as we know, such a hardware accelerated defensive JCVM has not been introduced in literature before.

The contribution of this work is the definition of a run-time security policy extracted from the Java Card specification [12] to ensure that the executing bytecode performs valid operations. This run-time policy prevents type confusion and overflow/underflow attacks on the operand stack (OS) and local variable (LV) memory inside the JCVM. With this policy it is not possible for an adversary to perform type confusion between values of type *integralData* and object *references* on the OS and LV. Furthermore, two hardware accelerated defensive JCVM designs are presented with their main parts, such as additional hardware protection units and new CPU instructions. The new CPU instructions are used inside the JCVM to process bytecodes and communicate directly with the hardware protection units leading to a very low computational overhead. This communication is depicted in Figure 1.

Section 2 gives an overview of attacks on Java Cards, bytecodes violating the Java frame bounds and how to enable a defensive JCVM. Section 3 describes the security policy and the design of all defensive JCVMs introduced in this work. Section 4 presents the prototype implementation of these designs on a SystemC 8051 derivate. Section 5 analyses the run-time costs on execution speed and hardware changes needed to activate our JCVM designs. Finally, conclusions and future work are drawn in Section 6.

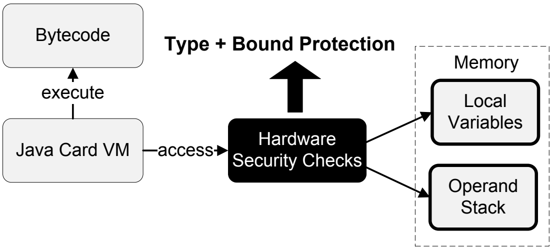


Fig. 1. In this work the operand stack and the local variables are protected during run-time by hardware accelerated security checks

2 Related Work

In this section an overview of possible attacks on the Java Card is given with focus on run-time fault attacks. Following this, previous work on run-time countermeasures and an overview of defensive JCVMs are presented.

2.1 Attack Overview

Attack scenarios on Java Cards are manifold [18,19]. Side Channel Attacks are used to draw conclusions of internal operations by studying physical phenomena of the chip. Invasive Attacks are used for optical or measurement analysis of internal components. Fault Attacks (FA) change the physical environment of the chip under attack [1]. These are for example, temperature changes, additional light of a laser or spikes in the power supply or clock source. These FA lead to an undefined behavior of the chip by skipping instructions or read/write errors to memory like the EEPROM. This is especially a problem for post-issuance loaded applets due to the fact that they are mostly installed in non-volatile memory. Therefore, a FA during the fetch process of a JCVM can lead to ill-formed applets even if a static verification was performed. This ill-formed code enables an adversary to circumvent the Java Card security model and enables an applet to have access to unauthorized resources. This security problem of FA to verified applets is well known in literature and is used to enable different attack paths [10,3,17,14].

2.2 Frame Bound Violation Attacks

Generally, inside every Java Frame, specific memory areas are reserved for OS, LV and internal frame data. Every time a new Java method is invoked, the JCVM creates a new frame and pushes it onto the Java stack. Specific implementation details for the Java Frame are not provided by Java Card specification. Therefore, the specific frame data depends on the particular implementation. In general the frame data contains a return address so that it is possible to return to the code of the old frame. The size needed for a frame and all its containing elements (e.g., OS, LV) is ascertained when the method is invoked and is not changing during method execution.

Ill formed bytecode can now access illegal memory regions by performing an OS or LV out of bound access as illustrated in Figure 2. In [5] an attack called EMAN2 was performed. There an invalid LV index was used by an ill formed bytecode *sstore* to access the memory region of the frame data where the return address of the current frame is stored. With the help of this ill formed bytecode, an adversary can set the return address to any value. In their attack the return address was set to the address of an array which leads to the security threat of executing adversary definable data. This illegal execution of data opens new security issues not treated here in detail. The threats of OS overflow/underflow and bytecodes using invalid LV index are thwarted by the run-time policy of this work.

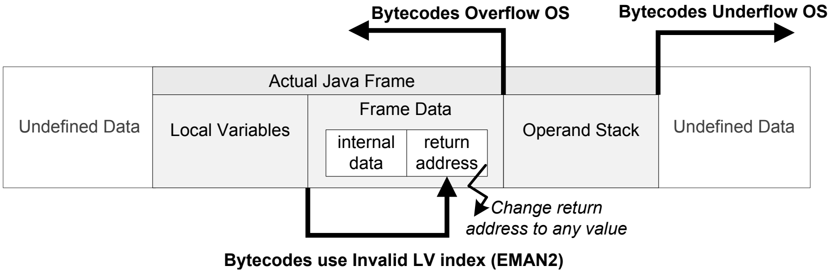


Fig. 2. OS overflow and underflow leads to illegal memory access outside the reserved OS memory space. An adversary who uses bytecode with invalid LV index can overwrite the return address of the current active frame [5].

2.3 Enabling a Defensive Virtual Machine

Currently the Java Card research community concentrates on finding attack paths to bypass the Java Card security model by FAs and Combined Attacks. [3,10,17,14]. Also a lot of effort is invested in exploiting and thwarting Side Channel Attacks [8]. In contrast to these big research topics, the question of how to enable a defensive JCVM is a research topic with little public attention. However, in the Java Card industry the know-how to enable defensive JCVM designs is of course available. This fact is proven by different works bringing the defensive nature of current available industrial Java Card products to light [10,7]. Techniques and knowledge that provide such a defensive design are of course not freely available. Currently research related to FA countermeasures is focussed on static verification of an applet and checking that the exact verified code is executed. This can be done by code integrity and control flow checks in software (SW) during run-time [13,5]. The annotations that enable these checks are stored in an additional component of a verified CAP-file. Research was also done to check the OS integrity against FA by performing double reads by SW [2].

This work focuses on a hardware accelerated defensive JCVM performing security checks during run-time. Based on a policy it checks if the executing bytecode is behaving correctly. Compared to current countermeasures in literature the approach in this work does not just check the integrity of the bytecode or OS, it performs checks based on a policy. This approach stops either manipulated applets loaded onto the card or run-time FA from violating this policy. Furthermore, performing these checks in hardware makes it more resistant to additional FA which are also able to skip additional software checks.

3 Design of the Defensive Virtual Machine

In this section the run-time security policies for all defensive JCVM designs in this work are shown. This is followed by our method of reducing all Java data types to two main types.

3.1 Defensive Run-Time Policy

The OS and LV, located in the Java Frame, are main parts of the JCVM. The JCVM is a stack machine and performs most operations on the OS. Therefore, securing these parts of the JCVM are the first steps to hampering or stopping Fault Attacks during run-time. The following two main policies for the OS and LV are retained by our defensive designs during run-time:

- **Frame Type Policy:** All bytecodes which access the OS or LV must use the right main data type (*integralData* or *reference*) which is expected by the bytecode during its execution. In this work all numerical types are combined (*boolean*, *byte*, *short*) to the main data type *integralData.* All object references (e.g., *short array*, *byte array*, *Class A*) are combined to the main data type *reference.*
- **Frame Bound Policy:** Bytecodes operating on the OS or LV are not allowed to access data outside the frame bounds. This means that bytecodes are not allowed to overflow or underflow the OS. Furthermore, all bytecodes accessing the LV must be inside the borders of the reserved LV memory area.

Policy Creation: The two policies above were extracted from the JCVM specification [12] where for every bytecode a textual description of the operation is given. In this specification it is for every bytecode defined from which JCVM component needed operands are taken and results are written back. Also the type information is specified for every operand and result value. Such a bytecode specification for the *sstore* instruction is listed below. This bytecode consists of two bytes, the opcode (0x29) and an index referencing to an item of the LV.

> *"The index is an unsigned byte that must be a valid index into the local variables of the current frame (Section 3.5, "Frames"). The value on top of the operand stack must be of type short. It is popped from the operand stack, and the value of the local variable at index is set to value."* [12]

The requirement that bytecodes perform no OS stack overflow/underflow, access the right LV index and operate with the right types on the OS and LV is crucial for the security concept of the Java Card and therefore checked by all defensive JCVM designs of this work.

3.2 Design of the Defensive JCVMs

In this section we introduce two designs for a defensive JCVM which fulfill the security policies defined in the previous section. A general overview of the defensive JCVM designs is shown in Figure 3 and described how they are used in more detail below. Note that our defensive JCVM designs are not able to thwart all sort of attacks on a Java Card. Examples of such undetected attacks are control flow changes (skipping a branch instruction) or data corruption (read corrupted values from the RAM).

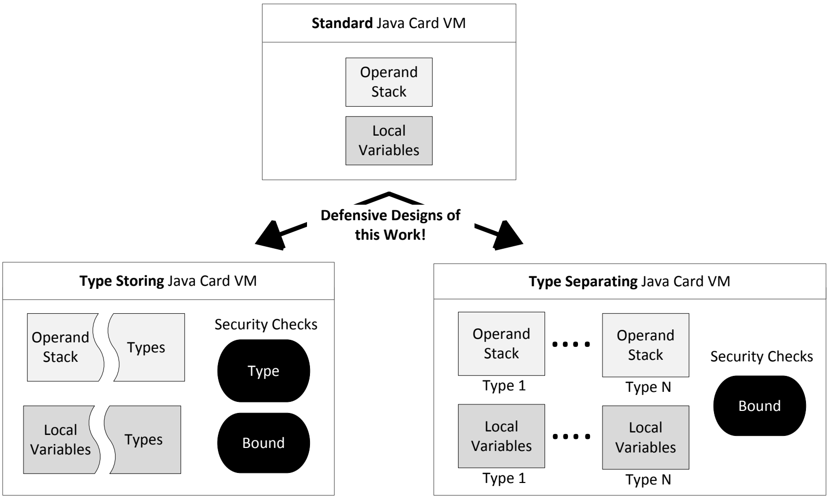


Fig. 3. In this work two designs are used to fulfill the run-time security policy

- **Type Storing:** Every entry on the OS or LV is extended with type information in order to distinguish between *integralData* and *reference* during run-time. During run-time it is now possible to check if the expected type for the bytecode is on the OS or LV which is the obvious defensive approach to enable a Defensive JCVM. A disadvantage of this approach is the additional memory needed for type storing and the computational overhead to perform type checking.
- **Type Separating:** Every Java main data type (*integralData* and *reference*) operates on its own OS and LV memory area. No general OS and LV area where all data types occur exists. Type confusion between the two main data types is therefore no longer possible during run-time because every bytecode always receives the right type. A disadvantage of the Type Separating design is that an attack is only detected by its security related side effects on the current frame. Such a side effect is for example an OS underflow for a specific type.

3.3 Two Types for Type Storing and Type Separating

In this section we introduce our approach for separating the Java Card types into two main data types to enable the Type Storing and Type Separating JCVM that was presented in the previous chapter. Java bytecodes are highly typed. This means that based on the data type different opcodes exist for the same operation [12, Table 3-1]. For example, only the *sstore* bytecode is allowed to push integral data types (boolean, byte, short) into the LV. Another Java bytecode is used to store an element of type *reference*, pointing to an object, into the LV. It is therefore possible to differentiate between two main data types and distinguish them just by looking at the bytecode. In this work they are called *integralData* and *reference*:

- ***integralData.*** These are the primitive constant data types that represent the numerical values of the JCVM: *boolean*, *byte*, *short*. Elements of this data type can be deliberately created by executing the bytecode *sconst_1*. This bytecode pushes an integral value 1 with type *short* on the OS.
- ***reference.*** These are all kinds of *references* to objects and the *returnAddress* type to enable sub-routines. An applet programmer can only indirectly create elements of type *reference*. For example the bytecode *new_array* pushes the reference of a newly created array object onto the OS. This address can have any logical structure and does not have to correlate with physical addresses of objects stored on the card. For example the JCVM can create a random number and a look up table maps this number to a real memory address.

By separating these two main data types it is no longer possible for an adversary to create object references with a defined value by confusing *integralData* and *reference*. It is also not possible for an adversary to get deeper insight into how the JCVM represents references. For this insight an adversary would have to perform type confusion between *reference* and *integralData* by sending the reference out of the card from the APDU buffer. The APDU class is responsible for receiving and sending data to off-card applications.

Thwarted Threats in Literature. This sort of type confusion between integral data and object references is well known and often used as the first step of an attack path [16,10,7,5,17]. This attack path can enable an adversary creating self mutable code by executing data from a Java array [7,5] or even gain access to forbidden methods of objects [17].

Note that type confusion between different objects such as *short array* and *Class A* object is not detected by the defensive JCVM designs in this work. This is because all object references are assigned to the *reference* main data type and cannot be distinguished. This determination also applies to the main data type *integralData* where it is not possible to detect type confusion between *byte* and *short*.

4 Prototype Implementation

In this work five different prototype JCVMs were implemented in C and assembly language. The JCVMs are based on the Classic Edition of the Java Card specification [12]. The hardware (HW) platform on which they run is an 8-bit Smart Card model written in SystemC [6]. This model is memory and instruction cycle accurate. Into this HW platform new typed CPU instructions were implemented. Furthermore, additional HW protection units were added to enable HW accelerated security checks for bytecodes accessing the OS and LV memory area.

4.1 Additional CPU Instructions

The information decoded into the new CPU instructions is illustrated in Figure 4. New typed CPU instructions are used by our JCVMs to process the bytecodes and perform access to the OS and LV memory regions. These decoded pieces of information are the access type (Read, Write), the destination of the accessing memory (OS, LV) and the type which should be written/read (*integralData*, *reference*, *untyped*). With the help of these pieces of information the protection units are able to check if the new CPU instruction doesn't perform a security policy violation during run-time. Such a violation is for example a LV element address which is outside the actual LV memory bounds.

Two examples of how to use these new instructions inside the JCVM program code in order to process the Java bytecodes is outlined in Figure 5. The *sadd* bytecode first reads two values from the OS by the new CPU instruction *Read_OS_integralData*. The result of the addition of these values is then written back by the instruction *Write_OS_integralData*. Another example is the bytecode *astore_0*. This bytecode uses the CPU instruction *Read_OS_reference* to read a reference value from the OS and stores it into the LV by the instruction *Write_LV_reference*. The big advantage in the sense of computational overhead of the new typed CPU instructions is that the JCVM can communicate very effectively with the HW protection units by using the new CPU instructions.

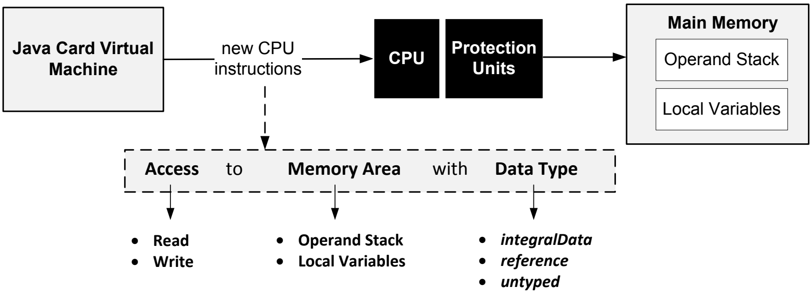


Fig. 4. The prototype JCVMs proposed in this work uses new assembly instructions to access the run-time protected OS and LV memory regions

```
sadd:                                //Add two short values on the OS
   short VAR1, VAR2, SUM;            //Create variables for sadd
   VAR1 = Read_OS_integralData;      //Read first operand from OS
   VAR2 = Read_OS_integralData;      //Read second operand from OS
   SUM  = VAR1 + VAR2;               //Sum the two operands
   Write_OS_integralData = SUM;      //Write the sum back onto the OS

astore_0:                            //Store reference from OS to LV
   short REF1;                       //Create variables for astore_0
   REF1 = Read_OS_reference;         //Read reference from OS
   Write_LV_reference(0) = REF1;     //Write reference into LV element 0
```

Fig. 5. Pseudocode example of the two bytecodes *sadd* and *astore*, processed by the JCVM. The JCVM uses our new CPU instructions to access the OS and LV memory.

4.2 Additional Hardware Protection Units

An overview of the new CPU instructions and the protection units needed to activate the Type Storing and Type Separating JCVM is presented in Figure 6. Based on the new CPU instructions introduced in the previous section, our HW protection units restrict the access to the security critical memory regions of the OS and LV. The Type Storing JCVM needs a type protection unit to check if the type expected by the bytecode is also available on the OS or LV.

- **Bound Protection Unit (BPU):** This unit is responsible for thwarting attacks performing an OS overflow or underflow. Furthermore, all bytecodes accessing the LV using a wrong index are detected. The BPU is used by the Type Storing and Type Separating JCVM prototype.
- **Type Protection Unit (TPU):** The TPU is responsible for checking that the bytecodes that are accessing the OS and LV are operating with the right data type. The TPU is only needed to enable the Type Storing JCVM.

4.3 Type Storing JCVM Implementation Details

To enable a Type Storing JCVM, the TPU must store additional type information for every element held by the OS and LV. Due to the fact that these two parts are located in RAM, one additional type bit was added to every 8-bit word. This bit enables the distinction between the two main data types *integralData* and *reference.*

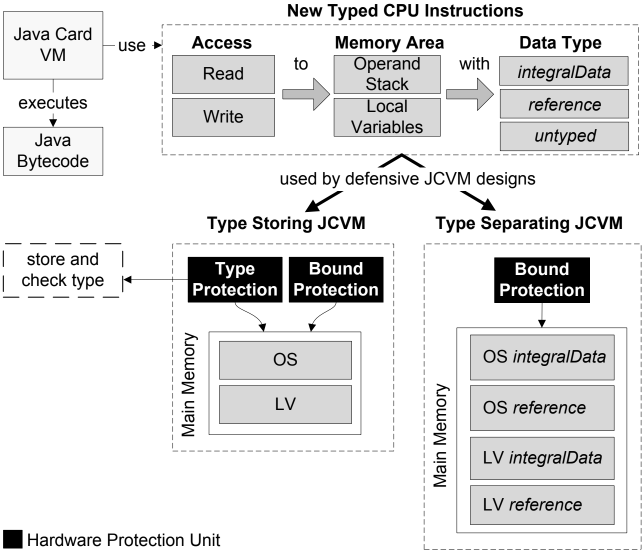


Fig. 6. Implementation overview of the two JCVM prototypes. Hardware protection units perform run-time checks on the OS and LV.

4.4 Type Separating JCVM Implementation Details

In this section we give insight into the detail of how the Type Separating JCVM was implemented and describe a tool chain to enable it. The Type Separating JCVM performs all bytecode operations on the right typed OS and LV. This Type Separating approach avoids type confusion. The type checking problem is reduced to a bound checking problem.

Most bytecodes work well with our run-time type separating approach to two main data types (*integalData*, *reference*). An exception are the bytecodes operating with undefined types on the OS: *pop*, *pop2*, *dup*, *dup2*, *dup_x* and *swap_x*. In this paper we call them untyped bytecodes. For these untyped bytecodes the JCVM does not know on which of the two separated OS, specific operations are performed during run-time.

As a solution for this problem the missing type information was added directly into the bytecode by using unused bytecodes which are not defined in the Java Card specification. For example, the *pop* instruction is either convert to *pop_reference* or *pop_integralData*. The JCVM now knows right after fetching a new bytecode, which typed OS it must operate on. Therefore, no execution speed is wasted searching for the type information in additional components uploaded on the card. In the JCVM specification [12] only 185 (0x00 to 0xb8) of all available 8-bit bytecodes are specified. The unused bytecodes from 0xb9 to 0xfd can be used to decode the operand stack type information that is needed directly into the instruction.

To perform the exchange of untyped bytecodes with new typed bytecodes we propose a static replacement process performed once for every method. To speed up this process, type information obtained during the bytecode verification process can be used to exchange the untyped bytecodes with typed ones. However, in this work the replacement process for the untyped bytecodes is not looked at in detail.

5 Prototype Results and Discussion

In this section we show the computational overhead coming from full software (SW) implementations compared to running our HW accelerated prototypes. The SW implementations perform the same security checks that are performed by the HW protection units. Furthermore, the additional hardware overhead is compared between all prototypes.

5.1 Computational Overhead

Performing all security checks in SW increases the computational overhead significantly for frequently executed bytecodes, as illustrated in Figure 7. For example the *sload* bytecode executed by a Type Storing prototype in SW has a computational overhead of around 107% caused by the following run-time SW operations:

- Check if the index parameter to the LV is valid.
- Check if the element at the LV index is of type *integralData*.
- Check if pushing a value from the LV index to the OS provokes an overflow.
- Store the fact that the new value on the OS is of type *integralData*.

If the *sload* bytecode is executed on a HW accelerated prototype the overhead decreases to 5%. In Table 1 different groups of bytecodes are compared to their computational overhead. As expected, the HW accelerated prototypes consume much less computational overhead compared to prototypes which implement the checks in SW.

5.2 Hardware Overhead

In this section we give an overview of the HW modifications used to activate our HW accelerated prototypes, as depicted in Table 2. The instruction set of a standard 8051 microcontroller consists of 255 opcodes. Adding our new CPU

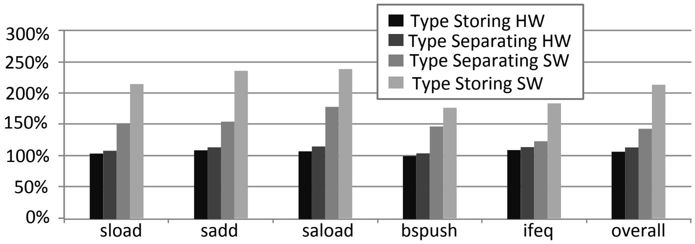


Fig. 7. Run-time measurement for specific bytecodes and the overall time of all implemented bytecodes for different JCVM implementations. Measurements are normalized to a JCVM without any run-time security checks.

Table 1. Computational overhead for all prototypes, normalized to a JCVM without performing run-time security checks

	Type Storing		Type Separating	
Bytecode Groups	HW	SW	HW	SW
1: Arithmetic/Logic	+7%	+123%	+7%	+47%
2: Local Variable Access	+5%	+152%	+9%	+52%
3: Operand Stack Manipulation	+5%	+119%	+3%	+54%
4: Control Transfer	+8%	+77%	+9%	+23%
5: Array Creation/Manipulation	+6%	+111%	+9%	+59%
Overall	+6%	+107%	+8%	+38%

Table 2. HW modifications needed to activate the HW accelerated run-time security checks of the prototypes

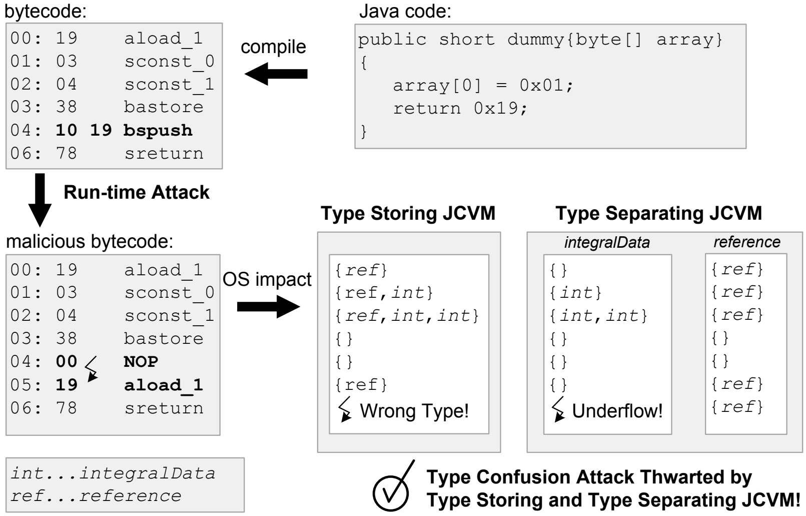
Additional Hardware	Type Storing	Type Separating
New 8051 CPU Instructions	9 (+3,5%)	8 (+3,1%)
New 8-bit Control Registers (SFRs)	11 (+52,4%)	15 (+71,4%)
New Bound Protection Unit (BPU)	Yes	Yes
New Type Protection Unit (TPU)	Yes	No
Extend RAM word with type bit	Yes	No

instructions means an overall CPU instruction increase of around only 3,5%. Another important hardware modification is that the RAM module of the HW accelerated Type Storing JCVM was extended with an additional type bit for every memory word in order to differ between the main data types *integralData* und *reference*. Therefore, overall RAM memory size increases to 12,5%.

5.3 Type Confusion Attack Example

An example of a run-time attack on the Java Card prototypes in order to perform type confusion between *integralData* and *reference* is illustrated in Figure 8.

There, a run-time attack changes the *bspush* code 0x10 0x19 to 0x00 0x19. The JCVM interprets 0x00 as NOP instruction and performs no action. The following byte 0x19 is interpreted as the bytecode *aload_1* which pushes an array reference onto the OS. The *sreturn* instruction would now take the array reference and push it back to the calling function. An adversary is now able to use the array *reference* as an *integralData* which enables different attack paths such as executing the data inside the array [16,10,7,5,17]. Both defensive JCVMs designs from this work are able to thwart this type confusion attack on the OS between *integralData* and *reference*.



bytecode:

```
00: 19    aload_1
01: 03    sconst_0
02: 04    sconst_1
03: 38    bastore
04: 10 19 bspush
06: 78    sreturn
```

Java code:

```
public short dummy{byte[] array}
{
   array[0] = 0x01;
   return 0x19;
}
```

malicious bytecode:

```
00: 19    aload_1
01: 03    sconst_0
02: 04    sconst_1
03: 38    bastore
04: 00    NOP
05: 19    aload_1
06: 78    sreturn
```

```
int...integralData
ref...reference
```

Fig. 8. Run-time type confusion attack on the OS to receive the address of an array. All defensive JCVM implementations of this work are able to thwart this attack.

Type Storing: By using the new typed CPU instructions, it is decoded inside the JCVM code that the *sreturn* instruction expects a value of type *integralData* on the OS. The previously executed instruction *aload_1* pushed the reference of an array with type *reference* on the OS. Therefore, the TPU hardware module finds the wrong type for the *sreturn* bytecode on the OS and throws a security exception.

Type Separating: Both main data types have their own OS and LV memory areas during run-time. Therefore, the malicious instruction *aload_1* pushes an array reference onto the *reference* OS containing all values from type *reference*. The *sreturn* bytecode tries to pop data from the *integralData* OS which is empty. The return operation is now aborted by a security exception because an underflow on the *integralData* OS is detected by the BPU hardware module.

6 Conclusion and Future Work

This work presents Java Card Virtual Machine (JCVM) designs to counter different Fault Attacks. This is done by performing run-time security checks based on a security policy for each bytecode. These policies ensure that each bytecode which operates on the operand stack or the local variables memory area uses the right data type (*integralData* or *reference*). Furthermore bytecodes which overflow or underflow the OS or LV are detected. These run-time checks are accelerated by hardware protection units to make it harder to skip these checks with additional Fault Attacks. Furthermore, the defensive JCVM designs are profiting from the parallel execution of the hardware checks by having very low computational overhead.

In this work the design of a Type Storing and Type Separating JCVM were shown. Both designs were implemented on a Java Card prototype platform with several additional hardware changes. The requirements of the hardware to enable run-time checking by hardware protection units were listed for both defensive JCVM designs. We measured that these hardware accelerated prototypes consume 6% and 8% more execution time overall compared to a JCVM without any additional run-time security checks. This overhead is very low compared to prototypes which perform all run-time security checks in software and consume around 107% and 38% more execution time. Therefore, we have shown that our approach of performing additional security checks during run-time by using hardware units is feasible especially in the case of resource constrained Java Cards.

For future work we will focus on the bytecode replacement process of untyped bytecodes required by the defensive Type Separating approach. This transformation is needed to give the JCVM type information needed during run-time to process untyped bytecodes like *pop*. Furthermore, we are working on increasing the number of separated main types so that it is also possible to detect type confusion between *integralData* like *short* and *byte*.

Acknowledgement. The authors would like to thank the Austrian Federal Ministry for Transport, Innovation, and Technology, which funded the CoCoon project under the FIT-IT contract FFG 830601. We would also like to thank our project partner NXP Semiconductors Austria GmbH.

References

1. Bar-El, H., Choukri, H., Naccache, D., Tunstall, M., Whelan, C.: The Sorcerer's Apprentice Guide to Fault Attacks. Proceedings of the IEEE 94(2), 370–382 (2006)
2. Barbu, G., Duc, G., Hoogvorst, P.: Java Card Operand Stack: Fault Attacks, Combined Attacks and Countermeasures. In: Prouff, E. (ed.) CARDIS 2011. LNCS, vol. 7079, pp. 297–313. Springer, Heidelberg (2011)
3. Barbu, G., Thiebeauld, H., Guerin, V.: Attacks on Java Card 3.0 Combining Fault and Logical Attacks. In: Gollmann, D., Lanet, J.-L., Iguchi-Cartigny, J. (eds.) CARDIS 2010. LNCS, vol. 6035, pp. 148–163. Springer, Heidelberg (2010)

4. Barthe, G., Dufay, G., Jakubiec, L., de Sousa, S.M.: A Formal Correspondence between Offensive and Defensive JavaCard Virtual Machines. In: Cortesi, A. (ed.) VMCAI 2002. LNCS, vol. 2294, pp. 32–45. Springer, Heidelberg (2002)
5. Bouffard, G., Iguchi-Cartigny, J., Lanet, J.-L.: Combined Software and Hardware Attacks on the Java Card Control Flow. In: Prouff, E. (ed.) CARDIS 2011. LNCS, vol. 7079, pp. 283–296. Springer, Heidelberg (2011)
6. IEEE: Open SystemC Language Reference Manual IEEE Std 1666-2005, IEEE
7. Iguchi-Cartigny, J., Lanet, J.L.: Developing a Trojan applets in a smart card. Journal in Computer Virology 6, 343–351 (2010)
8. Krieg, A., Grinschgl, J., Steger, C., Weiss, R., Haid, J.: A Side Channel Attack Countermeasure using System-On-Chip Power Profile Scrambling. In: 2011 IEEE 17th International On-Line Testing Symposium (IOLTS), pp. 222–227 (July 2011)
9. Leroy, X.: Java Bytecode Verification: An Overview. In: Berry, G., Comon, H., Finkel, A. (eds.) CAV 2001. LNCS, vol. 2102, pp. 265–285. Springer, Heidelberg (2001)
10. Mostowski, W., Poll, E.: Malicious Code on Java Card Smartcards: Attacks and Countermeasures. In: Grimaud, G., Standaert, F.-X. (eds.) CARDIS 2008. LNCS, vol. 5189, pp. 1–16. Springer, Heidelberg (2008)
11. Oracle: Runtime Environment Specification. Java Card Platform, Version 3.0.4, Classic Edition (2011)
12. Oracle: Virtual Machine Specification. Java Card Platform, Version 3.0.4, Classic Edition (2011)
13. Séré, A.A.K., Iguchi-Cartigny, J., Lanet, J.-L.: Checking the Paths to Identify Mutant Application on Embedded Systems. In: Kim, T.-H., Lee, Y.-H., Kang, B.-H., Ślęzak, D. (eds.) FGIT 2010. LNCS, vol. 6485, pp. 459–468. Springer, Heidelberg (2010)
14. Sere, A., Iguchi-Cartigny, J., Lanet, J.L.: Evaluation of Countermeasures Against Fault Attacks on Smart Cards. International Journal of Security and Its Applications 5(2), 49–61 (2011)
15. Sun Microsystems Inc.: Java Card 2.2 Off-card Verifier. White Paper (June 2002)
16. Vertanen, O.: Java Type Confusion and Fault Attacks. In: Breveglieri, L., Koren, I., Naccache, D., Seifert, J.-P. (eds.) FDTC 2006. LNCS, vol. 4236, pp. 237–251. Springer, Heidelberg (2006)
17. Vetillard, E., Ferrari, A.: Combined Attacks and Countermeasures. In: Gollmann, D., Lanet, J.-L., Iguchi-Cartigny, J. (eds.) CARDIS 2010. LNCS, vol. 6035, pp. 133–147. Springer, Heidelberg (2010)
18. Witteman, M.: Advances in Smartcard Security. Information Security Bulletin, 11–22 (July 2002)
19. Witteman, M.: Java Card Security. Information Security Bulletin, 291–298 (July 2003)

Dynamic Fault Injection Countermeasure
A New Conception of Java Card Security

Guillaume Barbu, Philippe Andouard, and Christophe Giraud

Oberthur Technologies
Security Group
4, allée du Doyen Georges Brus, 33 600 Pessac, France
{g.barbu,p.andouard,c.giraud}@oberthur.com

Abstract. Nowadays Fault Injection is the main threat for any sensitive applications being executed on embedded devices. Indeed, such an attack allows one to efficiently recover any secret or to gain unauthorized privileges if no appropriate countermeasure is implemented. In the context of Java Card applications, the main method to counteract Fault Injection consists in adding redundancy for sensitive operations and integrity verification for sensitive variables. While being efficient from a security point of view, such a method substantially impacts the performance of the application. In this article we introduce a new pragmatic approach to counteract Fault Injection by dynamically increasing the security level of the application. This methodology, based on upgrading the Java Card Virtual Machine, allows us to optimize the performance of sensitive applications in every day life while providing a strong security level as soon as an attacker tries to disturb their executions.

Keywords: Java Card, Fault Injection, Countermeasures.

1 Introduction

1996 was an amazing year for attacks in the embedded environment. Indeed, the concepts of *Side-Channel Analysis* (SCA) and *Fault Injection* (FI) were published that year and they allow an attacker to recover secrets stored in embedded devices even if they are protected by very strong cryptography.

The first kind of attacks have been published by Kocher in [1] where he noticed that the difference of time when executing an application could depend on the secrets manipulated by the device. This attack was then extended by using the power consumption or the electromagnetic radiation of the device as side-channel leakage instead of the execution timing [2,3]. SCA is now reinforced by numerous new attacks and countermeasures every year.

The second kind of attacks revealed in 1996 were published through a press release by the company Bellcore [4]. Three researchers of this company noticed that if the execution of a cryptographic implementation can be disturbed, then the analysis of the corresponding faulty output can lead to the disclosure of the secret key. The announcement of this new way of attacking embedded devices

S. Mangard (Ed.): CARDIS 2012, LNCS 7771, pp. 16–30, 2013.

aroused enthusiasm amongst the cryptographic community and a dozen articles dealing with this subject were published in the few weeks after the Bellcore's announcement. As for SCA, a very large amount of new attacks and countermeasures are published every year to extend the domain of FI which now applies not only to cryptographic algorithms but to each and every kind of application being executed on embedded devices.

These new attacks have been a major breakthrough for the smart card industry. Indeed, at the beginning of 1996, the security of embedded applications relied on the theoretical security of the cryptography which was implemented and on their resistance to *Logical Attacks* (LA) which were known for years and for which the countermeasures were well-known. One year later, basing the security of an application on these two factors only was hopeless. The developers had then had to invent and to implement ingenious side-channel and fault countermeasures which must have the lowest impact on the performances of the application.

At that time, another breakthrough occurred in the smart card environment: the first Java enabled smart card was produced [5]. Java Cards allow the developer to implement an application independently from the device on which it is going to be executed. Such an abstraction layer is provided by the Java Card Virtual Machine (JCVM) which interprets the Java basic instructions, called *bytecodes*, and executes the corresponding instructions for a specific device. Therefore, executing a brand new Java Card application on each and every Java Card on the market costs only one development, leading to the very fast deployment of such an application which cannot be achieved when using native products. Originally used in the mobile environment, Java Cards are now widely used in banking and identity environments where the constraints in terms of security are very strong. Therefore Java Card applications, called *applets*, possibly together with the JCVM, must implement logical, side-channel and fault countermeasures to prevent them from being tampered with.

In this work we present a new way of counteracting FI attacks in the context of Java Cards. Up to now, securing an applet against fault attacks means mainly adding redundancy on sensitive operations and verifying the integrity of the sensitive objects at the applet level. Although being efficient against FI, this leads to a very important overhead in terms of performance and memory consumption. This approach faces its limit when the application has strong constraints in terms of performances which is always the case for contactless applications for instance. Moreover, our new solution also aims at another problematic: why would honest customers (which are the vast majority) have to pay for the dishonest ones? For instance, why would they have to wait 400μs to perform a transaction whereas the same application could run twice as fast if fault countermeasures were removed? Our concept is based on upgrading the JCVM in order to dynamically enforce the security level of an application when detecting an attack. This allows the device to execute by default an application with critical countermeasures only, being thus very fast, and to activate the maximum security level as soon as an attack is detected.

The rest of this paper is organised as follows. In Section 2 we recall some generalities about fault attacks on Java Cards as well as a brief description of the main corresponding countermeasures. In Section 3 we present our new protection concept and we describe two different ways of applying it in practice. We also discuss the benefits and drawbacks of our proposals versus the traditional way of securing an application by adding redundancy and integrity verifications. Finally, Section 4 concludes the paper.

2 Fault Injection Attacks and Common Countermeasures

FI attacks and the analysis of their consequences are well-known in the context of embedded cryptography [6]. However, as stated in [7] and [8], such attacks are absolutely not restricted to arithmetic computations or cryptographic algorithms and are then likely to target any part of an embedded system.

In this section, we briefly present the most common FI attacks against Java Card platform and how they have been combined with LA to bypass specific Java Card security mechanisms. Secondly, we present the usual countermeasures to prevent such attacks.

2.1 Attacks against Java Card Platforms

The first attacks that have been mounted against Java Card platforms were LA. These are usually based on the corruption of the binary representation of a Java Card application (.CAP or .CLASS file) into a so-called *ill-formed application* before it is loaded on-card [9]. Such modifications aim at circumventing certain controls enforced by the JCVM. However in most cases, they also make the application illegal with regards to the Java Card specifications. Therefore the modified application should not be able to pass static analysis tools such as the Java *bytecode verifier*. The bytecode verification being a costly process, it is generally executed off-card on Java Card 2.2.2 and earlier, as a part of the application development tool chain. The usual philosophy of LA is then to skip this step and to directly load unverified applications on platforms allowing it.

On the other hand, FI has been mainly used on Java Card applications to disrupt conditional branching instructions to force a jump in a given branch, favorable to the attacker [10]. For instance, let us observe the piece of code depicted in Listings 1 and 2 which update the balance of an electronic wallet depending on whether there is a purchase or a refund.

Although the test `if` is performed on a boolean variable, one may note that there is no boolean type at the bytecode level. The Java compiler produces only bytecodes manipulating values of type `int` when processing operations on boolean variables. Therefore the Java specifications mandate that any value different from 0 will be considered as `true`. From this remark, an attacker disturbing b when performing a purchase will obtain a credit of her e-wallet instead of a debit.

Listing 1. Standard *if-then-else* statement

```
// assume b is a boolean.
if ( b ) {
    // e-wallet credit
}
else {
    // e-wallet debit
}
```

Listing 2. Standard *if-then-else* statement (bytecode sequence)

```
     iload_1
     ifeq L1
     // e-wallet credit
     goto L2
L1:  // e-wallet debit
L2:  ...
```

FI have also been used to disturb values returned by methods of the APIs. For instance, the `arrayCompare()` method returns 0 if the two buffers provided as input are identical. If such a method is used to compare a PIN (Personal Identification Number) or a MAC (Message Authentication Code), an attacker presenting a wrong value can force its acceptance by sticking at 0 the corresponding return value.

Moreover, it is now common to assume that attackers with high attack potential are able to perform two faults during the execution of the same command [11]. This statement has a strong impact on the cost of the countermeasures as it will be shown in Section 2.2.

To conclude this brief overview of attacks on Java Card platforms, we recall that recently the use of FI to provoke incorrect behaviours within a malicious but well-formed application appeared as a possible solution to attack Java Card platforms where the bytecode verification is mandatory [12, 13]. In these publications, FI is used to bypass certain security mechanisms in order to allow a LA. The so-called *Combined Attacks* allow then to take benefits of both FI and LA. Indeed, they are more realistic than LA since they do not rely on an unverified application loading and potentially more powerful than FI attacks since the malicious application can make permanent changes and act like a *Trojan* inside the card [14].

2.2 Common Countermeasures and Main Drawbacks

As presented above, FI is a real threat for sensitive applications being executed on a Java Card platform. To counteract such attacks, applet developers must implement specific countermeasures to detect any incoherence during the applet execution. As FI mainly focuses on disturbing conditional branchings and methods execution, one of the most efficient way to detect a disturbance is to add redundancy for each and every sensitive conditional branching and call to a sensitive method. For instance, if we want to protect the `if` used in the code of Listing 1 against double fault attacks, one has to add redundancy testing in the branch favorable to the attacker such as depicted in Listings 3 and 4.

Listing 3. 2^{nd}-order-secured *if-then-else* statement

```
// assume b is a boolean.
if ( !b ) {
  // e-wallet debit
}
else if ( b ) {
  if ( !b ) {
    ISOException.throwIt(
       ATT_DET_SW);
  }
  else if ( b ) {
    // e-wallet credit
  }
  else {
    ISOException.throwIt(
       ATT_DET_SW);
  }
}
else {
  ISOException.throwIt(
     ATT_DET_SW);
}
```

Listing 4. 2^{nd}-order-secured *if-then-else* statement (bytecode sequence)

```
     iload_1
     ifne L1
     // e-wallet debit
     goto L2
L1:  iload_1
     ifeq L3
     iload_1
     ifne L4
     bipush 18 <ATT_DET_SW>
     invokestatic #6 <throwIt>
     goto L2
L4:  iload_1
     ifeq L5
     // e-wallet credit
     goto L2
L5:  bipush 18 <ATT_DET_SW>
     invokestatic #6 <throwIt>
     goto L2
L3:  bipush 18 <ATT_DET_SW>
     invokestatic #6 <throwIt>
L2:  ...
```

As one can easily observe by comparing Listings 1 and 3, the cost of such a countermeasure is very important. Table 1 illustrates the size and the execution time of both the unsecured and secured versions of the previous sample code.

In order to overcome any specific platform implementation, and therefore specific optimizations, the extra execution time is expressed in terms of number of executed instructions multiplied by t_{ins} where t_{ins} is the average execution time for an instruction.

Table 1. Compared size and execution time overhead of unsecured and 2^{nd}-order-secured *if-then-else* statement

Mnemonic	Listing	Size (byte)	Timing
Unsecured	2	0	0
2^{nd}-order-secured	4	30	$6 \cdot t_{ins}$

Such an approach is also valid to protect disturbance of methods execution, for instance by executing three times the method `arrayCompare()` and checking that the three different outputs are coherent. This will therefore multiply by a factor 3 the time required to compare two buffers.

To ensure the security of an application, it is of common sense to implement protections against the most efficient practical attacks which are currently double fault attacks. However, even for the best attackers, it is impossible to achieve such attacks the first time round. Indeed, the attacker will have a few failures before succeeding in bypassing a double conditional testing for instance. From this observation, an obvious solution would be to deactivate all security sensitive applications if the card is under attack. However, such an option cannot be applied in some contexts where the provider wants to keep the functionality alive as long as possible (e.g. in the context of Secure Elements or SIM cards). In the next section, we present a new security concept for Java Card platforms. This new methodology is based on modifying the JCVM in such a way that it can dynamically increase the security level of the application when an attack attempt is detected.

3 Dynamic Fault Injection Countermeasure: A Generic Concept Available in Different Flavours

As stated in Section 2, common countermeasures against FI strongly penalize the performance of Java Card applications on a permanent basis. However, applications deployed on the field do not always have to face a real attacker. This section describes how the security of Java Card applications can be modulated by the JCVM without loss of security insurance and details two particular methods to implement this dynamic security concept.

3.1 The Dynamic Security Concept

The execution of an application within the JCVM relies on the interpretation of the bytecode instructions it is made of. Typically, we can consider that the JCVM interpreter associates a given bytecode value to a given function, implementing the Java instruction, according to the JCVM specification [15]. The interpretation of an application is then operated according to a **fetch-decode-execute** sequence, similarly to most *real* machines (by opposition to *virtual* machines), where:

fetch corresponds to the reading of the instruction value in the bytecode array of the current method;

decode corresponds to the translation from the read integer value to the function implementing the corresponding bytecode instruction in the underlying machine's language;

execute corresponds to the execution of the selected function.

Having hands on this sequence, the JCVM can dynamically alter these different steps in order to modify the behaviour of an application on the fly. The generic concept we describe here consists in using this capacity to adapt the security level of a given application to the threats it actually faces. That is to say that the JCVM initially enforces a given security level which will be raised if an attack is detected. As a result, the performance of the application are preserved for honest users whereas attackers will have to deal with the augmented security level.

The security insurance of our concept relies on the state of fact that a few attempts at least are necessary before achieving a successful FI attack. After the detection of a first attack, potentially all countermeasures are activated and the security of the application is ensured in a traditional way.

A prerequisite to the implementation of our concept is then to be able to categorize the different countermeasures ensuring the security of the application into different groups, so that the first group of countermeasures would be always activated, whereas the other group(s) would be activated only after an attack has been detected, according to our concept.

Several options can be adopted to achieve such a discrimination, either based on the attacker's fault injection capabilities or on the data to protect.

Fault attack order. First, we can categorize the countermeasures according to the order of the fault attack they are meant to counteract. That is to say that countermeasures against 1^{st}-order attacks (*i.e.* single fault attacks) would be always activated, 2^{nd}-order attack would only be activated if an attack is detected, etc.

Standardized asset hierarchy. Second, we can categorize the countermeasures according to the sensitivity of the assets they protect. For instance in the scope of a banking application, countermeasures protecting primary assets, such as the PIN and the DES and RSA secret keys, would be always activated, whereas countermeasures protecting secondary assets, such as the PTC (PIN Try Counter) and the CRM (Card Risk Management), would only be activated if an attack is detected.

Custom asset hierarchy. Similarly, we can imagine to let application developers define and organize the assets by using dedicated annotations for instance.

In the following sections, we propose different solutions to implement the dynamic security concept and we discuss their relative advantages.

3.2 *Vanilla*: Inhibiting Security Bytecode Instructions

To save memory in resource constrained devices like smart cards, Java Card bytecode run by the JCVM uses an encoding optimized for size. As a design tradeoff, Java Card bytecodes are coded on one byte. Nevertheless, only 186 bytecodes are standardized in the context of the JCVM [15], thus leaving free 70 possible proprietary bytecodes.

***Bourbon Vanilla*: Inhibiting Instructions.** Our first proposition is to take advantage of the unused bytecodes by completing the standard instruction set with some security-specific instructions. These new security-specific bytecodes will be recognized by the JCVM and not executed (i.e. inhibited) while no attack has been detected.

On the other hand, once an attack is detected, the execution of these bytecodes will be disinhibited and the extra security will be enabled for the next executions. Those bytecodes (the `inhib_*` below) would implement classical countermeasures, such as:

- executing a software desynchronization function (*e.g.* `inhib_desynchro`).
- verifying the integrity of the currently executing application (*e.g.* `inhib_appcrc`).
- verifying the types of the current local variables (*e.g.* `inhib_typesafe`).
- redundant check of a previous `if*` instruction (*e.g.* `inhib_if*red`).

Based on unused opcodes and classical countermeasures previously described, one is now able to fill up the Java Card instruction set. Table 2 gives one example of this concept.

Table 2. Filling up the instruction set

Bytecodes	0x00	0x01	...	0xB8	0xB9	...	0xBC	...	0xFE	0xFF
Instructions	`nop`	`aconst_null`	...	`putfield_i_this`	`inhib_desynchro`	...	`inhib_typesafe`	...	`impdep1`	`impdep2`

The next step consists in adding those new bytecodes in the code of an applet. One way to achieve this is to perform a post processing on the .CLASS file from custom rules that add security bytecodes when needed. For instance, each time the function `OwnerPIN.check` is invoked, the `inhib_desynchro` bytecode could be added just before the invocation. Listings 5 and 6 show how this can be applied to the if statement used so far as example.

Such insertion, as well as the numerous modifications on either the .CLASS or .CAP file it implies can be easily achieved by using public tools such as BCEL [16], or CapMap [17].

Listing 5. Initial source code

```
// assume b is a boolean.
if ( !b ) {
  // e-wallet debit
}
else if ( b ) {
  // e-wallet credit
}
else {
  ISOException.throwIt(
     ATT_DET_SW);
}
```

Listing 6. *Bourbon Vanilla*-secured bytecode

```
    iload_1
    ifne L1
    // e-wallet debit
    goto L2
L1: inhib_desynchro
    iload_1
    ifeq L3
    inhib_ifeq_red L3
    // e-wallet credit
    goto L2
L3: bipush 18 <ATT_DET_SW>
    invokestatic #6 <throwIt>
L2: ...
```

From now on, by using a dedicated flag of a state machine that indicates whether an attack has been detected or not, the JCVM can enable or disable the security-specific instructions freshly added in the instruction set. Listing 7 describes a way to achieve this goal by modifying the interpreter routine.

This proposition is based on a tradeoff between reaching a high-level of security without impacting the performances when it is not necessary. The solution presented above can be fully automated which implies that no human intervention is required in the process of applet protection.

Still exploiting the concept of inhibiting instructions, we propose another approach which involves the developer in the process of applet protection. The next section introduces such a concept and develops how it could be deployed.

***Tahitian Vanilla*: Inhibiting Sequences of Instructions.** The idea developed above is based on dedicated bytecodes that, when they will be interpreted by the JCVM, will trigger specific countermeasures implemented by the platform. Another embodiment of the previous concept is to leave the choice to the developer to inhibit certain portions of his code, in order to increase the applet

Listing 7. Inhibition of security-specific bytecodes

```
// The instruction is read from memory
instruction = fetch();

// Test Inhibiting or not the instruction
if ( isSecuritySpecific( instruction ) ) {
    if ( !flagAttack ) {
        if ( flagAttack ) {
            // Fault detected
        }
        // No attack, no execution of security instruction
    }
    else {
        if ( !flagAttack ) {
            // Fault detected
        }
        // An attack has been detected
        execute( decode(instruction) );
    }
}
else {
    // Protection against fault attacks
    if ( !isSecuritySpecific( instruction ) ) {
        // No attack has been detected
        execute( decode(instruction) );
    }
}
```

security when needed. So, one solution is that the JCVM conditionally triggers the execution of bytecode instructions. This mechanism can be implemented in two stages.

First, specific methods in the applet code delimit the part corresponding to the extra security added by the developer (e.g. `Protection.begin()` and `Protection.end()`) as depicted in Listing 9. These two methods are static and are defined *via* a proprietary API which implementation allow to activate/inhibit the instructions living between the static methods markups. In order to perform the activation/inhibition, again the interpreter routine must be modified as exposed in Listing 8 for instance.

Finally, two scenarii are possible:

- no attack has been detected and the JCVM does not execute the instructions comprised between the `invokestatic #X` and `invokestatic #Y` bytecodes where `#X` (resp. `#Y`) corresponds to the method that enable (resp. disable) the security (see Listing 10).

Listing 8. Inhibition of blocks of bytecodes

```
// The instruction is read from memory
instruction = fetch();

// Test Inhibiting or not the instruction
if ( openMarkup && !isCloseMarkup(instruction) ) {
    if ( !flagAttack ) {
        if ( flagAttack ) {
            // Fault Attack detected
        }
        // No attack, no execution of security instruction
        // except to close markup
    }
    else {
        if ( !flagAttack ) {
            // Fault detected
        }
        // An attack has been detected
        execute( decode(instruction) );
    }
}
else {
    // Protection against fault attack
    if ( !openMarkup || isCloseMarkup(instruction) ) {
        // No attack has been detected
        execute( decode(instruction) );
    }
}
```

– an attack has been detected and the JCVM executes all the bytecodes corresponding to the extra security added between the `invokestatic #X` and `invokestatic #Y` bytecodes.

Discussion. Ensuring that an applet can trigger tuned levels of security countermeasures only when specific threats are detected can be done via an enrichment of the language recognized by the JCVM. Our first proposition consists in defining new bytecodes in the JCVM. Thus, those security bytecodes can be added anywhere in the .CLASS file and are not executed while a specific flag is not raised. Our second proposition makes use of static methods as markups, to inhibit the non-crucial countermeasures until an attack is detected.

Table 3 presents the overhead for both propositions in terms of memory footprint and execution time when no attack has been detected compared to the traditional way of securing an applet. As exposed in Listings 7 and 8, an additional `if` is executed within the interpreter routine in order to determine whether an instruction should be first decoded and then executed or not.

Such instructions, at the native level, do not take more than a couple of cycles to be executed. Therefore their impact is limited, although it should not be

Listing 9. Source code with security markups

```
// assume b is a boolean.
if (!b) {
  // e-wallet debit
}
else {
  if (b) {
    Protection.begin();
    if ( !b ) {
      ISOException.throwIt(
         ATT_DET_SW);
    }
    else if (b) {
      Protection.end();
      //e-wallet credit
    }
    else {
      ISOException.throwIt(
         ATT_DET_SW);
    }
  }
  else {
    ISOException.throwIt(
       ATT_DET_SW);
  }
}
```

Listing 10. Bytecode with security markups

```
     iload_1
     ifne L1
     // e-wallet debit
     goto L2
L1:  iload_1
     ifeq L3
     invokestatic #7 <begin>
     iload_1
     ifne L4
     bipush 18 <ATT_DET_SW>
     invokestatic #6 <throwIt>
     goto L2
L4:  iload_1
     ifeq L6
     invokestatic #8 <end>
     // e-wallet credit
     goto L2
L6:  bipush 18 <ATT_DET_SW>
     invokestatic #6 <throwIt>
     goto L2
L3:  bipush 18 <ATT_DET_SW>
     invokestatic #6 <throwIt>
L2:  ...
```

completely omitted. Consequently, the execution time of an instruction, expressed as $t_{ins} = t_{fetch} + t_{decode} + t_{execute}$ becomes $t'_{ins} = t_{fetch} + 2 \cdot t_{if} + t_{decode} + t_{execute}$. Subsequently, we denote by n_{cred} the number of instructions for the credit operation.

Both these solutions afford a better time/security tradeoff than the common countermeasure. However, we observe that the *Bourbon Vanilla* implementation also allows to reduce the size of the applet. This is due to the fact that the security mechanisms are deported within the JCVM in this case. On the other hand, *Tahitian Vanilla* is equivalent to the traditional method in terms of memory footprint, only the markups being added.

However, these two solutions require modifications of both the applet and the JCVM. The main impact of this approach is that the applet loses its portability.

In the following we propose another approach where the applet is not modified to keep its portability feature.

3.3 *Strawberry*: Inhibiting Secured Bytecode Implementations

The main design goals of the Java Card technology are portability and security. Nevertheless, the protection mechanism presented in Section 3.2 is at the cost of the portability: the “write once, run everywhere” principle does not hold anymore.

On the other hand, one does not want a security mechanism that impacts performances (in term of execution time or code size) when the applet is not under attack.

Table 3. Compared size and execution time overhead of the 2^{nd}-order-secured and *vanilla*-secured `if`

Mnemonic	Listing	Size (byte)	Timing
2^{nd}-order-secured	4	30	$6 \cdot t_{ins}$
Bourbon Vanilla	6	13	$2 \cdot t_{if} \cdot (5 + n_{cred})$ $+2 \cdot (t_{fetch} + 2 \cdot t_{if})$
Tahitian Vanilla	10	36	$2 \cdot (3 + n_{cred}) \cdot t_{if}$ $+4 \cdot t_{ins'} + 7 \cdot (t_{fetch} + 2 \cdot t_{if})$

To circumvent this issue and keep the portability feature of an applet, an approach consists in different interpretations of a bytecode depending on the execution context (e.g. nominal or under attack). For instance one can implement two different versions of bytecodes: a secure and a non-secure. Thus, by default the JCVM executes the non-secure versions of the bytecodes while when an attack is detected, the secure implementation is executed (see Listing 11).

As FI on Java Card mainly focuses on disturbing conditional branchings and methods execution, it follows that not each and every bytecode in the applet needs to be protected. So, it is sufficient to apply this principle only on bytecodes

Listing 11. Inhibited secure bytecodes implementation

```
// The instruction is read from memory
instruction = fetch();

// Switch between non-secure or secure implementation
if ( flagAttack ) {
    // Execution of the secure implementation
    execute( decodeSecure(instruction) );
}
else {
    // Protection against fault attacks
    if ( !flagAttack ) {
        execute( decode(instruction) );
    }
    else {
        // Fault Attack detected
    }
}
```

that are sensitive to fault attacks (e.g. `ifeq`, `ifne`, `sipush`, *etc.*) and on sensitive API methods (e.g. `OwnerPIN.check`, `JCSystem.arrayCompare`, *etc.*).

A secured `ifeq` implementation should, for instance, ensure the integrity of the value read from the operand stack as well as that of the branch taken. The integrity of the value pushed onto the operand stack is liable to a secured implementation of the `sipush` instruction or of the `arrayCompare` API for instance.

The if-then-else statement could then be written straightforwardly, as in Listing 1, the security being dynamically ensured by the JCVM.

Discussion. Although coding two versions of bytecodes is a costly process that increases the JCVM's size, it nevertheless has several advantages.

Firstly, unlike the methods detailed in Section 3.2, this approach does not need a modification of the applet. It follows that the applet does not break off the portability paradigm and can be deployed on any standard JCVM.

Secondly, coding an applet while adding security requires a lot of experience to obtain a good tradeoff between execution time, code size and security. So, using that method, even a developer who is not familiar with the concept of FI can develop an applet on a product that could be evaluated and certified.

Moreover, with that approach, there is no need to involve a third party to perform a security proofreading of the applet.

Table 4 presents the additional cost for the *Strawberry* proposition in terms of memory footprint and execution time when no attack has been detected. As for the Vanilla methods, the execution time estimations take into account the additional `if` added within the interpreter routine which is the only overhead for this method, as exposed in Listing 11.

Table 4. Compared size and execution time overhead of the 2^{nd}-order-secured and *Strawberry*-secured if

Mnemonic	Listing	Size (byte)	Timing
2^{nd}-order-secured	4	30	$6 \cdot t_{ins}$
Strawberry	2	0	$(2 \cdot t_{if}) \cdot (3 + n_{cred})$

Anyway, all the solutions described in Sections 3.2 and 3.3 are not mutually exclusive and can be used in a very flexible way. Indeed, depending on the context, one can adopt a hybrid approach by combining the previous solutions leading to a greater security for the product.

4 Conclusion

In this paper we presented a new approach to protect Java Card platform against Fault Injection. This new methodology allows sensitive applications to run much faster in every day life while providing a very high security level against fault attacks. By modifying the JCVM, we showed how such a concept can be implemented in practice in two different ways. The first one consists in adding dedicated bytecodes which provides specific security features such as redundancy or desynchronization. These bytecodes will be inhibited until a threat is detected. The second solution consists in implementing some bytecodes twice: one implementation being the standard one and the second implementation including advanced fault countermeasures. The JCVM switches from the first implementation to the second one as soon as an attack is detected. We also compared the advantages and drawbacks of each solution in order to provide all useful information to Java Card developers allowing them to choose the best possible solution depending on their context.

This new methodology is definitely more pragmatic than the traditional approach as it allows Java Card applications to fulfill performances requirements more easily while successfully counteracting advanced fault injection attacks.

Acknowledgments. The authors would like to thank Emmanuelle Dottax for her helpful comments on the preliminary version of this paper.

References

1. Kocher, P.: Timing Attacks on Implementations of Diffie-Hellman, RSA, DSS, and Other Systems. In: Koblitz, N. (ed.) CRYPTO 1996. LNCS, vol. 1109, pp. 104–113. Springer, Heidelberg (1996)
2. Kocher, P., Jaffe, J., Jun, B.: Differential Power Analysis. In: Wiener, M. (ed.) CRYPTO 1999. LNCS, vol. 1666, pp. 388–397. Springer, Heidelberg (1999)
3. Quisquater, J.J., Samyde, D.: A New Tool for Non-intrusive Analysis of Smart Cards Based on Electro-magnetic Emissions, the SEMA and DEMA Methods. Presented during EUROCRYPT 2000 Rump Session (2000)

4. Bellcore: New Threat Model Breaks Crypto Codes. Press Release (1996)
5. du Castel, B.: Personal History of the Java Card (2012), French version originally published in MISC magazine, HS-2 (November 2008)
6. Joye, M., Tunstall, M.: Fault Analysis in Cryptography. Information Security and Cryptography. Springer (2012)
7. Bar-El, H., Choukri, H., Naccache, D., Tunstall, M., Whelan, C.: The Sorcerer's Apprentice Guide to Fault Attacks. IEEE 94, 370–382 (2006)
8. Giraud, C., Thiebeauld, H.: A Survey on Fault Attacks. In: Quisquater, J.J., Paradinas, P., Deswarte, Y., Kalam, A.E. (eds.) Smart Card Research and Advanced Applications VI – CARDIS 2004, pp. 159–176. Kluwer Academic Publishers (2004)
9. Mostowski, W., Poll, E.: Malicious Code on Java Card Smartcards: Attacks and Countermeasures. In: Grimaud, G., Standaert, F.-X. (eds.) CARDIS 2008. LNCS, vol. 5189, pp. 1–16. Springer, Heidelberg (2008)
10. Barbu, G., Duc, G., Hoogvorst, P.: Java Card Operand Stack: Fault Attacks, Combined Attacks and Countermeasures. In: Prouff, E. (ed.) CARDIS 2011. LNCS, vol. 7079, pp. 297–313. Springer, Heidelberg (2011)
11. Common Criteria: Application of Attack Potential to Smartcards (2009)
12. Barbu, G., Thiebeauld, H., Guerin, V.: Attacks on Java Card 3.0 Combining Fault and Logical Attacks. In: Gollmann, D., Lanet, J.-L., Iguchi-Cartigny, J. (eds.) CARDIS 2010. LNCS, vol. 6035, pp. 148–163. Springer, Heidelberg (2010)
13. Vetillard, E., Ferrari, A.: Combined Attacks and Countermeasures. In: Gollmann, D., Lanet, J.-L., Iguchi-Cartigny, J. (eds.) CARDIS 2010. LNCS, vol. 6035, pp. 133–147. Springer, Heidelberg (2010)
14. Bouffard, G., Iguchi-Cartigny, J., Lanet, J.-L.: Combined Software and Hardware Attacks on the Java Card Control Flow. In: Prouff, E. (ed.) CARDIS 2011. LNCS, vol. 7079, pp. 283–296. Springer, Heidelberg (2011)
15. Sun Microsystems Inc.: Virtual Machine Specification – Java Card Plateform, Version 3.0.1 (2009)
16. The Apache Software Foundation: (Apache Commons BCEL, The Byte Code Engineering Library), `http://commons.apache.org/bcel/`
17. Smart Secure Devices (SSD) Team – XLIM, Université de Limoges: CapMap – The CAP file manipulator, `http://secinfo.msi.unilim.fr`

Java Card Combined Attacks with Localization-Agnostic Fault Injection

Julien Lancia

SERMA Technologies, CESTI, 30, avenue Gustave Eiffel
33600 Pessac, France
j.lancia@serma.com

Abstract. In this paper, we present a paradigm for combined attacks on Java Cards that lowers the requirements on the localization precision of the fault injection. The attack relies on educated objects allocation to create favorable memory patterns that raise the chances of success of the combined attack. In order to maximize the probability of successful injection, we determine the optimal parameters depending on the physical properties of the targeted platform. Finally, we demonstrate the efficiency of our approach through fault injection simulation.

1 Introduction

Security in the smart card field is a main issue, due to the very nature of the data stored on chips. Up to now, several attack paths have been explored to compromise the security of these platforms [6,10,13,16,1,11], leading to new counter measures designed to prevent these attacks from succeeding [17,5,19,12]. In order to strengthen the security of the embedded systems, the Java Card approach is to rely on a strong-typed language and to factorize the security verifications in the Java Card Virtual Machine (JCVM) abstraction layer. The JCVM provides several security services to the application layer: the firewall mechanism enforces applets isolation, the JCVM checks for stack- and buffer-overflow in order to prevent illegal memory accesses, and finally the bytecode verifier (BCV) is responsible for guaranteeing the type safety of the executed programs.

Up to the 2.2.2 version of the Java Card specifications [23,24,22,25], the bytecode verification is performed off-card, and is therefore optional. Many attacks on the Java Card platforms [18,9,8] are based on manipulations of the binary representation of the applet (the cap file) aiming a circumvention of the type system of the virtual machine. Such attacks, called "logical attacks" have indeed huge impacts on the overall security of the platform, but are irrelevant on industrial processes where bytecode verification is mandatory. Moreover, in the last version of the Java Card specifications (3.0 connected edition [26,27,28]), the bytecode verifier is embedded on card, making the bytecode verification process mandatory.

Recently, a new paradigm of attacks called "combined attacks" emerged [3,29,4]. These attacks combine logical attacks and physical attacks to bypass the protection mechanisms of the Java Card platform. Physical attacks on smart cards usually rely on laser beams[20], which have to be precisely tuned both in timing and

S. Mangard (Ed.): CARDIS 2012, LNCS 7771, pp. 31–45, 2013.

localization for the combined attack to succeed. Therefore time and geographic localizations are strong constraints on the success rate of the combined attacks.

In this paper, we propose a paradigm for combined attacks on smart cards that allows an attacker to evade localization constraints of the hardware attack. This paradigm, inspired from an attack aiming classical Java Virtual Machines (JVM) [15], is based on specific memory patterns that are obtained through objects allocations by a malicious applet. These objects allocations, although legal regarding the Java Card specifications and bytecode verifier, end up with a particular memory pattern that raises the chances of success of the fault injection, regardless of the localization of the laser pulse.

The rest of this paper is structured as follows. In Section 2, we briefly present the concepts of the original attack on a JVM, we explain why this attack can't succeed on modern JVCM implementations, and we expose our work to exploit the original concept as part of a combined Java Card attack. In Section 3, we present our experimental results. In Section 4, we discuss how to prevent such attacks. In Section 5 we compare our work to existing approaches and conclude on the contribution this work brings to the Java Card attacks field.

2 The Attack Concept

2.1 Exploitation of Memory Errors on a Java Virtual Machine

Although using physical fault injection on a system's memory to evade software security mechanisms has been practiced for a while now on smart cards, the idea to induce memory errors likely in a PC's SRAM through physical fault injection has emerged recently. In [7], Govindavajhala and Appel present a concept of attack that allows arbitrary memory error to produce type confusion in a classical Virtual Machine.

Their attack applet declares the two classes A and B presented in Figure 1. The applet fills the heap with many instances of class *B* and one instance of class *A* named *a*. All the fields of all the *B* instances are initialized to point to the unique *A* instance. Figure 2 represents an excerpt of the memory after object initialization.

The algorithm of the applet loops through all *A* fields of all the objects in memory and checks through Java pointer equality whether they still contain the address of the A instance (noted x). In case an error switches any *A* field's value (let's say b31.a2), the original algorithm of the applet presented in Listing1.1 mutates into a two steps type confusion on the erroneous field reference.

Listing 1.1. Two steps type confusion on an erroneous field reference

```
A aObj; B b31; B fakeB;
aObj = b31.a2;      // type confusion, step 1
fakeB = aObj.b;     // type confusion, step 2
```

```
class A {                    class B {
  A a1;                        A a1;
  B b;                         A a2;
  int addr;                    A a3;
  A a4;                        A a4;
};                           };
```

Fig. 1. Classes of the original applet

Figure 2 illustrates the memory operations performed by the virtual machine during these two steps, with and without a fault injection. In step 1, the applet dereferences the b31.a2 field in aObj. The memory error changes the content of the aObj field, that normally contains the address x of the A instance, into a new value $x'=x+\Delta x$, . In step 2, the applet dereferences the B field aObj.b. Because the memory error induced a Δx bias of the aObj value, the resulting reference is likely to contain an *A* reference as most of the memory is filled with fields of type *A*. This produces a type confusion as the *fakeB* reference, that is statically typed as a *B* object, references an object of type *A*.

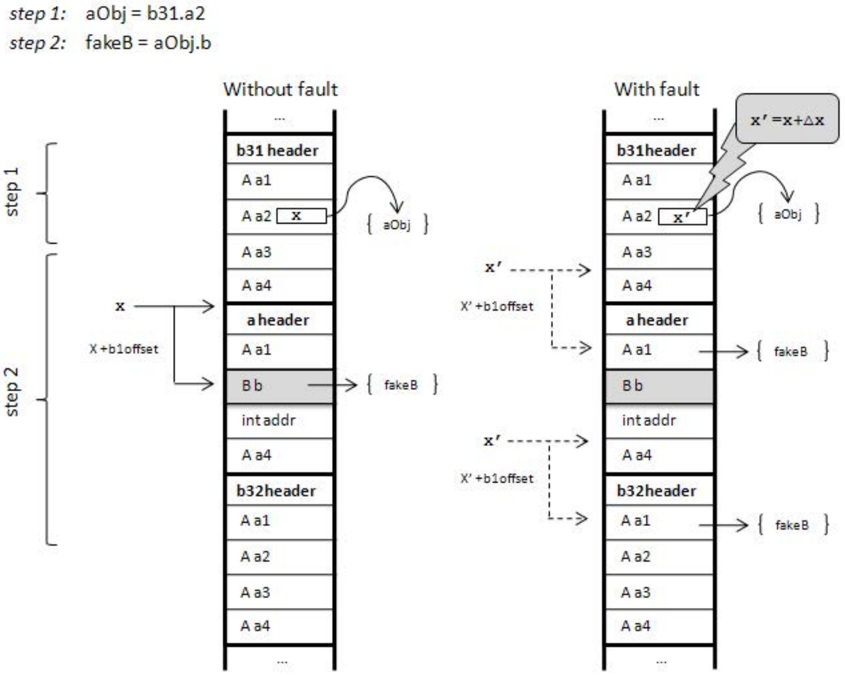


Fig. 2. Realization of the type confusion on a Java Virtual Machine

The exploitation of this confusion is very straightforward: the *addr* field of the unique *A* instance is set to forge an *A* object reference at the address *custom_ addr*ess, accessible through fakeB.a3 (as *addr* is the third field of *A*). The field fakeB.a3.addr sits in memory at the address *custom_ address+a3offset*. All the addressable memory is therefore accessible for reading and writing through this field, with a constant memory offset equal to *a3offset*. The exploitation of the type confusion is synthesized in Figure 3.

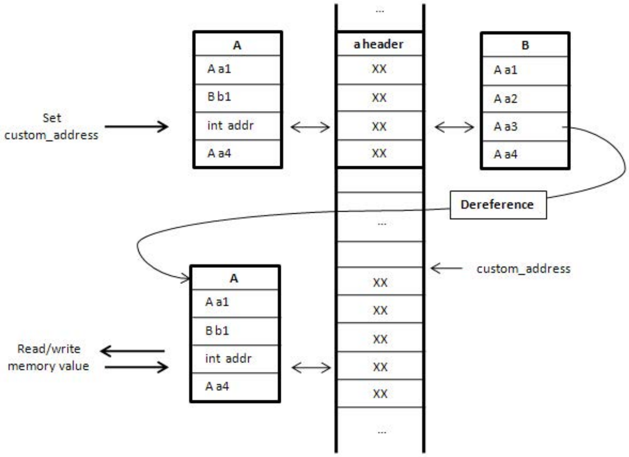


Fig. 3. Exploitation of the type confusion on a Java Virtual Machine

2.2 Specificity of Modern Java Card Virtual Machines

The attack presented in the previous subsection is dedicated to classical Java virtual machines. It could not be led identically on Java Card virtual machines because the embedded platforms they run on have very specific hardware constraints, that impacts the memory layout of the virtual machine instances and the physical properties of the memory where these instances are stored.

More precisely, Java virtual machines run on standard computer architectures, where object instances are allocated in RAM memory. Contrarily, in Java Card virtual machines, instances are allocated in persistent memory (i.e. flash or EEPROM).[1]

The physical properties of the persistent memories make it much more difficult for an attacker to realize a bit flip through external means [21]. In opposite to

[1] Java Card platform provides methods to create temporary (transient) arrays whose content is stored in RAM, but their headers are still in persistent memory. Only the objects stored in transient object arrays would have their headers in RAM memory. These objects are mandatorily of type Object, and are consequently not subject to type confusion.

the attack presented in the previous section where the faults in RAM memory are triggered using the heat of a portable light, errors in persistent memory generally require laser or electromagnetic injections [2,20]. Moreover, because of the scarcity of the resources in these embedded platforms, the amount of persistent memory available for object instantiation is quite limited. The memory layout of the objects used for the attack must therefore be optimized in order to maximize the chances of success.

Another main difference between standard Java virtual machines and Java Card virtual machines is the memory management. In standard Java virtual machines, references are represented using a direct memory addressing, which means that the physical memory address of the referenced instances are stored in memory like traditional pointers. In modern implementations of Java Card virtual machines, references are represented using an indirect memory addressing, where references are represented as an index in a global instance pool managed by the virtual machine. This difference between the Java and Java Card virtual machines addressing mode is illustrated in Figure 4. Because the indirect memory representation of the instances is decorrelated from the physical address of the instance, the attack presented in previous section can not apply on Java Card virtual machines and must therefore be adapted for these platforms.

The use of indirect addressing mode in Java Card virtual machine has another impact that concerns the exploitation of type confusion. A type confusion between a short field and an instance field in a direct memory addressing mode can easily be exploited by using the value of the short field as a pointer address to scan the whole memory. In case of an indirect memory addressing mode, the same type confusion gives access to an index in the global instance pool of the virtual machine. Therefore, the exploitation of the type confusion to scan the memory requires several additional operations. These operations are detailed further in Section 2.3.

2.3 Combined Attack on a Java Card Virtual Machine

Realization of the Attack. Combined attacks aim at taking benefit from hardware and logical attacks to create applets that are considered legitimate by the Java Card virtual machine and the verifier, and whose attack load is activated through fault injection. Our attack maximizes the chances of success of the fault injection by creating favorable pattern in persistent memory through instances allocation. As a result, almost any bit switch in persistent memory creates an exploitable type confusion that is detected and signaled by the applet. This combined attack is an adaptation of the attack presented in Section 2.1 to the specificity of the embedded Java Card virtual machine highlighted in section 2.2.

The preparation of the attack is almost similar. Firstly, the attack applet declares the two classes A and B presented in Figure 5. Secondly, the permanent memory is filled with many instances of class B and a single instance of class A. All the fields of all the B instances are initialized to point to the unique A instance. Figure 6 represents the persistent memory state and the instance pool after object initialization.

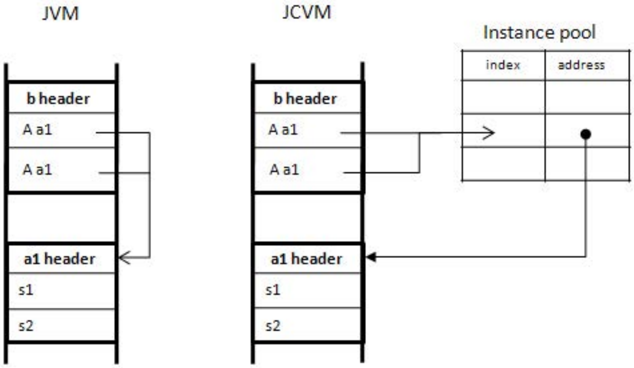


Fig. 4. Java and Java Card virtual machines addressing mode

```
class A {                class B {
short s1;                  A a1;
short s2;                  A a2;
short s3;                  A a3;
short s4;                  A a4;
};                       };
```

Fig. 5. Classes of our applet

The algorithm of the applet performs the following operations. The applet loops through all *A* fields of all the objects in memory and checks whether they still contain a reference to the unique A instance. When a perturbation (i.e. light fault injection) flips a bit in persistent memory, it will likely change the internal reference of a A field as the main part of the memory is filled with A field. These internal references are indexes in the instance pool of the Java Card virtual machine. Because all instances created by the applet are of type B except the only A instance, the dereference of this erroneous index will return an instance of class B provided it is inside the instance pool boundaries.

In opposite to the original attack, the type confusion is performed in a single step (see Listing 1.2). When the attack success is detected through the Java pointer equality test (let's say on the b31.a2 field), the resulting dereferencing produces a type confusion because the reference aObj is statically typed as an object of type A, whereas it references an object of type B.

Listing 1.2. Single step type confusion on an erroneous field reference

```
A aObj; B b31;
aObj = b31.a2;      // type confusion
```

It should be noted that the attack succeed when the fault injection flips any bit of the instances allocated by our applet, except the B instances headers and the only A instance. In consequence, the number of instances allocated by the applet and the ratio between the object headers and the object fields have a strong impact on the success rate of the attack. This aspect will be discussed further in Section 3.

When the Java pointer equality test between an A field and the unique A instance fails, the type confusion is successful. The applet exits the main loop and outputs a specific status word to indicate the attack success. The type confusion can then be exploited through the erroneous A field.

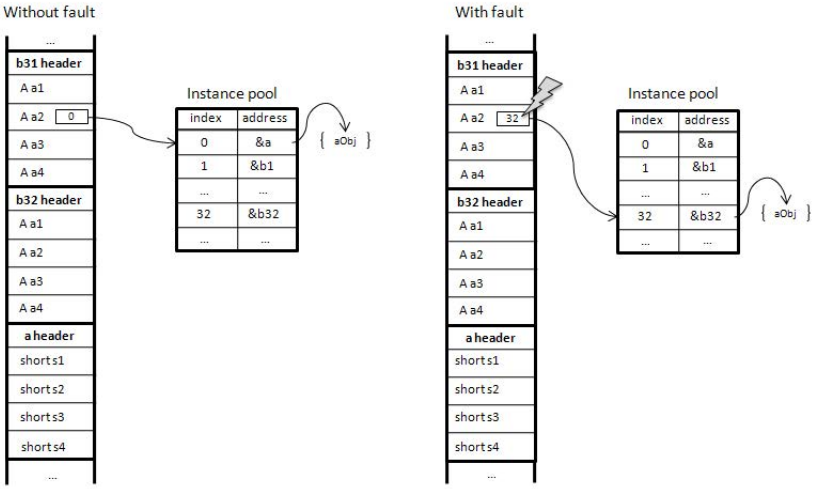


Fig. 6. Realization of type confusion on a Java Card virtual machine

Exploitation of the Type Confusion. The combined attack presented so far produces a reference of type A (aObj) that references an instance of type B (b32 in our previous example). The exploitation of the type confusion consists in accessing the fields of the B instance through the aObj reference and through the b32 reference.

However, when the fault injection switches the A field index, the new index can reference any B instance on the card. It is therefore necessary to identify the B instance that is referenced by the aObj reference. The easier way to obtain this information would be to code the index of the B instance in a field of the B object, but this approach has a major drawback. The bytes used to code the index in B instances would not produce a type confusion if switched through fault injection, thus lowering the chances of success of our attack.

The exploitation of the type confusion can be realized without impacting the attack success rate by realizing a second, indirect type confusion. After the first type confusion has been successfully realized, the aObj variable references the same object as the b32 variable which means that aObj.s1 and b32.a1 fields reference the same memory slot. In order to identify the B object that is referenced by the aObj variable, we set the aObj.s1 field to an arbitrary value, which produces a new type confusion as the equivalent b32.a1 field doesn't point to the unique A instance anymore. The applet then loops through all *A* fields except the one referenced by aObj and finds through Java pointer equality the one that differs from the unique A instance. This *A* field belongs to the B instance we can use to exploit our type confusion (b32 in our previous example).

Once the both references of type A and of type B that reference the same instance have been identified, the type confusion can be exploited to provide illegal access to the card memory. The exploitation of the type confusion is synthesized in Figure 7. The aObj.s1 field is set to forge an index of the Virtual machine instance pool. This index is dereferenced as an instance of type A through the equivalent b32.a1 field. The short fields of this instance (b32.a1.s1 to b32.a1.s4) can then be accessed in reading and writing to dump and modify arbitrary memory slots.

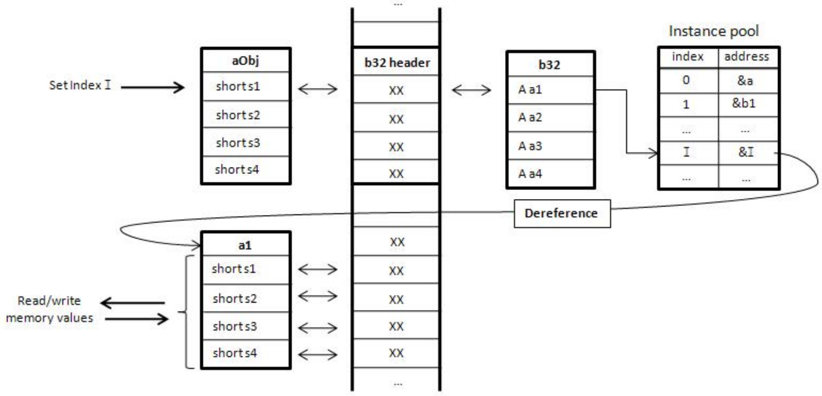


Fig. 7. Exploitation of type confusion on a Java Card virtual machine

3 The Practical Attack

3.1 Optimal Parameters

The goal of our attack is ultimately to raise the chances of success of the fault injection part of a combined attack by evading the localization constraints of the physical attack. This is made possible through a specific allocation of objects that creates a favorable pattern in persistent memory. Therefore, as we've seen in the section 2.3, the design of the classes have a strong impact on the success rate of the attack. The optimal ratio is not trivial as two orthogonal factors must be taken into account :

- The number of instances, that produce overhead through their headers,
- The number of fields, that produce overhead through the applet code necessary to test Java pointer equality.

In order to exploit the full possibilities of this attack, we determine the optimal ratio between these two factors. In the remainder of this paper, we use the following notation :

- A (constant) : allocated persistent memory space before applet loading, in bytes,
- B (constant) : size of the bytecode necessary to test a single reference through Java pointer equality, in bytes,
- C (constant) : size of the applet, except the bytecode necessary to test the references, in bytes,
- D (constant) : total size of the persistent memory, in bytes,
- x : number of instances
- y : number of references per instance
- xy : total number of references in persistent memory

Using these notations, we define the probability of hitting a reference with a random fault injection as the ratio between the size of references in memory and the total size of memory (provided references are coded on two bytes):

$$p(hit) = \frac{2xy}{D} \tag{1}$$

In addition, provided instance headers are coded on two bytes, we can decompose the persistent memory as follows:

$$D = 2xy + A + C + By + 2x \tag{2}$$

As a result, we can express the number of instances in function of the number of references per instance:

$$y = \frac{D - A - C - 2x}{2x + B} \tag{3}$$

This gives us the probability of hitting a reference with a fault injection in function of the number of instances:

$$p(hit) = \frac{2x(D - A - C - 2x)}{(2x + B)D} \tag{4}$$

We apply our approach on a chip with 80 kBytes of flash persistent memory. The size of our applet is 5005 bytes except the reference test bytecode, that represents 17 bytes per reference. The allocated persistent memory before we load the applet is 9567 bytes. We compute the optimal ratio between the number of instances and the he number of fields using the following parameters :

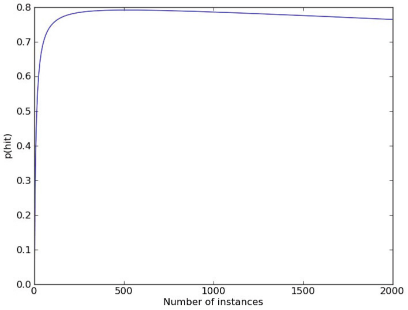


Fig. 8. Probability of hitting a reference with a fault injection in function of the number of instances

- A = 9567 bytes
- B = 17 bytes
- C = 5005 bytes
- D = 80 kBytes

The resulting probability of hit is shown on figure 8. This function is maximal for 519 instances. Using the formula 3, we determine that each instance should have 61 reference fields. As a result, we obtain a probability of hitting a reference with a random fault injection of 79%.

$$p(hit) = 0.79$$

3.2 Attack Simulation

In order to validate our approach, we have built a fault simulator as a plugin on the reference implementation provided by Oracle. Our plugin generates errors in the reference implementation's representation of the persistent memory and monitors the resulting behaviour of the applet. After the behaviour caused by the fault injection has been analyzed, the persistent memory is restored and another fault is generated. We consider only byte-fault models for the fault injection simulation, and we generate faults according to two different models :

- Byte modification fault model : the byte affected by the fault is replaced by an arbitrary hexadecimal value,
- Stuck-at fault model : the byte affected by the fault is replaced by either 0x00 or 0xFF hexadecimal value.

In addition, the fault simulator supports two fault localization model:

- Random localization model : the location of the fault is chosen randomly ,
- Scanning localization model : each byte of the memory is faulted one after the other.

We perform our attack simulation using the byte modification fault model. This fault model produces a superset of the faults generated by a bit-flip fault model which is representative of the effects of a fault injection on the persistent memory during a read operation. The Figure 9 shows maps of the smartcard's non-volatile memory where successful injections are represented as grey dots. The Figure on the left presents the simulation of an 8000 laser pulses campaign using a random localization model and a byte modification fault model, while the Figure on the right presents the simulation of a 64000 laser pulses campaign using a scanning localization model and a byte modification fault model.

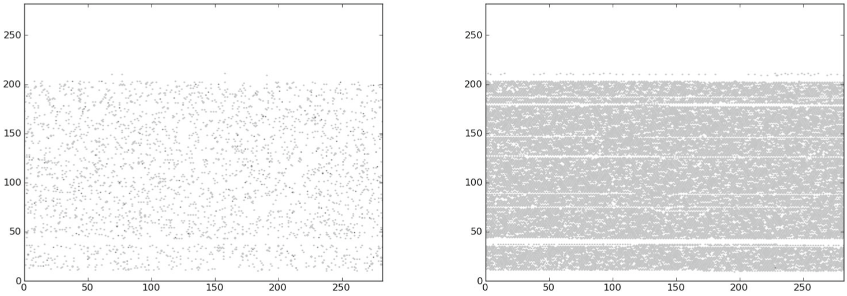


Fig. 9. EEPROM map of successful injections using a random localization model for an 8000 laser pulses campaign (left) and a scanning localization model for a 64000 laser pulses campaign (right)

The results obtained by simulation show that a large part of the non volatile memory produces a type confusion when exposed to external perturbation. The part of the non-volatile memory that does not produces type confusions is mainly filled with the applet's code (top white part of the graphic) and the virtual machine's persistent objects headers (white linear patterns). We can conclude that the presented attack fully aims its objective as most of the fault injections produce a type confusion, regardless of the localization of the laser pulse in the non-volatile memory.

4 Countermeasures

4.1 Defensive Virtual Machines

The Java Card specifications ensure that well-typed code executes identically on every implementation of the virtual machine. However, when it comes to ill-typed code, the specifications leave a lot of freedom concerning the implementation of dynamic checkings. Because ill-typed code is out of the scope of the Java Card specifications, some virtual machines are much more defensive than others, and the behavior in presence of ill-typed code can vary a lot. An overview of possible dynamic runtime checks is provided in [18]. Among the dynamic runtime checks implemented in Java Card virtual machines, the firewall checks can provide a defense against type confusion exploitation. These checks are enforced during object dereferencing and prevent access to objects belonging to other contexts.

However, our experiments with ill-typed code on modern virtual machines have shown that our attack can nevertheless grant illegal memory access. Indeed, firewall context checks are usually performed on metadata stored in the object's header. When our fault injection succeeds, the header of an A instance is assigned to an arbitrary memory slot whose data can be interpreted as the header of an instance belonging to our applet or even to the JCRE. In this case, the memory access is granted by the virtual machine.

When the defensive Java Card virtual machine enforces stronger defenses, the type confusion can be exploited in another way. The class A and the class B can be appended an additional field whose only purpose is to exploit the type confusion once the fault injection has succeeded. For example, the modified classes presented in Figure 10 allow the attacker to access the *tabConfusion* byte array as a short array (which have been proven to be extremely harmful, as presented in [14]). The type of this additional field can be chosen by the attacker to defeat the defences implemented in the virtual machine. However, this solution has a major drawback: the bytes used to code this additional field would not produce a type confusion if switched through fault injection. These bytes are only useful for the exploitation of the type confusion, and thus lower the chances of success of the fault injection.

```
class A {                        class B {
short s1;                          A a1;
short s2;                          A a2;
short s3;                          A a3;
short s4;                          A a4;
short[] tabConfusion;              byte[] tabConfusion;
};                               };
```

Fig. 10. Classes of our applet modified to cope with defensive virtual machines

More generally, combined attacks are a mean of embedding malicious code inside a legal applet whose attack load is activated by fault injection. Therefore, our combined attack can not succeed on defensive virtual machines that enforce efficient dynamic countermeasures against ill-typed code execution.

4.2 Memory Protection

Besides virtual machines countermeasures, some secured IC embed hardware and software memory protections that allow detection of memory errors.

These protections include redondant read operations in persistent memory to detect inconsistencies, parity checking and error-correcting codes to detect and correct errors in persitent memory.

When activated, such countermeasures are efficient at detecting and preventing the fault injections described in this paper.

5 Related Works and Conclusion

5.1 Related Works

Our work is an adaptation of the attack presented in [7] to the domain of smart cards. The specificity of the Java Card applets embedded on smart cards have already been presented in section 2.2. This specificity prevents exploitation of the attack originally designed for classical virtual machines on state-of-the-art Java Card virtual machines. In addition to adapt the original attack to the smart card domain, we also prove the efficiency of our approach through fault simulation.

Other publications [29,4] present combined attacks that exploit both malicious applets and fault injection to activate the attack load of the applet on-card, thus luring the bytecode verifier. However, unlike the attack we present in this paper, these attacks demand a high precision in the localization of the fault injection. Inversely, our attack allows to evade the localization constraint of the fault injection when performing the combined attack.

Finally, in [3], Barbu et. al. propose a combined attack that rely on the multithreading capacities of the Java Card 3.0 connected edition platform to interrupt the virtual machine execution and thus evade the timing constraint in the fault injection. Like our attack, their work eases the fault injection part of the combined attack by lowering the precision requirements of the laser pulse. However their attack targets a platform that is barely deployed nowadays, while ours targets the main stream Java Card 2 as well as the Java Card 3 standard edition platforms.

5.2 Conclusion

The emergence of Java Card 3 standard edition platforms with embedded bytecode verifier leads the security community to design new types of attacks that combine both software and physical attacks. These attacks, called combined attacks, inherit the constraints of the classical fault injection attacks: they require a high degree of precision in the timing and the localization of the physical fault injection.

As the smartcard market keeps growing, the security of the integrated circuits is getting more and more efficient. Upscale chips now embed a high range of invasion sensors that prevent attackers to compromise the security of the applications. Therefore, scanning the whole circuit is not an option anymore.

The combined attack paradigm presented in this paper allows to lower drastically the requirements on the localization of the physical fault injection. As a result, the attacker can choose any physical zone that shows less security, without trading the success rate of the combined attack. The resulting type confusion have been proved to be extremely compromising for a large range of nowadays Java Card platforms.

References

1. Agrawal, D., Archambeault, B., Rao, J., Rohatgi, P.: The EM side-channel(s). In: Kaliski Jr., B.S., Koç, Ç.K., Paar, C. (eds.) CHES 2002. LNCS, vol. 2523, pp. 29–45. Springer, Heidelberg (2003)
2. Bar-El, H., Choukri, H., Naccache, D., Tunstall, M., Whelan, C.: The sorcerer's apprentice guide to fault attacks (2004)
3. Barbu, G., Thiebeauld, H., Guerin, V.: Attacks on Java Card 3.0 Combining Fault and Logical Attacks. In: Gollmann, D., Lanet, J.-L., Iguchi-Cartigny, J. (eds.) CARDIS 2010. LNCS, vol. 6035, pp. 148–163. Springer, Heidelberg (2010)
4. Bouffard, G., Iguchi-Cartigny, J., Lanet, J.-L.: Combined software and hardware attacks on the Java Card control flow. In: Prouff, E. (ed.) CARDIS 2011. LNCS, vol. 7079, pp. 283–296. Springer, Heidelberg (2011)
5. Coron, J.-S.: Resistance against differential power analysis for elliptic curve cryptosystems. In: Koç, Ç.K., Paar, C. (eds.) CHES 1999. LNCS, vol. 1717, pp. 292–302. Springer, Heidelberg (1999)
6. Giraud, C., Thiebeauld, H.: A survey on fault attacks. In: Quisquater, J.-J., Paradinas, P., Deswarte, Y., El Kalam, A.A. (eds.) Smart Card Research and Advanced Applications VI. IFIP, vol. 153, pp. 159–176. Springer, Boston (2004)
7. Govindavajhala, S., Appel, A.W.: Using memory errors to attack a virtual machine. In: IEEE Symposium on Security and Privacy, pp. 154–165. IEEE Computer Society (2003)
8. Hogenboom, J., Mostowski, W.: Full memory read attack on a Java Card (2003)
9. Iguchi-Cartigny, J., Lanet, J.L.: Developing a trojan applets in a smart card. Journal in Computer Virology 6, 343–351 (2010)
10. Kim, C.H., Quisquater, J.J.: Faults, injection methods, and fault attacks. IEEE Des. Test 24, 544–545 (2007)
11. Kocher, P., Jaffe, J., Jun, B.: Differential power analysis. In: Wiener, M. (ed.) CRYPTO 1999. LNCS, vol. 1666, pp. 388–397. Springer, Heidelberg (1999)
12. Kömmerling, O., Kuhn, M.G.: Design principles for tamper-resistant smartcard processors. In: Proceedings of the USENIX Workshop on Smartcard Technology, p. 2. USENIX Association, Berkeley (1999)
13. Lancia, J.: Un framework de fuzzing pour cartes à puce: application aux protocoles EMV. In: Symposium sur la Sécurité des Technologies de l'Information et de la Communication, SSTIC, pp. 350–368 (2011)
14. Lancia, J.: Compromission d'une application bancaire JavaCard par attaque logicielle. In: Symposium sur la Sécurité des Technologies de l'Information et de la Communication, SSTIC (2012)
15. Lindholm, T., Yellin, F.: The Java(TM) Virtual Machine Specification, 2nd edn. Prentice Hall PTR (April 1999)
16. Messerges, T.S., Dabbish, E.A., Sloan, R.H.: Examining smart-card security under the threat of power analysis attacks. IEEE Trans. Comput. 51, 541–552 (2002)
17. Moore, S., Anderson, R., Cunningham, P., Mullins, R., Taylor, G.: Improving smart card security using self-timed circuits. In: Technology, Fourth AciD-WG Workshop, Grenoble, ISBN, pp. 211–218 (2002)

18. Mostowski, W., Poll, E.: Malicious code on Java Card smartcards: Attacks and countermeasures. In: Grimaud, G., Standaert, F.-X. (eds.) CARDIS 2008. LNCS, vol. 5189, pp. 1–16. Springer, Heidelberg (2008)
19. Quisquater, J.J., Samyde, D.: Electromagnetic analysis (ema): Measures and counter-measures for smart cards. In: Attali, I., Jensen, T. (eds.) E-smart 2001. LNCS, vol. 2140, pp. 200–210. Springer, Heidelberg (2001)
20. Skorobogatov, S.P., Anderson, R.J.: Optical fault induction attacks. In: Kaliski Jr., B.S., Koç, Ç.K., Paar, C. (eds.) CHES 2002. LNCS, vol. 2523, pp. 2–12. Springer, Heidelberg (2003)
21. Skorobogatov, S.P.: Semi-invasive attacks – a new approach to hardware security analysis. Tech. Rep. 630, University of Cambridge, Computer Laboratory (April 2005), `http://www.cl.cam.ac.uk/techreports/UCAM-CL-TR-630.pdf`
22. Sun Microsystems, Inc., Palo Alto/CA, USA: Java Card Platform Security, technical White Paper (2001)
23. Sun Microsystems, Inc., Palo Alto/CA, USA: Java Card 2.2 Application Programming Interface, API (2002)
24. Sun Microsystems, Inc., Palo Alto/CA, USA: Java Card 2.2 Runtime Environment (JCRE) Specification (2002)
25. Sun Microsystems, Inc., Palo Alto/CA, USA: Java Card 2.2 Virtual Machine Specification (2002)
26. Sun Microsystems, Inc., Palo Alto/CA, USA: Application Programming Interface (API) - Java Card(TM) Platform, Version 3.0.1 (2009)
27. Sun Microsystems, Inc., Palo Alto/CA, USA: Runtime Environment Specification - Java Card(TM) Platform, Version 3.0.1 (2009)
28. Sun Microsystems, Inc., Palo Alto/CA, USA: Virtual Machine Specification - Java Card(TM) Platform, Version 3.0.1 (2009)
29. Vetillard, E., Ferrari, A.: Combined attacks and countermeasures. In: Gollmann, D., Lanet, J.-L., Iguchi-Cartigny, J. (eds.) CARDIS 2010. LNCS, vol. 6035, pp. 133–147. Springer, Heidelberg (2010)

Improved (and Practical) Public-Key Authentication for UHF RFID Tags

Sébastien Canard[1], Loïc Ferreira[2], and Matt Robshaw[2]

[1] Applied Cryptography Group, Orange Labs
42 rue des Coutures BP 6243, 14066 Caen Cedex, France
[2] Applied Cryptography Group, Orange Labs
38-40 rue de General Leclerc, 92794 Issy les Moulineaux, France
{firstname.lastname}@orange.com

Abstract. CRYPTOGPS has been promoted as a public-key technology suitable for UHF RFID tag authentication. Since it is a classical *commitment-challenge-response* (CCR) scheme, it can be converted into a signature scheme using the transformation proposed by Fiat and Shamir. Previously this signature variant has not been considered for RFID, but in this paper we show how to achieve this transformation in a way that yields a compact and efficient scheme. Further, the three-pass CCR scheme is turned into a regular challenge-response scheme with the attendant protocol and implementation improvements. Since we use a block cipher rather than a hash function for the transformation, we justify our approach using results in the *ideal cipher model* and the net result is a variant of CRYPTOGPS that offers asymmetric UHF tag authentication with reduced communication and protocol complexity.

1 Introduction

The terms RFID, Internet of Things, sensor network, and pervasive computing are frequently used to indicate the anticipated widespread deployment of computationally limited devices. The difficulty of providing (reasonable) security on such devices, in a way that makes economic sense, is by now well-established.

The radio-frequency identification (RFID) tag is a particularly interesting case, with billions of tags in deployment, and while it is an over-simplification, two main operating frequencies are of particular interest. The short-range HF tag (13.56 MHz) is used, for instance, in public-transport applications and underpins the area of *Near Field Communication (NFC)*. However, tags that operate over UHF (860-960 MHz) [10] are cheaper, smaller, and can be read at a distance and it is typically these devices that one has in mind when discussing RFID and cryptography (since advanced standardised cryptography on an HF tag is readily available). One particularly interesting application when using cheap UHF tags is that of *product authentication* and there are many proposals to use cheap UHF RFID tags as part of an anti-counterfeiting solution [1,25,27]. Among the different approaches that might be used, it is dynamic cryptographic tag authentication that offers the greatest long-term promise. But since UHF tags are a demanding implementation environment, it is not so straightforward to identify particularly efficient cryptographic algorithms.

S. Mangard (Ed.): CARDIS 2012, LNCS 7771, pp. 46–61, 2013.

2 Cryptography and RFID Tags

The challenging physical constraints posed by RFID tags have been a significant spur to cryptographic research. Perhaps most success has been in the field of symmetric cryptography where we now have a range of block ciphers including PRESENT [4] and a range of stream ciphers [41] that might be suitable for UHF RFID deployment. Over the years some of these may feature in products; indeed some such as PRESENT already appear in ISO standards [23].

The field of asymmetric encryption is less clear. It could be that a symmetric-key solution works well enough, but that the kind of supporting key infrastructure that is required is somewhat at odds with the typical RFID model. Taking the supply chain as an example, millions of tags will be attached to products by a manufacturer with products being distributed to shops and customers worldwide. Ensuring the right key is available to the right reader at the right time is not trivial.

There is therefore considerable interest in any asymmetric solution that might yield a more flexible supporting key infrastructure. Unfortunately, since the typical algorithms from Internet and PC applications are not at all suited to UHF RFID tags, there are not so many alternatives. However, there has been some renewed interest in what are termed *commitment-challenge-response* (CCR) schemes, since these allow lightweight tag authentication. Among them is CRYPTOGPS.

2.1 CRYPTOGPS

The scheme CRYPTOGPS, due to Girault, Poupard, and Stern, is well-established in the cryptographic literature [12,16,33,40]. Several variants feature in ISO/IEC 9798-5 [21] while the most efficient variant, namely that based around elliptic-curves, is undergoing standardisation in ISO/IEC 29192-4 [24] which is devoted to asymmetric lightweight cryptography. Over the years several optimisations have been proposed [15,17] and the performance of the scheme has been studied by implementors [14,28,29,38].

The essential form of CRYPTOGPS using the typical optimisations one might expect to use is given in Figure 1. In implementation papers, the PRNG is typically instantiated using the lightweight block cipher PRESENT [4] in an appropriate mode of use and the most accurate (post-fabrication) implementation figures [38] show that all the on-tag cryptographic components can be implemented in around[1] 2800 GE with a processing time of around 720 cycles.

The small area required for CRYPTOGPS is due to one property and one optimisation. The property is that no modulo arithmetic is used on the tag. All integer computations are regular addition and multiplication. The optimisation comes in the form of coupons that contain the results of a pre-computation; this avoids the need to support elliptic curve operations on the tag. Certainly limited-use tokens are familiar in a wide range of applications from pre-paid telephone cards to public transport ticketing. However, their use is not to everyone's taste and they are not suitable for all use-cases.

[1] It is typical to use the *gate equivalent (GE)* to compare implementations. The physical area is divided by the size of a nand gate to give a broadly technology-neutral estimate of its size. While not perfect, it remains sufficiently useful.

Tag		**Reader**
PARAMETERS		
Curve $\mathcal{C}$, point P		Curve $\mathcal{C}$, point P
KEYS		
Secret key *(sk)* $s \in_R \{0,1\}^\sigma$ Secret key $k \in_R \{0,1\}^\kappa$		Public key *(pk)* $V = -sP$
COUPON PRE-COMPUTATION WITH PRNG		
For $0 \leq i \leq n-1$ Let $r_i = \text{PRNG}_k(i)$ where $\lvert r_i \rvert = \rho$ Set $x_i = \lceil \text{HASH}(r_i P) \rceil_t$ Store coupon x_i		
PROTOCOL USING ON-TAG PRNG		
At time i fetch x_i	$\xrightarrow{x_i}$	
	$\xleftarrow{c_i}$	Pick $c_i \in_R \{0,1\}^\delta$
Generate $r_i = \text{PRNG}_k(i)$		
$y_i = r_i + (s \times c_i)$	$\xrightarrow{y_i}$	$\lceil \text{HASH}(y_i P + c_i V) \rceil_t \overset{?}{=} x_i$

Fig. 1. The typical description of CRYPTOGPS using the most common implementation optimisations, where PRNG is a pseudo-random generator, HASH is a hash function, and where σ, κ and t and δ are security parameters that will be discussed further (see Section 5.1)

However, this is not the focus of the paper and issues around the use of coupons, a topic that is broader than their use with CRYPTOGPS, are discussed in the Appendix. Instead, we will be concerned with the well-known Fiat-Shamir conversion [11] of a basic CCR scheme into a signature scheme. And our goal is to make this conversion, and the resultant scheme, more computationally efficient than was previously recognised.

2.2 This Paper

It is well-known that an interactive identification scheme can be converted into a digital signature scheme [11,31] and the security provided by this conversion was proved by Pointcheval and Stern [36,37]. Indeed two signature variants of CRYPTOGPS have already been standardised within ISO/IEC; details are available in ISO/IEC 14888-2 [22].

Classically the tool to perform this conversion is a hash function HASH. In general terms, the commitment x_i from the original CCR scheme is combined with the message m to be signed using a hash function, $\text{HASH}(x_i, m)$. The output from $\text{HASH}(x_i, m)$, or part of it, is then used as the challenge c_i. Some previous work in the literature has tried to establish the implementation profile of such a scheme when using CRYPTOGPS [30] but this only confirms its unsuitability for UHF RFID tags, at least in this classical form. To change this view we need something new.

As a first step we observe that ISO/IEC 14888-2 makes a distinction between two types of signatures. The first is referred to as a *transmissible signature*; that is a digital signature that can be verified by a third party at any time. The second type of signature is referred to as a *non-transmissible signature* and is used solely in a dynamic setting. Here the "message" to be signed comes from the verifier. The prover (or tag) computes the signature on this message and returns the result. The verifier can set a time-limit, or time-out, to fix the amount of time that is available for the tag to respond. If the tag responds in time (with a valid signature) then the verifier is convinced that the tag is genuine. However, the verifier would be unable to convince a third-party of this unless he can further guarantee that the signature was computed within a certain time on a fresh challenge. Nevertheless we have what we want; we have a dynamic tag authentication scheme that means an interrogator can be certain a tag is genuine.

While helpful, we would still have a proposal that is too large for UHF RFID deployment. The technical contribution of this paper, therefore, is to find a more efficient (and secure) way of making the CCR-to-signature conversion. In this paper we outline a full solution with preferred parameter sets. In fact it is only by replacing the function HASH that we can derive a practical scheme. This, along with the opportunity to use pre-existing components on the tag, allows us to move to a new improved variant of CRYPTOGPS with only a moderate increase in area on the tag. And we reap the operational advantage that the scheme now becomes *challenge-response* instead of *commitment-challenge-response*, improving both the communication burden and the system complexity.

To provide context, we note that various papers consider the practicability of implementing elliptic curve schemes on RFID tags. These suggest that the area needed to implement elliptic curve operations is in excess of 13 000 GE and the time to process an operation requires several tens of thousands of cycles, *e.g.* [48,47]. These numbers are markedly greater than what one could expect from an implementation of the non-transmissible signature variant of CRYPTOGPS with the coupon optimization (see Appendix). This paper relies on the use of a hash function based around a block cipher. This has been [45,46], and is likely to remain, an active area of research which may have some future bearing on the work considered in this paper.

3 Moving to Non-transmissible Signatures

The classical CCR-signature conversion requires the use of a hash function. Yet typical[2] hash functions are not at all suitable for UHF RFID tags [5]. Instead we would like to instantiate the conversion using a block cipher, particularly since one already implements PRESENT on the tag in support of CRYPTOGPS [38].

[2] Some lightweight hash functions have been recently proposed [2,6,18] but they are new and tend to have long processing times.

Tag		Reader
	PARAMETERS	
Curve $\mathcal{C}$, point P		Curve $\mathcal{C}$, point P
	KEYS	
Secret key *(sk)* $s \in_R \{0,1\}^\sigma$ Secret key $k \in_R \{0,1\}^\kappa$		Public key *(pk)* $V = -sP$
	COUPON PRE-COMPUTATION WITH PRNG	
For $0 \leq i \leq n-1$ Let $r_i = \text{PRNG}_k(i)$ where $\lvert r_i \rvert = \rho$ Set $x_i = \lceil \text{HASH}(r_iP) \rceil_t$ Store coupon x_i		
	PROTOCOL USING ON-TAG PRNG	
	$\xleftarrow{w}$	Pick $w \in_R \{0,1\}^\delta$
At time i fetch x_i Compute $c_i = \text{F}(x_i, w)$ Generate $r_i = \text{PRNG}_k(i)$		
$y_i = r_i + (s \times c_i)$	$\xrightarrow{y_i, c_i}$	Compute $x' = \lceil \text{HASH}(y_iP + c_iV) \rceil_t$ $\text{F}(x', w) \overset{?}{=} c_i$

Fig. 2. The non-transmissible signature variant of CRYPTOGPS for dynamic authentication. We propose that the function F be built around the block cipher PRESENT, see Section 3.1.

3.1 Choice of Conversion Function

The conversion of an three-pass identification scheme into a signature scheme has been well-studied in the literature. The first proposals by Fiat and Shamir [11] remain the foundation for this conversion and proofs of security followed when a more rigorous theoretical foundation had been established [36].

Our essential requirement is that the output of the conversion function, which we will denote F, is unpredictable for different inputs while the same inputs yield the same output. A simple and elegant way to construct F, while respecting the preferred parameters derived in Section 3.2 is to set

$$c_i = \text{F}(x_i, w) = \text{ENC}_{x_i \| w}(0^n)$$

where encryption is performed using the 128-bit key version of PRESENT [4].

Digression. There might be some interest in understanding how we arrived at this choice for F. Indeed, at first sight it is not clear that the *same* function F is required at both the tag and the reader and we could consider a protocol as follows:

PROTOCOL USING ON-TAG PRNG

$$\begin{array}{rcl} & \xleftarrow{\quad w \quad} & \text{Pick } w \in_R \{0,1\}^{\delta} \\ \text{At time } i \text{ fetch } x_i & & \\ \text{Compute } c_i = \text{F}(x_i, w) & & \\ \text{Generate } r_i = \text{PRNG}_k(i) & & \\ y_i = r_i + (s \times c_i) & \xrightarrow{\quad y_i, c_i \quad} & \text{Compute } x' = \lceil \text{HASH}(y_i P + c_i V) \rceil_t \\ & & \text{VERIF.}(x', w, c_i) \overset{?}{=} \text{TRUE} \end{array}$$

In this case, some hypothetical candidates for F might include:

$\text{F}(x_i, w)$	$\text{VERIF.}(x', w, c_i)$
$c_i = \text{ENC}_{x_i \Vert w}(x_i)$	$\text{DEC}_{x' \Vert w}(c_i) = x'$
$c_i = \text{ENC}_{x_i \Vert w}(w)$	$\text{DEC}_{x' \Vert w}(c_i) = w$
$c_i = \text{ENC}_{w}(x_i)$	$\text{DEC}_{w}(c_i) = x'$
$c_i = \text{ENC}_{x_i}(w)$	$\text{DEC}_{x'}(c_i) = w$

It can be easily seen that not all of these approaches are secure. Further, we concentrated our efforts on the simplest and most efficient-to-implement schemes. In turn, this matched the theoretical analysis presented in Section 4. In the remainder of the paper, therefore, F will refer to the following transformation:

$$c_i = \text{F}(x_i, w) = \text{ENC}_{x_i \Vert w}(0^n).$$

3.2 Setting Parameter Sizes

In this section we consider some attacks that help us better understand the trade-offs between different parameters. As a baseline, however, we assume that all the secret keys held on the tag, both for PRESENT and CRYPTOGPS, have a length that is intended to provide 80-bit security.

For all challenge-response protocols there are some basic on-line attacks, *i.e.* without any pre-processing. These attempts to fool the reader into accepting a fake tag as genuine and have a certain probability of success at each run of the protocol.

1. The attacker chooses random y_i and c_i and sends these as a response. The probability that $\text{F}(\lceil \text{HASH}(y_i P + c_i V) \rceil_t, w) = c_i$, where w was sent by the verifier, is given by $2^{-|c_i|}$ so the probability of success is related to the size of c_i.
2. The attacker picks x_i at random and computes c_i on receiving w. The attacker then randomly chooses y_i and sends the response. He will be successful if $\lceil \text{HASH}(y_i P + c_i V) \rceil_t = x_i$. The probability of success is $2^{-|x_i|}$ and is related to the coupon size.

The net result of these attacks is to set $|x_i| = |c_i| \geq z$ if we are aiming for an impersonation probability less than 2^{-z}. This per-session forgery probability can be improved in an obvious way using off-line pre-computation and storage. Essentially, one uses either of the two techniques above to construct a valid and consistent $\{x_i, w, c_i, y_i\}$ quadruple, though only w, c_i, and y_i need to be stored. Using the first approach, computing d quadruples take an off-line work effort proportional to $d2^{|c_i|}$ operations while the second requires a work effort of $d2^{|x_i|}$ operations. The probability of success for each session then becomes $d2^{-|w|}$ since for d potential values of w the fake tag contains a good response.

A related result stems from the phenomenon of u-collisions, explored by Girault and Stern in two papers [13,17]. Since the coupons are constructed using a hash function it is possible that $\lceil \text{HASH}(y_iP + c_iV) \rceil_t = \lceil \text{HASH}(y_i'P + c_i'V) \rceil_t$ for two potentially different sets of inputs. This is the familiar hash function collision and the probability of finding a 2-collision is related to the birthday paradox. If we move to larger values of u then, depending on the size of x_i and the amount of computation devoted to an off-line attack, an attacker can expect to find some x_i for which u values of $\{y_i, c_i\}$ will hash to x_i. In more detail, he fixes y_i and searches over c_i storing the resulting x_i. With a work effort of, say, 2^{80} operations and coupons of size $|x| = 64$ he can expect to have a 2^{16}-collision for a given value of x_i and y_i. This then means that 2^{16} values of c_i will allow a forgery, and the probability of alighting on one of these values in practice—when w is chosen at random by the verifier—is $2^{16-|c|}$.

There are two complementary aspects to this observation. The first is that the size of the coupons has an impact on the success probabilities of both on-line and off-line attacks. The second is that, in practice, an attacker is unlikely to use 2^{80} operations to gain a per-session advantage. Since the same work effort can recover the long-term secret CRYPTOGPS key, which was chosen to offer 80-bit security, this latter attack would in fact be preferable. Indeed, since efforts to recover s are probabilistic [31], even a work effort significantly less than 2^{80} operations will have some probability of yielding the long-term secret key (and fully compromising the tag). So the cost-benefit for the attacker in devoting vast amounts of pre-computation to giving an advantage in a single run of the authentication protocol is not clear.

Finally, we observe that a device impersonating a genuine passive UHF tag might not, itself, be a passive UHF tag. It could be a tag that is connected via a relay to a much more powerful device, or it could even be self-powered; if the "tag" is not visually inspected, *e.g.* because it is inside a crate, then it could be anything. So after receiving w a false "tag" can choose/search x_i and y_i until finding values for which $\lceil \text{HASH}(y_iP + c_iV) \rceil_t = x_i$. Whether or not this is likely to be accomplished within a specified response time depends on the parameter values and the computational complexity of the emulating/remote device.

4 Security Foundations

The previous section was concerned with the practical aspects of setting parameter sizes. Here we consider the theoretical foundations of our preferred conversion method.

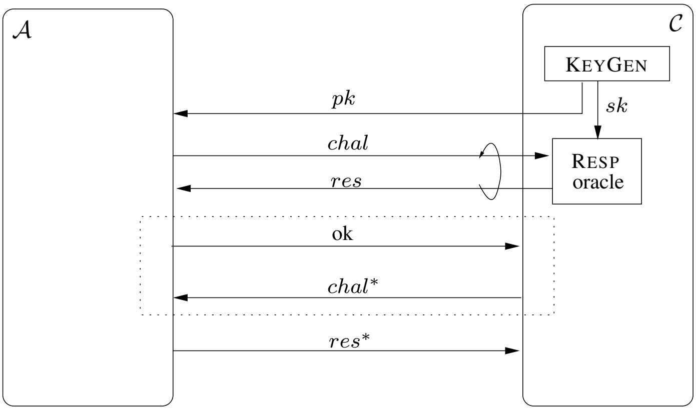


Fig. 3. Soundness: $\mathcal{A}$ wins if $\mathsf{Verif}(pk, chal^*, res^*) = 1$ and $(chal^*, res^*)$ does not come from the Res oracle

For this it might be helpful to consider the different components of an (RFID) authentication scheme.

- KEYGEN: on input a security parameter λ, KEYGEN generates key pair (pk, sk), possibly certified by some certification authority CA.
- CHALL: the Reader generates a challenge $chal$, on input the public key pk of the Tag.
- RESP: the Tag uses sk and the challenge $chal$ to generate a string res.
- VERIF: on input pk, $chal$ and res, the Reader outputs a bit 0/1 to denote either reject or accept.

Clearly we require *correctness*, that is if (sk, pk) is output by KEYGEN and if res is computed using both sk and $chal = \text{CHALL}(pk)$, then $\text{VERIF}(pk, chal, res) = 1$ with overwhelming probability. The background to the security notion, *soundness*, is illustrated in Figure 3. A challenger $\mathcal{C}$ is matched against $\mathcal{A}$, an adversary against soundness. $\mathcal{A}$ can receive legitimate challenge-response pairs but is then required to reply to a previously unseen challenge. We say that an authentication scheme is secure if, and only if, the probability that the adversary $\mathcal{A}$ can provide a good response is negligible (in the security parameter).

The security of our proposal is substantiated in several steps.

From signature to authentication. Given a signature scheme, it is easy to design a 2-pass authentication scheme. The reader sends a challenge $chal$ and the tag produces the response res as a signature on the message $chal$. The verification procedure is given by signature verification. It is well known that such an authentication scheme is secure (*i.e.* sound) if the used signature scheme is unforgeable.

(In fact this can be seen from Figure 3 since the signature unforgeability experiment is similar to that described in Figure 3 where the response res^* is composed of both a challenge message w^* and the corresponding forged signature σ^*.)

The Fiat-Shamir transformation and the random oracle model. As we have seen before, the Fiat-Shamir heuristic [11] consists in replacing the random challenge c by the output of a hash function HASH taking as input the prover's commitment x_i and the message w: $c = \text{HASH}(x_i, w)$. Pointcheval and Stern proved [36,37] that the resulting signature scheme is unforgeable, assuming that the hash function HASH is a random oracle [3]. In a nutshell, a random oracle idealizes the hash function that behaves as a random function that gives unpredictable outputs (but the same input always gives the same output).

As said previously, hash functions are not suitable for UHF RFID tags and we should instead find something more interesting.

ICM. The *ideal cipher model* (ICM) is an idealized model [42] in which a random block cipher (seen as an ideal cipher) with an n-bit input/output and a κ-bit randomly-chosen secret key is computationally indistinguishable from a randomly chosen n-bit permutation. More formally, given an ideal cipher denoted $\text{ENC} : \{0,1\}^\kappa \times \{0,1\}^n \longrightarrow \{0,1\}^n$, an adversary having the possibility to make both encryption and decryption queries to the ideal block cipher, for any key, cannot distinguish a given output from that of a randomly-chosen permutation. In such a case, we can consider the ideal cipher ENC outputs as being those of a randomly chosen n-bit permutation.

As we want to use instead of a hash function the block cipher PRESENT, the use of an ideal cipher may help us to obtain a (proven to be) secure construction.

The Fiat-Shamir heuristic and the ideal cipher. Luckily, Coron *et al.* proposed in [9] a black-box transformation of any ideal cipher into a random oracle. Given an ideal cipher $\text{ENC} : \{0,1\}^\kappa \times \{0,1\}^n \longrightarrow \{0,1\}^n$ and the message $(w_1 \| \cdots \| w_\ell)$ to be hashed, the construction works as follows:

- set y_0 to 0^n (or to any fixed IV);
- for $i = 1$ to ℓ do $y_i = \text{ENC}_{w_i}(y_{i-1}) \oplus y_{i-1}$;
- output y_ℓ.

With a single block w_1 this corresponds to the computation $y_1 = \text{ENC}_{w_1}(0^n)$. Thus, given an ideal cipher ENC, one can replace the hash function/random oracle of the Fiat-Shamir transform by the above construction and the security proof immediately follows, with a security parameter corresponding to the block size n used in ENC.

We can next use this construction within the Fiat-Shamir heuristic, which corresponds to our construction, in the particular case of CRYPTOGPS.

Signatures and CRYPTOGPS. In fact, CRYPTOGPS was proven to be a secure zero-knowledge proof [40,16]. Thus we can directly apply the above results. If, on input w and commitment x_i, the tag computes challenge $c = \text{ENC}_{x_i \| w}(0^n)$ and response y, then the signature $\sigma = (c, y)$ is that of an unforgeable signature scheme and the authentication scheme described in Figure 2 is also secure.

5 Implementation Perspectives

While the conversion of an identification scheme to a signature scheme is well-known, previous work on CRYPTOGPS has avoided this. The main reason is the additional complexity of supporting the conversion function F.

However, in this paper we show that existing components can be re-used and we instantiate the ideal cipher with the 128-bit key version of PRESENT. The cipher has been analysed widely in the community [7,8,26,32,34,43,44] including under related-key attacks [35]. So while the CCR-to-signature conversion necessarily implies some overhead to the area that is required on the tag, these overheads are slight and can be predicted with a reasonable level of confidence. In return, the gain is significant in terms of system complexity as the three-pass CCR scheme is replaced by a simple challenge-response protocol.

5.1 Preferred Parameter Sets

It is typical in environments where computational devices are quite constrained to aim for a security level of “80 bits”. Of course, if a greater security level can be comfortably accommodated then all the better. But this can compromise performance or even, in the worst case, mean security cannot be implemented at all on a passive UHF tag.

For implementations of CRYPTOGPS there are two per-tag secrets and the loss of either would entirely compromise the tag. The first is the secret CRYPTOGPS key. This could be attacked using techniques to solve the elliptic curve problem and we can turn to the established literature to establish an appropriate security level. Since any key is specific to a single tag, it is unlikely that we would need to protect against a widely distributed Internet-based effort. Thus the security level of 2^{80} seems reasonable and is attained using a CRYPTOGPS secret of length $|s| = 160$. Each tag also uses a PRNG to regenerate r_i. This requires a per-tag secret key and, again, an 80-bit key would be appropriate. This fits well with the goal of using PRESENT in this role, though since we aim to use PRESENT with 128-bit keys as the basis for the function F, we can either use a 128-bit key with PRESENT-128 as the PRNG or we can use a padded 80-bit key. There is no significant impact in performance for either choice.

Taking into account the security analysis given in Section 3.2, we propose to use a reader-provided challenge w of size 64 bits, and to fix the size of the coupons x_i and the derived challenges c_i to 64 bits. And, in turn, we can set $|y_i| = |r_i| = \rho = |s| + |c_i| + 80 = 304$.

Turning to wider considerations, the per-tag public key needs to be delivered to the reader in an authenticated way. There are a variety of architectural ways that this might be done. However, the conventional solution would be to sign the per-tag public key—using a system-side signature algorithm—and the tag can deliver its public key and associated signature to the reader. The reader holds the signing verification key needed to authenticate the per-tag public key. Unlike the per-tag public-private keys, the system-wide signing key would be a single point of failure for a widely-deployed system and a security level greater than 80-bit is likely to be preferred.

5.2 Area, Time, and Complexity

There are many factors to consider when implementing cryptography on an RFID tag. In this section we take the preferred parameter set outlined above and estimate the impact of implementing the non-transmissible signature variant of CRYPTOGPS. Our calculations are best-effort, but since they are based on a wealth of data from synthesized and fabricated versions of PRESENT and CRYPTOGPS we expect them to be reasonably accurate.

The on-tag requirements for the CRYPTOGPS computation are PRESENT and an integer multiplication. There are numerous implementations of PRESENT with 80-bit keys, some fabricated [38] but not so many with 128-bit keys. Instead we refer to [4] where the same technology is used to synthesize both variants and requiring 1570 GE for PRESENT-80 and 1886 GE for PRESENT-128. Both require the same time to complete an encryption operation.

When we turn to the computation of $y = r + sc$, the most useful source of information data is [39]. There, different strategies for implementing the computation of sc are described and compared. More usefully, the implementations outlined in [39] give area and time estimates for combining the multiplication operation with the regeneration of r when using PRESENT. This is an important issue since the combined operation can suffer from unexpected latencies unless optimisations to the two components are done in a coherent manner. Happily, the results in [39] cover the case of using a 160-bit secret s and a 64-bit challenge c and so there is no need to take any liberties in extrapolating from smaller parameters. The anticipated area and time requirements for the non-transmissible signature variant of CRYPTOGPS, denoted CRYPTOGPS-NTS, are given below for two different implementations strategies, denoted (A) and (B), which give different area/time trade-offs.

	CRYPTOGPS		CRYPTOGPS-NTS	
F	-		PRESENT-128	
PRNG	PRESENT-80		PRESENT-128	
	(A)	(B)	(A)	(B)
estimated area (GE)	3 424	3 300	3 740	3 616
estimated time (cycles)	389	713	421	745

In short, the area overhead in moving to CRYPTOGPS-NTS could be as little as 316 GE. However, during fabrication there are inevitable increases (typically of the order of 18-20%) to the area that are not reflected in synthesis results. Further, we have an additional complexity on the tag; namely the computation of $\mathrm{ENC}_{x_i \| w}(0^n)$. While this won't add too much in terms of area, there will be an additional complexity to the implementation as the key for the PRESENT unit is swapped with the key k that is used to generate r. This will be reflected in some increase to the control logic and some additional time. The time for changing the key will not be significant; it consists merely of writing over the key state with a new value and this will depend on the internal operand size. It will likely remain a small fraction of the total processing time on the tag. For the increase in the control logic, we note from [39] that the control logic for the implementation of the computation $r + sc$ consumes around 6-10% of the total area. Even a doubling of the control logic, which is somewhat unlikely, would yield an additional overhead of 10%

to the complete implementation, which is within the margin of error that this kind of computation inevitably carries.

In terms of time for the on-tag computation, the computation of F will be an overhead, but depending on the working unit for the computation it is likely to be 32 or 64 cycles. This additional cost will be more than compensated for by the fact that the protocol is now challenge-response. The protocols used in RFID applications require that the tag responds to the reader. To illustrate, for a CCR scheme we would expect something like the following schema:

$$\begin{array}{rcl} \text{Reader} & & \text{Tag} \\ \texttt{start} & \longrightarrow & \\ & \longleftarrow & \texttt{send}\, x_i \\ \texttt{send challenge}\, c & \longrightarrow & \texttt{compute}\, y_i \\ & \longleftarrow & \texttt{send}\, y_i \end{array}$$

For the signature variant we would have

$$\begin{array}{rcl} \text{Reader} & & \text{Tag} \\ \texttt{send challenge}\, w & \longrightarrow & \texttt{compute}\, c_i = \mathrm{F}(x_i, w) \\ & & \texttt{compute}\, y_i \\ & \longleftarrow & \texttt{send}\, c_i\ \texttt{and}\, y_i \end{array}$$

The amount of operational data sent would be the same in both cases, namely 432 bits[3] for our preferred parameter sets. However, in a multi-tag (potentially multi-reader) environment a single challenge-response interchange is easier and more reliable to maintain. Further, each message sent between reader and tag has an operational overhead; there is header information and trailing information that carries the results of a CRC computation. Much depends on the specific formats of the commands and messages, but a saving of around 40 bits in total is likely. This may not sound like a lot, but each bit counts and suggests a reduction of around 10% in the total communication overhead.

To summarise, we estimate that the non-transmissible signature version of CRYPTOGPS can be implemented in around 4000 GE within around 800 cycles. Given the increased reliability and simplicity when using a challenge-response protocol, it seems likely that this variant of CRYPTOGPS will be of some interest in future prototyping. Of course, it should be noted that we have concentrated on performance issues such as area and processing time. In fact, the average and peak power consumption are also crucial and while existing work on PRESENT and CRYPTOGPS are very promising in this regard, this aspect of the scheme we have presented will be considered further in future work.

6 Conclusions

In this paper we have considered a signature variant of the CRYPTOGPS commitment-challenge-response (CCR) scheme. This variant has not been widely considered for UHF RFID tag deployment, despite featuring as an ISO standard, since the classical

[3] According to the parameters choice (see Section 5.1), $|x_i|+|c_i|+|y_i| = 64+64+304 = 432$.

conversion from CCR to signature scheme is too costly on the tag. However, we have shown that it is possible to re-use the block cipher that is already required to support CRYPTOGPS and to define a different conversion method. Fully supported by theoretical security arguments, the preferred parameter set for this variant of CRYPTOGPS appears to be well-suited for UHF RFID deployment.

Acknowledgements. This work has been partially supported by the European Commission under Contract ICT-2007-216676 ECRYPT II. We are moreover grateful to the anonymous referees for their interesting and useful comments.

References

1. Aigner, M., Burbridge, T., Ilic, A., Lyon, D., Soppera, A., Lehtonen, M.: RFID Tag Security, BRIDGE white paper, `http://www.bridge-project.eu`
2. Aumasson, J.-P., Henzen, L., Meier, W., Naya-Plasencia, M.: QUARK: A Lightweight Hash. In: Mangard, S., Standaert, F.-X. (eds.) CHES 2010. LNCS, vol. 6225, pp. 1–15. Springer, Heidelberg (2010)
3. Bellare, M., Rogaway, P.: Random Oracles are Practical: A paradigm for designing efficient protocols. In: ACM CCS 1993, pp. 62–73. ACM (1993)
4. Bogdanov, A., Knudsen, L.R., Leander, G., Paar, C., Poschmann, A., Robshaw, M.J.B., Seurin, Y., Vikkelsoe, C.: PRESENT: An Ultra-Lightweight Block Cipher. In: Paillier, P., Verbauwhede, I. (eds.) CHES 2007. LNCS, vol. 4727, pp. 450–466. Springer, Heidelberg (2007)
5. Bogdanov, A., Leander, G., Paar, C., Poschmann, A., Robshaw, M.J.B., Seurin, Y.: Hash Functions and RFID Tags: Mind the Gap. In: Oswald, E., Rohatgi, P. (eds.) CHES 2008. LNCS, vol. 5154, pp. 283–299. Springer, Heidelberg (2008)
6. Bogdanov, A., Knežević, M., Leander, G., Toz, D., Varıcı, K., Verbauwhede, I.: SPONGENT: A Lightweight Hash Function. In: Preneel, B., Takagi, T. (eds.) CHES 2011. LNCS, vol. 6917, pp. 312–325. Springer, Heidelberg (2011)
7. Cho, J.Y.: Linear Cryptanalysis of Reduced-Round PRESENT. In: Pieprzyk, J. (ed.) CT-RSA 2010. LNCS, vol. 5985, pp. 302–317. Springer, Heidelberg (2010)
8. Collard, B., Standaert, F.-X.: A Statistical Saturation Attack against the Block Cipher PRESENT. In: Fischlin, M. (ed.) CT-RSA 2009. LNCS, vol. 5473, pp. 195–210. Springer, Heidelberg (2009)
9. Coron, J.-S., Dodis, Y., Malinaud, C., Puniya, P.: Merkle-Damgård Revisited: How to Construct a Hash Function. In: Shoup, V. (ed.) CRYPTO 2005. LNCS, vol. 3621, pp. 430–448. Springer, Heidelberg (2005)
10. EPCglobal. EPC Radio-Frequency Identity Protocols, Class-1 Generation-2 UHF RFID, Protocol for Communications at 860-960 MHz, version 1.2.0 (October 23, 2008), `http://www.epcglobalinc.org`
11. Fiat, A., Shamir, A.: How to Prove Yourself: Practical Solutions to Identification and Signature Problems. In: Odlyzko, A.M. (ed.) CRYPTO 1986. LNCS, vol. 263, pp. 186–194. Springer, Heidelberg (1987)
12. Girault, M.: Self-certified Public Keys. In: Davies, D.W. (ed.) EUROCRYPT 1991. LNCS, vol. 547, pp. 490–497. Springer, Heidelberg (1991)
13. Girault, M.: Low-Size Coupons for Low-Cost IC Cards. In: Domingo-Ferrer, J., Chan, D., Watson, A. (eds.) Proceedings of Smart Card Research and Advanced Applications, pp. 39–50. Kluwer Academic Press (2001)

14. Girault, M., Juniot, L., Robshaw, M.: The Feasibility of On-the-Tag Public Key Cryptography. RFIDsec 2007, Workshop Record (2007), http://rfidsec07.etsit.uma.es/slides/papers/paper-32.pdf
15. Girault, M., Lefranc, D.: Public Key Authentication with One (Online) Single Addition. In: Joye, M., Quisquater, J.-J. (eds.) CHES 2004. LNCS, vol. 3156, pp. 413–427. Springer, Heidelberg (2004)
16. Girault, M., Poupard, G., Stern, J.: On the Fly Authentication and Signature Schemes Based on Groups of Unknown Order. Journal of Cryptology 19, 463–487 (2006)
17. Girault, M., Stern, J.: On the Length of Cryptographic Hash-Values Used in Identification Schemes. In: Desmedt, Y.G. (ed.) CRYPTO 1994. LNCS, vol. 839, pp. 202–215. Springer, Heidelberg (1994)
18. Guo, J., Peyrin, T., Poschmann, A.: The PHOTON Family of Lightweight Hash Functions. In: Rogaway, P. (ed.) CRYPTO 2011. LNCS, vol. 6841, pp. 222–239. Springer, Heidelberg (2011)
19. Hofferek, G., Wolkerstorfer, J.: Coupon Recalculation for the GPS Authentication Scheme. In: Grimaud, G., Standaert, F.-X. (eds.) CARDIS 2008. LNCS, vol. 5189, pp. 162–175. Springer, Heidelberg (2008)
20. Hutter, M., Nagl, C.: Coupon Recalculation for the Schnorr and GPS Identification Scheme: A Performance Evaluation. In: Proceedings of RFIDSec 2009 (2009), http://www.cosic.esat.kuleuven.be/rfidsec09/
21. ISO/IEC 9798: Information Technology – Security Techniques – Entity Authentication – Part 5: Mechanisms using Zero-Knowledge Techniques, http://www.iso.org
22. ISO/IEC 14888-2: Information Technology – Security Techniques – Digital Signatures – Part 2: Factoring Based Techniques
23. ISO/IEC 29192-4: Information Technology – Security Techniques – Lightweight Cryptography – Part 2: Block ciphers
24. ISO/IEC 29192-4: Information Technology – Security Techniques – Lightweight Cryptography – Part 4: Public key techniques
25. Jenkins, J., Mills, P., Maidment, R., Profit, M.: Pharma Traceability Business Case Report. BRIDGE white paper (May 2007), http://www.bridge-project.eu
26. Leander, G.: On Linear Hulls, Statistical saturation attacks, PRESENT and a cryptanalysis of PUFFIN. In: Paterson, K.G. (ed.) EUROCRYPT 2011. LNCS, vol. 6632, pp. 303–322. Springer, Heidelberg (2011)
27. Lehtonen, M., Al-Kassab, J., Michahelles, F., Kasten, O.: Anti-counterfeiting Business Case Report. BRIDGE white paper (December 2007), http://www.bridge-project.eu
28. McLoone, M., Robshaw, M.J.B.: Public Key Cryptography and RFID Tags. In: Abe, M. (ed.) CT-RSA 2007. LNCS, vol. 4377, pp. 372–384. Springer, Heidelberg (2007)
29. McLoone, M., Robshaw, M.J.B.: New Architectures for Low-Cost Public Key Cryptography on RFID Tags. In: Proceedings of SecureComm 2005, pp. 1827–1830. IEEE Computer Society Press (2007)
30. McLoone, M., Robshaw, M.J.B.: Low-cost Digital Signature Architecture Suitable for Radio-Frequency Identification Tags. IET Computers and Digital Techniques 4(1), 14–26 (2010)
31. Menezes, A.J., van Oorschot, P.C., Vanstone, S.A.: Handbook of Applied Cryptography, 1st edn. CRC Press, Boca Raton (1996)
32. Nakahara Jr., J., Sepehrdad, P., Zhang, B., Wang, M.: Linear (Hull) and Algebraic Cryptanalysis of the Block Cipher PRESENT. In: Garay, J.A., Miyaji, A., Otsuka, A. (eds.) CANS 2009. LNCS, vol. 5888, pp. 58–75. Springer, Heidelberg (2009)
33. NESSIE consortium. Final Report of European Project IST-1999-12324: New European Schemes for Signatures, Integrity, and Encryption (NESSIE) (April 2004), https://www.cosic.esat.kuleuven.be/nessie/

34. Ohkuma, K.: Weak keys of reduced-round PRESENT for linear cryptanalysis. In: Jacobson Jr., M.J., Rijmen, V., Safavi-Naini, R. (eds.) SAC 2009. LNCS, vol. 5867, pp. 249–265. Springer, Heidelberg (2009)
35. Özen, O., Varıcı, K., Tezcan, C., Kocair, Ç.: Lightweight Block Ciphers Revisited: Cryptanalysis of Reduced Round PRESENT and HIGHT. In: Boyd, C., González Nieto, J. (eds.) ACISP 2009. LNCS, vol. 5594, pp. 90–107. Springer, Heidelberg (2009)
36. Pointcheval, D., Stern, J.: Security Proofs for Signature Schemes. In: Maurer, U. (ed.) EUROCRYPT 1996. LNCS, vol. 1070, pp. 387–398. Springer, Heidelberg (1996)
37. Pointcheval, D., Stern, J.: Security Arguments for Digital Signatures and Blind Signatures. Journal of Cryptology 19, 361–396 (2000)
38. Poschmann, A., Robshaw, M., Vater, F., Paar, C.: Lightweight Cryptography and RFID: Tackling the Hidden Overheads. In: Lee, D., Hong, S. (eds.) ICISC 2009. LNCS, vol. 5984, pp. 129–145. Springer, Heidelberg (2010)
39. Poschmann, A., Robshaw, M.J.B.: On Area, Time, and the Right Trade-Off. In: Susilo, W., Mu, Y., Seberry, J. (eds.) ACISP 2012. LNCS, vol. 7372, pp. 404–418. Springer, Heidelberg (2012)
40. Poupard, G., Stern, J.: Security Analysis of a Practical "on the fly" Authentication and Signature Generation. In: Nyberg, K. (ed.) EUROCRYPT 1998. LNCS, vol. 1403, pp. 422–436. Springer, Heidelberg (1998)
41. Robshaw, M.: The eSTREAM Project. In: Robshaw, M., Billet, O. (eds.) New Stream Cipher Designs. LNCS, vol. 4986, pp. 1–6. Springer, Heidelberg (2008)
42. Shannon, C.: Communication theory of secrecy systems. Bell System Technical Journal 28, 656–715 (1949)
43. Wang, M.: Differential Cryptanalysis of Reduced-Round PRESENT. In: Vaudenay, S. (ed.) AFRICACRYPT 2008. LNCS, vol. 5023, pp. 40–49. Springer, Heidelberg (2008)
44. Z'aba, M.R., Raddum, H., Henricksen, M., Dawson, E.: Bit-Pattern Based Integral Attack. In: Nyberg, K. (ed.) FSE 2008. LNCS, vol. 5086, pp. 363–381. Springer, Heidelberg (2008)
45. Black, J., Rogaway, P., Shrimpton, T.: Black-Box Analysis of the Block-Cipher-Based Hash-Function Constructions from PGV. In: Yung, M. (ed.) CRYPTO 2002. LNCS, vol. 2442, pp. 320–335. Springer, Heidelberg (2002)
46. Preneel, B., Govaerts, R., Vandewalle, J.: Hash functions based on block ciphers: A synthetic approach. In: Stinson, D.R. (ed.) CRYPTO 1993. LNCS, vol. 773, pp. 368–378. Springer, Heidelberg (1994)
47. Braun, M., Hess, E., Meyer, B.: Using Elliptic Curves on RFID Tags. IJCSNS International Journal of Computer Science and Network Security 8(2) (February 2008); Jun, J.M. (ed.)
48. Batina, L., Guajardo, J., Kerins, T., Mentens, N., Tuyls, P., Verbauwhede, I.: An Elliptic Curve Processor Suitable For RFID-Tags. In: 1st Benelux Workshop on Information and System Security (WISSec 2006), Antwerpen, Belgium, November 8-9, 14 pages (2006)

Appendix: On the Use of Coupons

As noted in the main text, CRYPTOGPS is ideally suited for use with coupons. A precomputed quantity that is used once and then discarded, coupons can be well-suited to many RFID applications. Often we expect tags to be read 10 to 20 times and then discarded or recommissioned. With a coupon of 64 bits, see Section 3.2, storing 10 or even 20 coupons does not pose a significant incremental cost for many applications.

However, the use of coupons is not to everyone's taste and certainly they are not suitable for all use-cases. Indeed some commentators are concerned that coupons could be consumed in a denial-of-service attack, *i.e.* by an attacker that maliciously exhausts coupons on a target RFID tag. This is true. But the benefit of such a time-consuming attack, that needs to be repeated on a tag-by-tag basis, is rarely if ever articulated. Nevertheless, in response to this concern there has been some work regarding on-the-tag coupon regeneration [19,20] though this does not seem to be realistic in deployment. Other more practical approaches have considered ways of reloading coupons and on mechanisms to deliver coupons directly to the reader so that they don't need to be carried on the tag.

All in all, the suitability of coupons depends fundamentally on the use-case and the kind of adversary we are likely to encounter.

Unlinkable Attribute-Based Credentials with Practical Revocation on Smart-Cards

Jan Hajny and Lukas Malina

Department of Telecommunications
Brno University of Technology, Brno, Czech Republic
{hajny,malina}@feec.vutbr.cz
http://crypto.utko.feec.vutbr.cz

Abstract. Attribute-based credentials are cryptographic schemes designed to enhance user privacy. These schemes can be used for constructing anonymous proofs of the ownership of personal attributes. The attributes can represent any information about a user, e.g., age, citizenship or birthplace. The ownership of these attributes can be anonymously proven to verifiers without leaking any other information. The problem of existing credential schemes is that they do not allow the practical revocation of malicious or expired users when slow off-line devices (for example, smart-cards) are used for storing attributes. This prevents existing systems from being used on eIDs (electronic ID cards), employees' smart-cards or, for example, library access cards. In this paper, we propose a novel cryptographic scheme which allows both expired user revocation and de-anonymization of malicious users on commercially available smart-cards. In addition to the full cryptographic specification of the scheme, we also provide implementation results on .NET V2+ and MultOS smart-card platform.

Keywords: Revocation, privacy, anonymity, smart-cards, credentials.

1 Introduction

Attribute-based credential schemes were proposed [9] to provide more privacy during the verification of users' attributes. By using attribute-based credentials, users can anonymously prove their possession of some attributes. These attributes can represent any personal data such as age, citizenship or valid driver's license. In contrast to classical authentication, the identity of attribute holders is never released. Thus, the verification process is anonymous and with many additional features protecting users' privacy. By using attribute-based credentials in eID (electronic ID cards), citizens would be able to prove their age, citizenship or any other attribute without releasing their identity or any other private information which might be abused by verifiers. With the increasing number of electronic services, smart-card applications and the approaching European eID cards, it is necessary to provide a cryptographic scheme with as many privacy-enhancing features as possible. These features have been demanded in both U.S. and EU official documents [22,18,21]. To preserve privacy, attribute-based credential schemes should provide following features.

S. Mangard (Ed.): CARDIS 2012, LNCS 7771, pp. 62–76, 2013.

- **Anonymity**: user's identity stays hidden during the verification of attribute ownership.
- **Untraceability**: attribute issuers are unable to trace issued attributes and their owners.
- **Unlinkability**: verification sessions of a single user are mutually unlinkable. This feature prevents from user profiling.
- **Selective disclosure of attributes**: users can selectively choose to disclose only a subset of private attributes to verifiers. Only attributes necessary for obtaining a service are disclosed.
- **Non-transferability**: lending of credentials is prevented.
- ***Revocation***: invalid, lost, stolen or expired credentials are revocable.
- ***Malicious user identification***: although proving attribute ownership is an anonymous process, the identity of malicious users and attackers can be revealed.

In particular, the last two items are very difficult to achieve using existing schemes. In this paper, we present a scheme which supports all the features.

1.1 Example Scenario

In this paper, we use a sample demonstration scenario to provide a practical example of using attribute-based credentials. Although many examples are available (e.g., proving age on a teenage webchat, proving citizenship on borders, proving legal drinking age), we chose a municipal library scenario. In a municipal library, users are required to pay quarterly fees to be allowed to borrow books. A citizen can be issued an attribute attesting to the paid fee. The attribute is stored on citizen's eID card (a smart-card) and the card is able to compute a proof of ownership of that attribute. By using the attribute-based credentials, a user can use the eID card to rent books. He just simply waves his contactless smart-card when he leaves the library with books. The first privacy-enhancing feature, *anonymity*, assures that nobody can link the identity of the user to the type of books he reads. This protects reader's privacy because his reading habits should be considered a private information. The *untraceability* feature prevents even the library which issued the attribute from seeing what books are read by a particular person. The *unlinkability* feature prevents from the profiling of users, all visits of a single user at the library are mutually unlinkable and the library is unable to get to know what books were read subsequently. In some scenarios, user profile can be so specific that it allows de-anonymization. The unlinkability feature prevents user profiling and such de-anonymization. The *selective disclosure of attributes* lets the user show only the attribute attesting to paid quarter fees. No other information or attributes are released. The *non-transferability* feature prevents readers from lending their cards to users who don‘t pay annual fees. Finally, the *revocation* and *malicious user identification* features allow library to expel and identify readers who violate the library's rules (e.g., steal books).

2 Current State in Revocation Techniques

To provide user privacy, the verification session must be totally anonymous, unlinkable to other sessions and revealing no personal or traceable information. In contrast to this requirement, some linking is required in situations where the eID card is lost, stolen or destroyed. In these cases, the credential must be revoked so that it cannot be used any time in future. Furthermore, if the user breaks the rules of the library (e.g., steals some books or does not return books in time), there must be a mechanism for the identification of such misbehaving users. The problem of existing attribute-based credentials is that they do not support the revocation of credentials and the identification of malicious users if off-line, computationally weak devices (such as smart-cards) are used for storing attributes. We provide a short overview of revocation techniques used in existing credential systems together with reasons why we consider them impractical.

Blacklisting of Credential IDs
Some credential systems, for example U-Prove [20], use a credential identifier embedded to each transaction. The identifier is a unique and unchangeable number linked to the credential. This number can be used for revoking the credential by putting it on a blacklist. Nevertheless, this approach destroys unlinkability (the unique credential identifier creates a link among user's verification sessions). Furthermore, a credential can be revoked only if a verifier has already seen the credential before and there is no mechanism for revoking credentials by their issuers. That is why the issuer has no power to revoke invalid credentials.

Blacklisting of Secrets
The technique for blacklisting of secrets, used, for example, in [2], allows an invalid credential to be revoked by using the knowledge of secret keys used for its construction. This technique can be used in cases where secret keys of users are revealed and for example made public on the Internet. In that case, a revocation authority can create a blacklist based on these keys to prevent verifiers from accepting credentials based on leaked keys. Nevertheless, this technique works only if the user secrets are revealed. But in most cases, the secret keys never leave a protected device (like a smart-card), therefore they cannot be revoked. Moreover, lost, stolen or expired credentials (e.g., stored on a smart-card) cannot be revoked because their secret keys never become public.

Epochs of Lifetime
Epochs of lifetime are the official revocation technique of idemix [5]. Here, a credential carries an epoch of validity as a special attribute. In this case, the verifier can check whether the credential is fresh. The user is required to renew his credential for every new epoch. The disadvantage of this mechanism is that the revocation of credentials is never immediate, the revoker must wait until their expiration and the issuer must stop issuing new credentials. The second major disadvantage is that the user must periodically run the issuance protocol with the issuer (or designated entity) to update his credential.

Accumulator Proofs
The most recent technique, used for example in [6,16], allows both issuers and verifiers to revoke credentials immediately by publishing so called whitelists. In this technique, a user must provide a proof that his credential is included on a list of valid credentials. This can be done anonymously and efficiently by using so called accumulators which accumulate all non-revoked users. Most efficient techniques are based on bilinear maps [16]. The disadvantage of these solutions is that the user must update his secrets every time any other user is revoked from the system. This is not a big problem when the user uses an online computer for his verification. On the other hand, if the user uses only an offline device, like a smart-card, then he is unable to update his secrets. Therefore, the user is unable to use his credentials after some other users are revoked from the system. We aim to provide a system which can be used for everyday verification in libraries, pubs or hotels, therefore the smart-card implementation is crucial. That is the reason why we consider the accumulator-based techniques impractical.

Verifiable Encryption of Secrets
The user identity or personal secrets can be encrypted inside the credential in such a manner that only a trusted authority can do the revocation or identity disclosure using decryption. In this case, the system might be considered insecure from the perspective of a user who does not fully trust the authority. In fact, this is likely a problem since users would not welcome a scheme where a fixed third party can learn all information about their verification sessions, including their identities. In practical scenarios, the user would have no choice from multiple trusted authorities. This even more degrades his trust in such a dictated authority. Furthermore, there is a problem with unlinkability because the verifiable encryption must be randomized for each session, which might be inefficient. Revocation by verifiable encryption is mentioned in specifications [5,20] without further details and supporting infrastructure description.

2.1 Our Contribution

In this paper, we propose the first scheme with practical revocation and malicious user identification which is deployable on off-line smart-cards. By adding the support of smart-cards, we allow the application of attribute-based credentials to eIDs. Furthermore, our scheme provides scalable revocation of particular privacy-enhancing features. This allows not only the revocation of credentials but also the revocation of untraceability and the revocation of anonymity of malicious users.

- **Revocation of Credentials**:
  - **Immediate Revocation**: there is no need to wait for the credential lifetime expiration, credentials can be revoked immediately.
  - **Issuer and Verifier Driven Revocation**: Revocation is available to both attribute issuers and verifiers. Any of these entities can initiate the revocation process.

- **Verifier Local Revocation (VLR)**[4]: valid users do not have to update their credentials or download any values after some other users are revoked.
- **Computationally Efficient Revocation**: computational complexity of the user's part of verification protocol does not depend on the number of revoked users.
- **Off-line Verification**: the verification session runs between the user and verifier only. There is no need to contact other parties.

The revocation of credentials is an extremely important feature but in some cases it is not enough just to revoke users from the system. In cases where damage was done, the service providers need a technique for learning the identity of attackers to make them responsible. We add more granularity to revocation in our scheme by allowing the revocation of particular privacy-enhancing features.

- **Revocation of Unlinkability**: in non-critical policy breaches, the verifier can inspect user's past behavior by revoking unlinkability. All past sessions of a particular user can be inspected without releasing his identity.
- **Revocation of Anonymity**: in critical policy breaches, it is possible to revoke the anonymity of a user to make him responsible for his acts.

We acknowledge that these revocation features must be strongly protected against a misuse. That is the reason why we spread the ability to do revocation over more entities. In our system, the issuer, verifier and a third authority must cooperate to revoke any privacy-enhancing feature. By such distribution, we limit the probability of misusing the revocation by a single authority. To provide more security (and user trust in our system), the third party can be distributed using multi-party computations. Moreover, the user has the freedom to choose his own attribute issuer among many commercial subjects, therefore he does not have to trust a fixed designated revocation authority but rather liberally chooses an entity he trusts most.

3 Proposed Attribute-Based Credential Scheme Architecture

In this section, the novel scheme for attribute-based credentials is proposed. The entities, general communication pattern and cryptographic design of underlying protocols are described in this section. The security analysis of the scheme is provided in Section 4.

3.1 Entities

There are four entities in the proposed scheme. Some of them are in possession of secret keys. The cryptographic construction of keys and their usage are described further in Chapter 3.4.

- **Issuer - I**: the entity who issues personal attributes to Users. Issuers also cooperate during the revocation of anonymity. All Issuer, Revocation Referee and Verifier must cooperate to reveal the identity of a malicious user. Issuer is equipped with a key K_I.
- **Revocation Referee - RR**: a single entity who generates system parameters *params*, cooperates with the Issuer during the attribute issuance and with Issuer and Verifier during the revocation of anonymity. RR works as a privacy guarantee because he decides about the type of revocation (credential revocation, unlinkablity revocation or anonymity revocation) based on the evidence provided by the Verifier. No entity is able to revoke without RR, yet RR is not a fully trusted party. RR cannot revoke or reveal any private information alone, only in cooperation with I and V. RR is equipped with a secret key K_{RR}.
- **User - U**: the entity who is in possession of a smart-card with issued attributes. The user can anonymously prove the attribute ownership by using the smart-card. Each smart-card has a secret master key K_U unique for each User needed for the attribute proof generation. Additionally, a secret session key K_S is generated for each verification session. The K_S key randomizes the sessions to make them completely unlinkable.
- **Verifier - V**: the entity who verifies User's attribute ownership (sometimes called relying party). Using the transcript of the verification session and the evidence of a rule breach, Verifiers can ask RR for revocation. If RR decides that revocation is rightful, User's master key can be anonymously revoked or, in more serious cases, identity of a malicious User can be disclosed. Verifiers need only pre-shared system parameters. They do not communicate with other parties during User verification (the process runs off-line).

3.2 General Overview of Proposed Scheme

The architecture of the proposed scheme, briefly introduced in [15], is depicted in Figure 1. The scheme is composed of four protocols - `Setup`, `IssueAtt`, `ProveAtt` and `Revoke`.

- $(params, K_{RR}, K_I) \leftarrow \texttt{Setup}(k, l, m)$: this algorithm is run by RR and Issuer. `Setup` inputs security parameters (k, l, m) and outputs system parameters *params*. RR's private output of the protocol is the K_{RR} key and I's private output is the K_I key.
- $K_U \leftarrow \texttt{IssueAtt}(params, K_I, K_{RR})$: the protocol outputs User's master key K_U. The master key is needed by the User for creating the attribute ownership proof in the `ProveAtt` protocol. By using advanced cryptographic techniques, the K_U is generated in such a way that only User's smart-card learns it although both RR and Issuer must contribute data and collaborate on K_U creation.

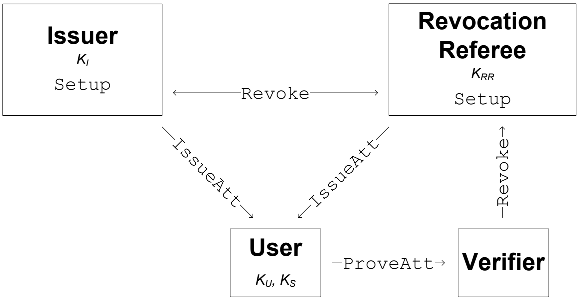


Fig. 1. Architecture of Proposed Scheme

- $proof$ ←ProveAtt($params, K_U$): using public system parameters and the K_U generated by the IssueAtt protocol, it is possible to build attribute $proof$ using the ProveAtt. For each $proof$, a unique session key K_S is generated by the User. The proof is anonymized and randomized by K_S. The protocol runs between the Verifier and User's smart-card. By ProveAtt, the User proves his ownership of attributes.
- rev ←Revoke($params, proof, K_{RR}, K_I$): in special cases (e.g., smart-card loss, theft or damage), the issued attributes can be revoked or even the malicious Users can be de-anonymized. In that case, the $proof$ transcript is sent by the Verifier to the RR with adequate evidence for revocation. RR evaluates the evidence and opens the $proof$ transcript using his K_{RR}. Depending on the type of revocation chosen by RR, the RR can either blacklist the attribute by publishing anonymous revocation information rev on a public blacklist or provide the Issuer with information necessary for User identification. The Issuer is then able to identify and charge the malicious User.

3.3 Used Cryptographic Primitives and Notation

The crucial building blocks of our scheme are: discrete logarithm commitments, Σ-protocols [10] for proofs of discrete logarithm knowledge and representation [8], proofs of discrete logarithm equivalence [8] and the Okamoto-Uchiyama trapdoor one-way function [19].

DL Commitments

To commit to a secret value $w \in \mathbb{Z}_q$, where q is a large prime, we use a simple computationally hiding and perfectly binding commitment. Let $p : q|p-1$ be a large prime and g a generator of order q in $\mathbb{Z}_p^*$. Then, $c = g^w \bmod p$ is a simple commitment scheme secure under the DL assumption. After publishing c, the secret w is computationally hidden (hiding property) but the committer is perfectly bound to his w (binding property) and unable to change w without changing c.

Σ-protocols

Σ-protocols [10] can be used for proving the knowledge of secrets and for proving the construction correctness without leaking additional information. We use the protocols described in [8] to prove the knowledge of a discrete logarithm (the protocol $PK\{\alpha : c = g^\alpha\}$), discrete logarithm equivalence (the protocol $PK\{\alpha : c_1 = g_1^\alpha \wedge c_2 = g_2^\alpha\}$) and discrete logarithm representation with respect to public generators (the protocol $PK\{(\alpha, \beta, \gamma) : c = g_1^\alpha g_2^\beta g_3^\gamma\}$). These protocols can be translated to full zero-knowledge protocols [11] thus they can be proven to leak no more information than intended. They can run non-interactively with computational security using [14]. With some restrictions, the protocols can be used in groups with hidden order by sending answers in $\mathbb{Z}$ [13]. Various types of proofs of knowledge and a framework for creating proofs can be found in [8].

Okamoto-Uchiyama Trapdoor One-Way Function

Let $n = r^2 s$ and r, s be large safe primes. Pick $g \in \mathbb{Z}_n$ such that $g \bmod r^2$ is a primitive element of $\mathbb{Z}_{r^2}^*$. Then $c = g^x \bmod n$ is a trapdoor one-way function with r as a trapdoor [19]. Value x can be computed using the trapdoor as $x = \dfrac{((c^{r-1} \bmod r^2) - 1)/r}{((g^{r-1} \bmod r^2) - 1)/r} \bmod r$. The function is secure if the factorization of n is hard. Size recommendations for n are the same as for RSA.

Notation

For various proofs of knowledge or representation, we use the efficient notation introduced by Camenisch and Stadler [8]. The protocol for proving the knowledge of discrete logarithm of c with respect to g is denoted as $PK\{\alpha : c = g^\alpha\}$. The proof of discrete log equivalence with respect to different generators g_1, g_2 is denoted as $PK\{\alpha : c_1 = g_1^\alpha \wedge c_2 = g_2^\alpha\}$. A signature by a traditional PKI (e.g., RSA) scheme of a user U on some data is denoted as $Sig_U(data)$. The symbol ":" means "such that", "|" means "divides", "$|x|$" is the bitlength of x and "$x \in_R \{0,1\}^l$" is a randomly chosen bitstring of maximum length l.

3.4 Cryptographic Specification of Protocols

$(params, K_{RR}, K_I) \leftarrow \texttt{Setup}(k, l, m)$ protocol: the goal of the protocol is to generate system parameters $params$, RR's revocation key K_{RR} and Issuer's key K_I. The protocol inputs security parameters k, l, m (k is the length of the challenge/hash function used, l relates to the length of Users' secrets, and m is the verification error parameter). The Issuer generates a group H defined by a large prime modulus p, generators h_1, h_2 of prime order $q : |q| = 2l$ and $q|p-1$. The Revocation Referee RR generates group G for the Okamoto-Uchiyama Trapdoor One-Way Function. G is defined by the modulus $n = r^2 s$ with r, s large primes ($|r| > 720, |r| > 4.5l, |n| \geq 2048$, $r = 2r' + 1$, $s = 2s' + 1$, r', s' are primes), generator $g_1 \in_R \mathbb{Z}_n^*$ of order $ord(g_1 \bmod r^2) = r(r-1)$ in $\mathbb{Z}_{r^2}^*$ and $ord(g_1) = rr's'$ in $\mathbb{Z}_n^*$. RR also randomly chooses its secrets $S_1, S_2, S_3 : |S_1| = 2.5l, |S_2| = l, |S_3^{-1} \bmod \phi(n)| = l$ and $GCD(S_1, \phi(n)) = GCD(S_2, \phi(n)) = GCD(S_3, \phi(n)) = 1$. Finally, RR computes an attribute seed $A_{seed} = g_1^{S_1} \bmod n$

(public, common for all Users, linked to a specific personal data type, e.g. citizenship) and values $g_2 = g_1^{S_2} \bmod n, g_3 = g_1^{S_3} \bmod n$. There might be more types of attribute seeds (different A_{seed_i}'s and S_{1_i}'s) related to different attributes Users want to prove. In that case, each unique A_{seed_i} represents one attribute, e.g. nationality, driving permission or legal voting age.[1] These seeds can be aggregated together by multiplying mod$n$. Thus, in general, the credential construction gathers more attributes. In the rest of the paper, we consider for simplicity only one A_{seed} in credential, making attribute and credential the same.

The values $q, p, h_1, h_2, n, g_1, g_2, g_3, A_{seed}$ are made public as system parameters $params$, while r, s, S_1, S_2, S_3 are securely stored at RR as K_{RR} key. Additionally, we use a traditional digital signature scheme (e.g., RSA). Issuers and Users are equipped with a private/public key-pair for digital signatures. This can be accomplished by existing techniques for PKI. The Issuer's private key represents the K_I.

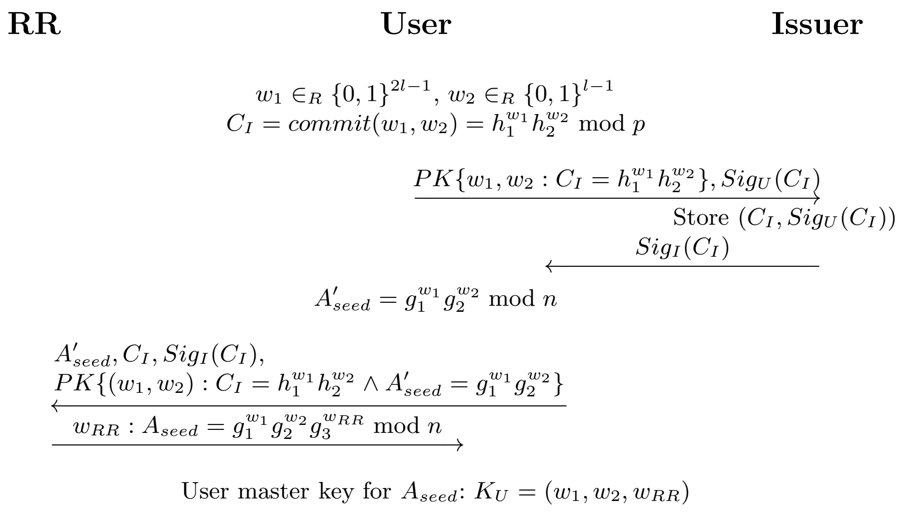

RR
User
Issuer
$w_1 \in_R \{0,1\}^{2l-1}$, $w_2 \in_R \{0,1\}^{l-1}$
$C_I = commit(w_1, w_2) = h_1^{w_1} h_2^{w_2} \bmod p$
$PK\{w_1, w_2 : C_I = h_1^{w_1} h_2^{w_2}\}, Sig_U(C_I)$
Store $(C_I, Sig_U(C_I))$
$Sig_I(C_I)$
$A'_{seed} = g_1^{w_1} g_2^{w_2} \bmod n$
$A'_{seed}, C_I, Sig_I(C_I)$,
$PK\{(w_1, w_2) : C_I = h_1^{w_1} h_2^{w_2} \wedge A'_{seed} = g_1^{w_1} g_2^{w_2}\}$
$w_{RR} : A_{seed} = g_1^{w_1} g_2^{w_2} g_3^{w_{RR}} \bmod n$
User master key for A_{seed}: $K_U = (w_1, w_2, w_{RR})$

Fig. 2. `IssueAtt` Protocol

$K_U \leftarrow$`IssueAtt`$(params, K_I, K_{RR})$ protocol: the first part of the `IssueAtt` protocol runs between the User's smart-card and the Issuer. The communication is not anonymous here, thus the Issuer can physically check the identity of the User, his other attributes etc. Then, User's smart-card generates User's contribution to the master key (w_1, w_2) and commits to these values. The commitment C_I is digitally signed[2] by the User and sent with an appropriate construction

[1] A public list of attributes and their assigned A_{seed_i}'s is maintained by RR. Additional A_{seed_i}'s can be computed and published dynamically, on demand from Issuers.

[2] Here, we rely on already established PKI, e.g., RSA signatures.

correctness proof $PK\{w_1, w_2 : C_I = h_1^{w_1} h_2^{w_2}\}$ to the Issuer. The Issuer checks the proof, the signature and replies with his digital signature on the commitment. In this phase, the User generated and committed to his key contribution. It will be used in all his future attribute proofs. The Issuer approved a new User by signing the committed key contribution.

The second part of the protocol runs between the User's smart-card and RR. In this phase, RR checks the signature of the Issuer on User's commitment C_I and computes his contribution w_{RR} to User's master key such that $A_{seed} = g_1^{w_1} g_2^{w_2} g_3^{w_{RR}} \bmod n$ holds. As a result, the User's smart-card learns all parts of the K_U, namely User's part (w_1, w_2) and RR's part w_{RR}. This triplet forms the discrete logarithm representation of the seed A_{seed} such that $A_{seed} = g_1^{w_1} g_2^{w_2} g_3^{w_{RR}} \bmod n$. This representation can be computed only in cooperation with RR (who knows the factorization of n). Although A_{seed} is shared among all Users as a system parameter, the triplet (w_1, w_2, w_{RR}) is unique for each user, since (w_1, w_2) is randomly generated by each User's smart-card and w_{RR} is generated by RR. Due to the discrete logarithm assumption, Users are stuck to their keys and they are unable to compute other valid keys without knowing K_{RR}. The master key K_U never leaves the smart-card and is stored in card's hardware-protected memory. All operations involving (w_1, w_2, w_{RR}) are computed on the card.

The second part of the `IssueAtt` protocol can be repeated to obtain keys for all demanded attributes. All attributes can be aggregated by multiplying $\bmod n$, keys are aggregated using plain addition. For simplicity, we describe the proof of only 1 attribute. The `IssueAtt` is depicted in Figure 2.

proof $\leftarrow$`ProveAtt`$(params, K_U)$ protocol: the protocol is used by User's smart-card to construct a proof about attribute ownership. In the protocol, the User proves the knowledge of his master key $K_U = (w_1, w_2, w_{RR})$. The session is randomized by a session key K_S. User creates a commitment C_2 to the session key K_S and proves its correctness. The protocol transcript forms the *proof* output. The protocol is illustrated in Figure 3 in CS notation and in Figure 4 in full.

RR **User** **Verifier**

$$A_{seed} = g_1^{w_1} g_2^{w_2} g_3^{w_{RR}} \bmod n$$
$$K_S \in_R \{0,1\}^l$$
$$A = A_{seed}^{K_S} \bmod n$$
$$C_1 = g_3^{K_S w_{RR}} \bmod n$$
$$C_2 = g_3^{K_S} \bmod n$$

$$\xrightarrow[\wedge A = A_{seed}^{K_S} \wedge C_1 = g_3^{K_S w_{RR}} \wedge C_2 = g_3^{K_S}\}]{PK\{(K_S, K_S w_1, K_S w_2, K_S w_{RR}) : A = g_1^{K_S w_1} g_2^{K_S w_2} g_3^{K_S w_{RR}}}$$

Fig. 3. `ProveAtt` Protocol in Camenisch-Stadler Notation

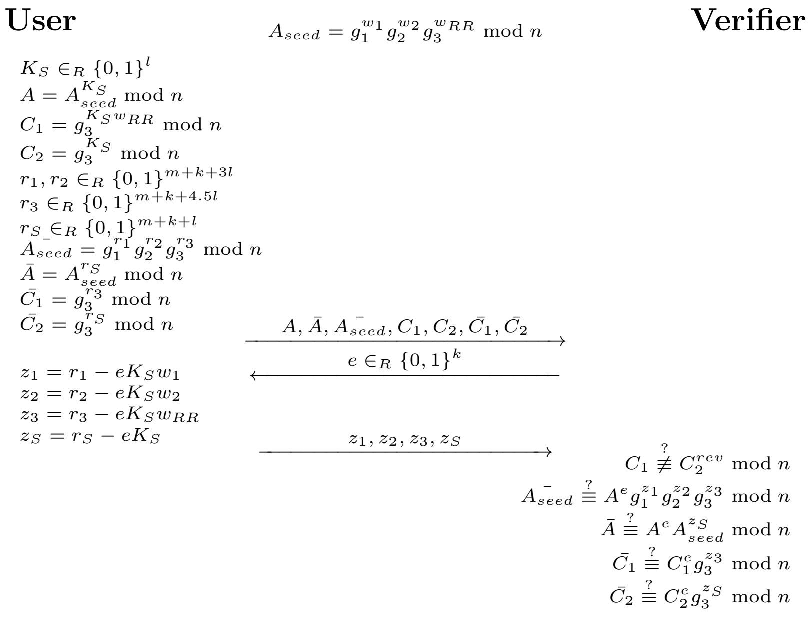

User
$A_{seed} = g_1^{w_1} g_2^{w_2} g_3^{w_{RR}} \bmod n$
Verifier
$K_S \in_R \{0,1\}^l$
$A = A_{seed}^{K_S} \bmod n$
$C_1 = g_3^{K_S w_{RR}} \bmod n$
$C_2 = g_3^{K_S} \bmod n$
$r_1, r_2 \in_R \{0,1\}^{m+k+3l}$
$r_3 \in_R \{0,1\}^{m+k+4.5l}$
$r_S \in_R \{0,1\}^{m+k+l}$
$\bar{A}_{seed} = g_1^{r_1} g_2^{r_2} g_3^{r_3} \bmod n$
$\bar{A} = A_{seed}^{r_S} \bmod n$
$\bar{C_1} = g_3^{r_3} \bmod n$
$\bar{C_2} = g_3^{r_S} \bmod n$
$\xrightarrow{A, \bar{A}, \bar{A}_{seed}, C_1, C_2, \bar{C_1}, \bar{C_2}}$
$\xleftarrow{e \in_R \{0,1\}^k}$
$z_1 = r_1 - eK_S w_1$
$z_2 = r_2 - eK_S w_2$
$z_3 = r_3 - eK_S w_{RR}$
$z_S = r_S - eK_S$
$\xrightarrow{z_1, z_2, z_3, z_S}$
$C_1 \overset{?}{\not\equiv} C_2^{rev} \bmod n$
$\bar{A}_{seed} \overset{?}{\equiv} A^e g_1^{z_1} g_2^{z_2} g_3^{z_3} \bmod n$
$\bar{A} \overset{?}{\equiv} A^e A_{seed}^{z_S} \bmod n$
$\bar{C_1} \overset{?}{\equiv} C_1^e g_3^{z_3} \bmod n$
$\bar{C_2} \overset{?}{\equiv} C_2^e g_3^{z_S} \bmod n$

Fig. 4. `ProveAtt` Protocol in detail

rev ←`Revoke`($params, proof, K_{RR}, K_I$) protocol: the protocol is executed if a User needs to be revoked from the system or if Verifier wants to reveal malicious users (and has a strong evidence for doing so). The transcript of the `ProveAtt` protocol can be forwarded to the RR entity in case of rule breaking. The RR entity can decide about the type of revocation. Credential revocation, unlinkability revocation or anonymity revocation are available.

Credential Revocation

RR knows the factorization of n thus he knows the trapdoor to the Okamoto-Uchiyama trapdoor function. From C_2, he learns the session key K_S and from C_1, his contribution w_{RR} to the User key K_U. RR can publish revocation information $rev = w_{RR}$ on a public blacklist. Then, each Verifier is able to check if the User is blacklisted or not by checking $C_1 \overset{?}{\equiv} C_2^{rev} \bmod n$. The equation holds only for revoked Users. Using this type of revocation, no identity is revealed and no valid users have to update their keys. The revocation does not influence non-revoked users in any sense. Verifiers only need to periodically download the blacklist with short rev values. Also Issuers can initiate the revocation, by sending C_I to RR who is able to link C_I to w_{RR}. The revocation information rev is then published by RR in the same way as if revocation was initiated by Verifiers.

Unlinkability Revocation

RR can reveal w_{RR} and w'_{RR} from two transcripts of the `ProveAtt` protocol. If $w_{RR} = w'_{RR}$, then the session has been carried out by the same User.

The revocation of unlinkability can be used by Verifiers to inspect past behavior of a suspected User. Again, a strong evidence of rule breach must be provided to RR who can either allow or reject unlinkability revocation.

Anonymity Revocation
Sometimes, it is necessary to identify malicious users. In that case, RR reveals w_{RR} and finds corresponding C_I since both values are linked by the IssueAtt protocol. C_I is then forwarded to Issuer who can de-anonymize the User since he has a database of digitally signed C_I's. The identification is non-repudiable since C_I is digitally signed and perfectly binds the User to the key inside.

4 Security Analysis

The ProveAtt protocol assures that legitimate attribute owners are accepted (completeness), dishonest Users are rejected (soundness) and that no additional information about Users is released (zero-knowledge).

Completeness
The honest Users know a valid representation of A_{seed} in the form (w_1, w_2, w_{RR}), so they are almost always accepted. There is a small verification error probability. Since a User does not know $\phi(n)$, he must send answers in proofs of knowledge/representation in $\mathbb{Z}$. Based on [1], to retain the zero-knowledge property, answers must fit within a certain interval, which happens with high probability $P = 1 - 2^{-m}$. Details in [1].

Soundness
The ProveAtt protocol is the parallel composition of a subprotocol denoted as $PK\{\alpha : c = g^\alpha\}$ described in Section 3.3. We prove its soundness by following the proof of the RSA variant of this protocol [7]. Our proof is adapted to the Okamoto-Uchiyama group which we use. The environment, already specified in sections devoted to setup and issuance, is following: $n = r^2 s, r = 2r' + 1, s = 2s' + 1, g \in_R \mathbb{Z}_n^* : ord(g \bmod r^2)$ in $\mathbb{Z}_{r^2}^*$ is $r(r-1), ord(g)$ in $\mathbb{Z}_n^*$ is $rr's'$ and r', s' are random large primes such that r, s are also primes.

Theorem 1. *Under the assumptions that factoring of n is hard and $\log_g c$ is unknown, given a modulus n, along with elements g, c, it is hard to compute integers a, b such that*

$$1 \equiv g^a c^b \bmod n \textit{ and } (a \neq 0 \textit{ or } b \neq 0). \tag{1}$$

Proof. Suppose there is an algorithm $\mathcal{A}$ that inputs n, g, c and outputs a, b valid in (1). Then we can use $\mathcal{A}$ to either factor n or compute $\log_g c$, both violating assumptions. The output (a, b) satisfies $1 \equiv g^a c^b \equiv g^a g^{\alpha b} \equiv g^{a+\alpha b} \bmod n$, therefore $a + \alpha b \equiv 0 \bmod ord(g)$. We have two cases:

Case 1. Let us consider $a + \alpha b = 0$. Then discrete logarithm $\log_g c$ can be efficiently computed as $\log_g c = \alpha = -\frac{a}{b}$. Case 1 violates the discrete logarithm assumption.

Case 2. Let us consider $a + \alpha b \neq 0$. $\mathcal{A}$ can be used to factor n by choosing α in random, inputting (n, g, g^α) and getting the output (a, b). Using $(a + \alpha b)$, which is a non-zero multiple of $\phi(n)/4$, the adversary can factor n. To efficiently compute a proper factor of n, the adversary can use the technique originally developed for RSA [3]. Case 2 violates the factorization assumption.

Using the Theorem 1, we can prove the soundness like in [7], thus by constructing the knowledge extractor and assuming that the factorization of n is hard. The extractor uses the standard rewinding technique, thus inputs two different valid answers $z, \bar{z}$ on two different challenges $e, \bar{e}$ with the fixed first step $\bar{c}$. The verification equation must hold for both answers: $\bar{c} \equiv g^z c^e \bmod n$ and $\bar{c} \equiv g^{\bar{z}} c^{\bar{e}} \bmod n$. From these two equations, we get $1 \equiv g^{z-\bar{z}} c^{e-\bar{e}} \bmod n$. From the Theorem 1, the User must have used $\log_g c$, since the factorization of n is unknown. Based on the Case 1 of Theorem 1, $(e - \bar{e})$ divides $(z - \bar{z})$, therefore the extractor can extract $\alpha = \frac{z-\bar{z}}{\bar{e}-e}$.

Zero-Knowledge

The subprotocol denoted $PK\{\alpha : c = g^\alpha\}$, as well as the whole `ProveAtt` protocol is composed of classical proof of knowledge Σ-protocols. The zero-knowledge protocol simulator can be constructed in the standard way [12], by choosing random answers and computing the first steps of the protocol from the verification equations. By constructing the Zero-Knowledge simulator, it is possible to prove that no additional information leaks from the protocol. For simplicity, we used challenge e from the Verifier in Figure 4, which would make the protocol secure only against honest Verifiers. Nevertheless, the protocol can be easily modified to become computationally secure against any Verifiers using [14] or fully secure using [11]. In our implementation, we use the Fiar-Shamir heuristic [14] to make the protocol non-interactive and computationally secure. By using a randomized zero-knowledge protocol for each `ProveAtt` session, no other information than the ownership validity is leaked. Thus, the User cannot be identified, traced or profiled.

5 Implementation Results

User's smart-card requires 9 modular exponentiations, 10 modular multiplications and 4 subtractions to construct an attribute proof. The scheme with 2048b modulus n has been implemented on both PC platform and smart-card platform. For PC implementation, we simulated the protocol in Mathematica software. A batch of 500 000 `ProveAtt` sessions has been successfully evaluated with the time of a single session under 61 ms (including both proof generation and verification) on a middle-class computer (2.53GHz Intel X3440 processor).

For the smart-card implementation, we chose .NET smart-cards and MultOS smart-cards. Gemalto .NET V2+ cards do not allow direct access to modular arithmetic operations [17], thus the time of verification is quite slow and comparable to idemix implementation [2]. We are able to reach the time of verification between 8 and 10 s which is impractical. Therefore, we implemented the scheme on the MultOS ML2-80K-65 cards. These cards allow a hardware acceleration of arithmetic operations through a cryptographic co-processor. With these cards, we are able to run the `ProveAtt` protocol in cca 2 s. Recently, we have measured only a proof-of-concept implementation. A major performance improvement is expected if the code is optimized.

6 Conclusion

In this paper, we present a novel scheme for revocable anonymous credentials. Using the scheme, a User can anonymously convince a Verifier about the possession of an attribute, typically about his age, citizenship or some authorization. By staying anonymous and having the control over all released data, users can protect their privacy during the verification process. Our scheme gathers all required features, so far supported only individually. The proposal is the first practical scheme implementable on off-line smart-cards.

Additionally, we add features unavailable before, mainly scalable off-line revocation and malicious user identity revelation. Finally, we present smart-card implementation results which show the scheme to be very practical and ready for commercial application.

In the proposal, we rely on smart-cards' tamper resistance during the generation and storing of User keys. This hardware-based protection against collusion attacks is sufficient for small-to-medium scale deployment. Our future task is to add cryptographic protection to make the scheme secure even on devices without hardware protection.

Acknowledgement. Research was sponsored by the Technology Agency of the Czech Republic project TA02011260 and the Ministry of Industry and Trade of the Czech Republic project FR-TI4/647.

References

1. Bao, F.: An efficient verifiable encryption scheme for encryption of discrete logarithms. In: Quisquater, J.-J., Schneier, B. (eds.) CARDIS 2000. LNCS, vol. 1820, pp. 213–220. Springer, Heidelberg (2000)
2. Bichsel, P., Camenisch, J., Groß, T., Shoup, V.: Anonymous credentials on a standard java card. In: Proceedings of the 16th ACM Conference on Computer and Communications Security, CCS 2009, pp. 600–610. ACM, New York (2009)
3. Boneh, D.: Twenty years of attacks on the rsa cryptosystem. Notices of the AMS 46, 203–213 (1999)
4. Boneh, D., Boyen, X., Shacham, H.: Short group signatures. In: Franklin, M. (ed.) CRYPTO 2004. LNCS, vol. 3152, pp. 41–55. Springer, Heidelberg (2004)
5. Camenisch, J., et al.: Specification of the identity mixer cryptographic library, Tech. rep. (2010)

6. Camenisch, J., Lysyanskaya, A.: Dynamic accumulators and application to efficient revocation of anonymous credentials. In: Yung, M. (ed.) CRYPTO 2002. LNCS, vol. 2442, pp. 61–76. Springer, Heidelberg (2002)
7. Camenisch, J., Shoup, V.: Practical verifiable encryption and decryption of discrete logarithms. In: Boneh, D. (ed.) CRYPTO 2003. LNCS, vol. 2729, pp. 126–144. Springer, Heidelberg (2003)
8. Camenisch, J., Stadler, M.: Proof systems for general statements about discrete logarithms. Tech. rep. (1997)
9. Chaum, D.: Security without identification: transaction systems to make big brother obsolete. Commun. ACM 28, 1030–1044 (1985)
10. Cramer, R.: Modular Design of Secure, yet Practical Cryptographic Protocols. Ph.D. thesis, University of Amsterdam (1996)
11. Cramer, R., Damgård, I., MacKenzie, P.: Efficient zero-knowledge proofs of knowledge without intractability assumptions. In: Imai, H., Zheng, Y. (eds.) PKC 2000. LNCS, vol. 1751, pp. 354–373. Springer, Heidelberg (2000)
12. Cramer, R., Damgård, I., Schoenmakers, B.: Proofs of partial knowledge and simplified design of witness hiding protocols. In: Desmedt, Y.G. (ed.) CRYPTO 1994. LNCS, vol. 839, pp. 174–187. Springer, Heidelberg (1994)
13. Damgård, I., Fujisaki, E.: A statistically-hiding integer commitment scheme based on groups with hidden order. In: Zheng, Y. (ed.) ASIACRYPT 2002. LNCS, vol. 2501, pp. 125–142. Springer, Heidelberg (2002)
14. Fiat, A., Shamir, A.: How to prove yourself: Practical solutions to identification and signature problems. In: Odlyzko, A.M. (ed.) CRYPTO 1986. LNCS, vol. 263, pp. 186–194. Springer, Heidelberg (1987)
15. Hajny, J., Malina, L.: Practical revocable anonymous credentials. In: De Decker, B., Chadwick, D.W. (eds.) CMS 2012. LNCS, vol. 7394, pp. 211–213. Springer, Heidelberg (2012)
16. Lapon, J., Kohlweiss, M., De Decker, B., Naessens, V.: Performance analysis of accumulator-based revocation mechanisms. In: Rannenberg, K., Varadharajan, V., Weber, C. (eds.) SEC 2010. IFIP AICT, vol. 330, pp. 289–301. Springer, Heidelberg (2010)
17. Malina, L., Hajny, J.: Accelerated Modular Arithmetic for Low-Performance Devices. In: 34th International Conference on Telecommunications and Signal Processing, pp. 131–135. IEEE (2011)
18. Naumann, I., Hogben, G.: Enisa: Privacy features of eid cards. Network Security Newsletter 2008, 9–13 (2008)
19. Okamoto, T., Uchiyama, S.: A new public-key cryptosystem as secure as factoring. In: Nyberg, K. (ed.) EUROCRYPT 1998. LNCS, vol. 1403, pp. 308–318. Springer, Heidelberg (1998)
20. Paquin, C.: U-prove cryptographic specification v1.1, Tech. rep. (2011)
21. The European Commission: Safer internet programme (2012), http://ec.europa.eu/information_society/activities/sip/policy/index_en.htm
22. The White House: National strategy for trusted identities in cyberspace (2011), http://www.whitehouse.gov/sites/default/files/rss_viewer/NSTICstrategy_041511.pdf

On the Use of Shamir's Secret Sharing against Side-Channel Analysis

Jean-Sébastien Coron[1], Emmanuel Prouff[2], and Thomas Roche[2]

[1] Tranef
jscoron@tranef.com
[2] ANSSI, 51, Bd de la Tour-Maubourg, 75700 Paris 07 SP, France
{firstname.name}@ssi.gouv.fr

Abstract. At CHES 2011 Goubin and Martinelli described a new countermeasure against side-channel analysis for AES based on Shamir's secret-sharing scheme. In the present paper, we exhibit a flaw in this scheme and we show that it is always theoretically broken by a first-order side-channel analysis. As a consequence of this attack, only a slight adaptation of the scheme proposed by Ben-Or *et al.* at STOC in 1988 can securely process multiplications on data shared with Shamir's technique. In the second part of this paper, we propose an improvement of this scheme that leads to a complexity $\tilde{\mathcal{O}}(d^2)$ instead of $\mathcal{O}(d^3)$, where d is the number of shares per data.

1 Introduction

The observation of a device during its execution (*e.g.* through power consumption measurements) can give information on the internal values actually manipulated by the device. Based on this idea, a powerful attack targeting symmetric cipher implementations called Differential Power Analysis (DPA for short) has been proposed by Kocher et al. in 1998 [14]. The main idea is to observe the device during the manipulation of key-dependent data (called *sensitive data* in the sequel), and to retrieve information about the key (and eventually the whole key) from this observation.

Since the introduction of DPA, and more generally of *Side Channel Analysis* (SCA for short), many works have focused either on the enhancement of such attacks or on the search of sound countermeasures. In the latter area of research, masking techniques are currently the most promising type of countermeasure. The idea is to split any sensitive variable manipulated by the device into several shares such that the knowledge of a subpart of the shares does not give information on the sensitive value itself. When the number of shares is $d+1$, the countermeasure is usually called a d^{th}*-order masking scheme*. In this case the attacker has to retrieve information about the $d+1$ shares — *i.e.* to observe at least $d+1$ leakage points on the device — in order to gain knowledge about the targeted sensitive variable. Such an attack is called a $(d+1)^{th}$*-order SCA attack* and it has been shown that its complexity increases exponentially with the order d [4]. While some 1^{st}-order masking techniques have been proved to

S. Mangard (Ed.): CARDIS 2012, LNCS 7771, pp. 77–90, 2013.

be secure against 1^{st}-order SCA attacks (see for instance [2,16]), the practicality of 2^{nd}-order attacks has been also demonstrated [15,17,27]. The construction of an efficient d^{th}-order masking scheme thus became of great interest. The main difficulty resides in the handling of $d+1$ shares of a unique intermediate variable through a non-linear function (*i.e.* the cipher s-boxes and more precisely the internal multiplications that are not squarings). We call this issue the *higher-order masking problem.*

State of the Art. The first scheme successfully dealing with the masking problem for any order d has been specified by Ishai, Sahai and Wagner in [12] for hardware implementations (where every internal operation is done over $\mathbb{F}_2$). In [21], this seminal result has been generalized from $\mathbb{F}_2$ to any finite field in the case of AES. It has subsequently been generalized to any block cipher in [3]. In parallel, Kim, Hong and Lim presented in [13] an improvement of [21]'s scheme which reduces, in the case of AES, the constant terms of the complexity for small orders d; it is based on the tower-field approach from [23]. The complexity of all those methods is $\mathcal{O}(d^2)$ where d is the masking order.

A common property of those previously cited works is that a Boolean masking is used to split the sensitive data. Namely, every sensitive variable A is assumed to be represented under the form of a $(d+1)$-tuple $(A_0, A_1, \cdots, A_d)$ such that $A = A_0 \oplus A_1 \oplus \cdots \oplus A_d$. However, other masking techniques have recently been investigated. On the one hand, Genelle, Prouff and Quisquater proposed in [9] a higher-order scheme based on the alternate use of Boolean masking and a *multiplicative masking* where the shares satisfy $A_0 \cdot A_1 \cdot \cdots \cdot A_d = A$. Its complexity is still $\mathcal{O}(d^2)$ but the constant terms are significantly reduced in the case study of AES. On the other hand, Goubin and Martinelli [11] and Prouff and Roche [20,22] have proposed to use Shamir's secret-sharing scheme to split the sensitive data. Starting from the same core observation as previous works [6,8], the authors' goal was to use a sharing technique with complex algebraic structure, in order to reduce the amount of sensitive information provided by the observation of the shares when involved *e.g.* in a correlation SCA. Additionally, the authors of [20,22] have shown that this way of sharing data enables the construction of masking schemes which thwart higher-order side-channel attacks *in the presence of hardware glitches.* The security argumentation is essentially based on a link which is established between the problematic of securing s-box processings against SCA in the presence of glitches and the Multi-Party Computation problematic. In both [11] and [20,22] the practical security gain is achieved at the cost of a complexity overhead which is $\mathcal{O}(d^3)$ instead of $\mathcal{O}(d^2)$ (for Boolean masking).

Our Contribution. In this paper, we first show that the secure multiplication scheme published in [11] is flawed and that a first-order SCA can always be successfully performed against it. Then, we show that the single remaining scheme to process secure multiplications between variables shared with Shamir's technique (namely the adaptation of [1] in the SCA context), can be improved to have complexity $\tilde{\mathcal{O}}(d^2)$ instead of $\mathcal{O}(d^3)$. This is essentially done by computing polynomial evaluations with a DFT instead of a naive evaluation.

2 Goubin and Martinelli's Schemes

2.1 Preliminaries

In this paper, random variables will be denoted by capital letters (*e.g.* A) and they take their values (realisations) in $GF(2^\ell)$. Realisations of a random variable will be denoted in small-case letters (*e.g.* a). Throughout this paper, we will make the (common) assumption that the side-channel leakage emanating from the manipulation of a variable can be rightfully modelled by a deterministic function of this variable and the addition of an independent Gaussian noise. Under this assumption, an implementation is said to be *secure against d^{th}-order SCA attacks* if it satisfies the following property [5,11,20,21,24].

Definition 1 ($d^{\mathbf{th}}$-order SCA security). *The implementation of an algorithm achieves d^{th}-order SCA security if no family of at most d intermediate variables is dependent on a sensitive variable.*

If a family of $j \leq d$ intermediate variables depends on a sensitive variable, then the implementation is said to have a *j^{th}-order flaw.*

Sharing/Masking. To achieve d^{th}-order security, a common countermeasure is to specify the implementation such that every sensitive variable is manipulated in a *$(d+1)^{th}$-order sharing* form. A classical choice is the Boolean masking, but other alternatives exist [9,11,20].

Masking Schemes. When the concept of $(d+1)^{\text{th}}$-order sharing is involved to protect an algorithm implementation, so-called *d^{th}-order masking schemes* are specified for each elementary operation (*e.g.* affine transformations or field multiplications). They aim at specifying how to build the sharing of the operation output from the sharing of the input(s), without introducing any j^{th}-order flaw with $j \leq d$.

In this paper we focus on a particular higher-order sharing based on Shamir's secret sharing [25]. For this technique, two families of masking schemes have been proposed in [11] and [20,22] respectively. We recall some of them along with the outlines of the sharing process itself in the next section.

2.2 Shamir's Secret Sharing Scheme

In a seminal paper [25], Shamir has introduced a simple and elegant way to split a secret $A \in GF(2^\ell)$ into n shares such that no tuple of shares with cardinality lower than a so-called *threshold* $d < n$ depends on A. Shamir's protocol consists in generating a degree-d polynomial with coefficients randomly generated in $GF(2^\ell)$, except the constant term which is always fixed to A. In other terms, Shamir proposes to associate A with a polynomial $P_A(X)$ defined such that $P_A(X) = A + \sum_{i=1}^{d} u_i X^i$, where the u_i denote random coefficients. Then, n distinct non-zero elements $\alpha_0, \ldots, \alpha_{n-1}$ are publicly chosen in $GF(2^\ell)$ and the polynomial $P_A(X)$ is evaluated in the α_i to construct a so-called (n,d)-*sharing* $(A_0, A_1, \cdots, A_{n-1})$ of A such that $A_i = P_A(\alpha_i)$ for every $i \in [0; n-1]$.

To re-construct A from its sharing, polynomial interpolation is first applied to re-construct $P_A(X)$ from its n evaluations A_i. Then, the polynomial is evaluated in 0. Those two steps indeed leads to the recovery of A since, by construction, we have $A = P_A(0)$. Actually, using Lagrange's interpolation formula, the two steps can be combined in a single one thanks to the following equation:

$$A = \sum_{i=0}^{n-1} A_i \cdot \beta_i \ , \tag{1}$$

where the constants β_i are defined as follows:

$$\beta_i := \prod_{k=0, k \neq i}^{n-1} \frac{\alpha_k}{\alpha_i + \alpha_k} \ .$$

Remark 1. The β_i can be precomputed once for all and will hence be considered as public values in the following.

Notation. The value β_i will sometimes be considered as the evaluation in 0 of the polynomial:

$$\beta_i(x) := \prod_{k=0, k \neq i}^{n-1} \frac{x + \alpha_k}{\alpha_i + \alpha_k} \ .$$

2.3 Multiplication of Shares

To define a d^{th}-order masking scheme for a block cipher implementation where each intermediate result is split with Shamir's technique, one must specify a secure method for the processing of field multiplications over $GF(2^\ell)$. Recently, two papers have been published on this issue respectively by Goubin and Martinelli [11] and by Prouff and Roche [20]. Both of them start from a multiplication protocol introduced by Ben-Or *et al.* in the context of the Multy-Party Computation Theory [1]. For this protocol to work, the number of shares n per variable must be at least $2d+1$ and for $n = 2d+1$, it is proved that it satisfies a security property encompassing the d^{th}-order SCA security. Whereas [20] is a straightforward rewriting of Ben-Or *et al.* (BGW) protocol for the SCA context, the scheme in [11] may be viewed as an efficiency improvement attempt in the SCA context. It is called GM protocol in the following. We recall hereafter the two solutions.

2.4 BGW Protocol in the SCA Context

Let us assume that A and B are two variables in $GF(2^\ell)$ that have been (n,d)-shared into $(A_i)_i$ and $(B_i)_i$ respectively, by evaluating the secret polynomials $P_A(X) = A + \sum_{1 \leq j \leq d} u_j X^j$ and $P_B(X) = B + \sum_{1 \leq j \leq d} v_j X^j$ in the public points α_i for $1 \leq i \leq n$. We give hereafter the adaptation of [1] in the SCA context as proposed in [20,22][1].

Algorithm 1. BGW's Secure Multiplication

INPUT: two integers n and d such that $n \geq 2d+1$, the (n,d)-sharings $(A_i)_i = (P_A(\alpha_i))_i$ and $(B_i)_i = (P_B(\alpha_i))_i$ of A and B respectively.
OUTPUT: the (n,d)-sharing $(P_C(\alpha_i))_i$ of $C = A \cdot B$.
PUBLIC: the n distinct points α_i, the interpolation values $(\beta_0, \cdots, \beta_{n-1})$

```
for $i = 1$ to $n$
    do $W_i \leftarrow P_A(\alpha_i) \cdot P_B(\alpha_i)$

*** Compute a sharing $(Q_i(\alpha_j))_{j \leq d}$ of $W_i$ with $Q_i(X) = W_i + \sum_{j=1}^{d} a_j \cdot X^j$
    for $j = 1$ to $d$ do $a_j \leftarrow \texttt{rand}(GF(2^\ell))$
    for $j = 1$ to $n$ do $Q_i(\alpha_j) \leftarrow W_i + \sum_{k=1}^{d} a_k \cdot \alpha_j^k$

*** Compute the share $C_i = P_C(\alpha_i)$ for $C = A \cdot B$
for $i = 1$ to $n$
    do $C_i \leftarrow \sum_{j=1}^{n} Q_j(\alpha_i) \cdot \beta_j$.
return $(C_i)_i$
```

The completeness of Algorithm 1 is discussed in [1]. Its d^{th}-order SCA security can be straightforwardly deduced from the proof given by Ben-Or *et al.* in [1] in the secure multi-party computation context. Eventually, for $n = 2d+1$ (which is the parameter choice which optimizes the security/efficiency overhead), the complexity of Algorithm 1 in terms of additions and multiplications is $\mathcal{O}(d^3)$.

2.5 GM Multiplication Protocol

The scheme proposed in [11] has the same asymptotic complexity as BGW but with much smaller constant terms. Indeed, the functional condition $n \geq 2d+1$ is replaced by the minimal one $n \geq d+1$ which enables to process the multiplication on $(d+1,d)$-sharings of A and B (instead of $(2d+1,d)$-sharings).

[1] The protocol is an improved version of the protocol originally proposed by Ben-Or *et al.* [1], due to Gennaro *et al.* in [10].

We recall hereafter the proposal in [11] with the notations $\beta_{j,k}(X)$ standing for the polynomials $\beta_j(X)\cdot\beta_k(X)$ truncated by removing the terms of degree strictly greater than d.

Algorithm 2. Goubin and Martinelli's Secure Multiplication

Input: the $(d+1,d)$-sharings $(A_i)_i$ and $(B_i)_i$ of A and B respectively.
Output: the $(d+1,d)$-sharing $(C_i)_i$ of $C = A\cdot B$.
Public: the public elements α_i and the public polynomials $\beta_{j,k}(X)$.

```
1. for $j = 0$ to $d$
2.     for $k = 0$ to $d$
3.         do $t_{j,k} \leftarrow A_j \cdot B_k$
4. for $i = 0$ to $d$
5.     do $C_i \leftarrow \sum_{j=1}^{d}\sum_{k=0}^{j-1}(t_{j,k}+t_{k,j})\cdot\beta_{j,k}(\alpha_i) + \sum_{j=0}^{d} t_{j,j}\cdot\beta_{j,j}(\alpha_i)$
6. return $(C_i)_i$
```

The completeness of Algorithm 2 is argued in [11]. Here, we only point out that the fifth step may be rewritten:

$$C_i = \sum_{j=0}^{d}\sum_{k=0}^{d} t_{j,k}\cdot\beta_{j,k}(\alpha_i) = \sum_{j=0}^{d}\sum_{k=0}^{d} A_j\cdot B_k\cdot\beta_{j,k}(\alpha_i)\ , \tag{2}$$

in which the evaluation in α_i of the degree-d part of the polynomial $P_C(X) = P_A(X)\cdot P_B(X)$ can be clearly recognized. Hence, (2) can be written:

$$C_i = \left(P_A(X)\cdot P_B(X)\right)_{|d}(\alpha_i)\ , \tag{3}$$

where the notation $(\cdot)_{|d}$ stands for the polynomial truncation obtained by suppressing all the monomials of degree strictly greater than d.

In [11], the authors assume that Algorithm 2 satisfies d^{th}-order SCA security and let the proof for future work. In the next section, we invalidates this assumption by exhibiting a first-order flaw which occurs whatever the input order d of the algorithm.

3 Attack against GM Protocol

Hereafter we show that Goubin and Martinelli's Algorithm 2 always has a first-order flaw whatever the masking order d. For clarity reasons, we first exhibit the flaw for $d = 1$ and generalize it afterward. We moreover give, in Annex A, an information theoretic evaluation of this first-order leakage for $d = 1$ and $d = 2$.

Attack Description for $d = 1$. In this case, (3) becomes:

$$C_i = A \cdot B + A \cdot V \cdot \alpha_i + B \cdot U \cdot \alpha_i \ , \tag{4}$$

where we denoted by U and V the random variables associated to the coefficients of the non-constant monomials in P_A and P_B respectively. By construction, those coefficients have been randomly generated and we hence assume that both U and V have a uniform distribution.

When $A = B = 0$, it can be checked that C_i is always null. Otherwise, if $(A, B) \neq (0, 0)$, say $A \neq 0$ w.l.o.g., then the term $A \cdot V \cdot \alpha_i$ is not null (since $\alpha_i \neq 0$ by construction) and it depends neither on $B \cdot U \cdot \alpha_i$ nor on $A \cdot B$. As a consequence, C_i always follows an uniform distribution when $(A, B) \neq (0, 0)$. We hence deduce that C_i leaks information on (A, B) (whether it is null or not) and hence that the first-order countermeasure has a flaw.

More generally, we state in the following proposition that such first-order flaw exists for any masking order d.

Proposition 1. *Algorithm 2 always has a first-order flaw for any input parameter d.*

Proof. The flaw in Algorithm 2 has already been exhibited for $d = 1$. In the rest of the proof, we hence assume $d > 1$ and we show that, even in this case, a flaw can be exhibited. By developing (3) we get:

$$C_i = A \cdot B + \sum_{j=1}^{d} A \cdot V_j \cdot \alpha_i^j + \sum_{j=1}^{d} B \cdot U_j \cdot \alpha_i^j + \sum_{j=1}^{d-1} \sum_{k=1}^{d-j} U_j \cdot V_k \cdot \alpha_i^{j+k} \ , \tag{5}$$

where we denoted by U_j and V_k the random variables associated to the coefficients of the non-constant monomials in P_A and P_B respectively. Thanks to the law of total probability, for every $(a, b) \in GF(2^\ell)^2$ the probability $\Pr[C_i | A = a, B = b]$ satisfies:

$$\Pr[C_i | A = a, B = b] = 2^{-\ell d} \sum_{\boldsymbol{u} \in GF(2^\ell)^d} \Pr[C_i(a, b, \boldsymbol{u})] \ , \tag{6}$$

where $C_i(a, b, \boldsymbol{u})$ denotes $(C_i \mid A = a, B = b, \boldsymbol{U} = \boldsymbol{u})$ and $\boldsymbol{U}$ refers to $(U_1, \cdots, U_d)$.

Let us focus on $C_i(a, b, \boldsymbol{u})$. By definition, it satisfies:

$$C_i(a, b, \boldsymbol{u}) = a \cdot b + \sum_{j=1}^{d} b \cdot u_j \cdot \alpha_i^j + a \cdot \alpha_i^d \cdot V_d + \sum_{j=1}^{d-1} V_j \cdot \alpha_i^j \cdot (a + \sum_{k=1}^{d-j} u_k \cdot \alpha_i^k) \ . \tag{7}$$

It can hence be viewed as an affine combination of random variables V_j that all have uniform distribution and are mutually independent (by construction of the polynomial P_A). This linear combination always contains the term $\alpha_i^d \cdot a \cdot V_d$ which is independent of the other ones and has an uniform distribution as long as a is non-zero (since α_i^d itself is non-zero). Based on this observation, we can split our analysis into two cases related to the condition $a = 0$.

If $a \neq 0$, then $C_i(a, b, \boldsymbol{u})$ has uniform distribution for every b and every $\boldsymbol{u}$. This implies that $\Pr[C_i|A = a, B = b]$ is an uniform distribution.

If $a = 0$, then a sufficient condition for $C_i(a, b, \boldsymbol{u})$ to be uniform is that at least one of the terms $\sum_{k=1}^{d-j} u_k \cdot \alpha_i^k$ is non-zero when j ranges from 1 to $d - 1$ which is equivalent with $(u_1, \cdots, u_{d-1}) \neq (0, \cdots, 0)$ since the α_i^j are all non-zero (by construction). When this sufficient condition is not satisfied, *i.e.* when $(u_1, \cdots, u_{d-1}) = (0, \cdots, 0)$, then we have:

$$\Pr[C_i|A = 0, B = b, u_1 = 0, \cdots, u_{d-1} = 0] = \Pr[b \cdot \alpha_i^d \cdot U_d] \ .$$

We deduce that, if $b \neq 0$, then $\Pr[C_i|A = 0, B = b, (u_1, \cdots, u_{d-1}) \neq (0, \cdots, 0)]$ and $\Pr[C_i|A = 0, B = b, (u_1, \cdots, u_{d-1}) = (0, \cdots, 0)]$ are both uniform, which implies (due to the law of total probability[2]) that $\Pr[C_i|A = 0, B = b]$ is uniform. On the other hand, if $b = 0$, then the variable $(C_i|A = 0, B = b, u_1 = 0, \cdots, u_{d-1} = 0)$ is constant and its distribution is the function which is zero everywhere except in 0 where it takes the value 1. Eventually for $(a, b) = (0, 0)$ we get:

$$\Pr[C_i = c|A = a, B = b] = \begin{cases} \frac{1}{2^\ell} - \frac{1}{2^{\ell d}} & \text{if } c \neq 0 \\ \frac{1}{2^\ell} + \frac{2^\ell - 1}{2^{\ell d}} & \text{if } c = 0 \end{cases} ,$$

which implies that the distribution $\Pr[C_i|A = 0, B = 0]$ is non-uniform. This concludes the proof since it shows that the distribution of C_i depends on the value of the pair of sensitive variables A and B. □

Remark 2. It can be observed that the distance between the two distributions that can take C_i decreases as the order increases and they actually merge when the order tends to infinity.

4 Improvement Proposal

In this section we describe a simple improvement of Algorithm 1 so that the complexity of the secure multiplication algorithm becomes $\tilde{\mathcal{O}}(d^2)$ instead of $\mathcal{O}(d^3)$. In Algorithm 1, the $\mathcal{O}(d^3)$ complexity comes from Step 4; namely this corresponds to the evaluation of a polynomial of degree d at n points α_i, which takes $\mathcal{O}(n \cdot d)$ times; since Step 4 is performed n times, the full complexity is then $\mathcal{O}(n^2 \cdot d) = \mathcal{O}(d^3)$.

Thanks to the Discrete Fourier Transform (DFT), the evaluation of a polynomial of degree $d < n$ at n points can actually be computed in time $\tilde{\mathcal{O}}(n)$ instead of $\mathcal{O}(n^2)$. Therefore the full complexity of the algorithm becomes $\tilde{\mathcal{O}}(n^2) = \tilde{\mathcal{O}}(d^2)$ instead of $\mathcal{O}(d^3)$.

[2] Th law of total probability states that for any r.v. X and any tuple of n r.v. $(Y_i)_i$, we have $\Pr[X] = \sum_i \Pr[X \mid Y_i] \Pr[Y_i]$.

In the following we describe, in the context of our SCA secure multiplication problematic, the fast evaluation algorithm based on the DFT. In the similar context of Multi-Party Computation, such improvement was already known (see *e.g.* [7]).

4.1 Fast Polynomial Evaluation Based on DFT

Let ω a primitive n^{th} root of unity in $GF(2^\ell)$ with $n = 2^\ell - 1$. The n points α_i are defined as $\alpha_i = \omega^i$ for $0 \leq i < n$. For simplicity we restrict ourselves to a finite field of characteristic 2; however the algorithm can be generalized to any characteristic.

Given as input a polynomial

$$a(x) = \sum_{j=0}^{n-1} a_j \cdot x^j \quad , \tag{8}$$

the algorithm described hereafter aims at efficiently process the values $a(\omega^i)$ for all $0 \leq i < n$. Noting that: $a(\omega^i) = a(x) \bmod (x - \omega^i)$, the values $a(\omega^i)$ can be computed using a remainder tree. As illustrated in Figure 1, the polynomial is progressively reduced modulo the polynomials $u_{i,j}(x)$, starting from the root polynomial:

$$u_{\ell,0}(x) = (x - 0) \cdot \prod_{i=0}^{n-1} (x - \omega^i) = x^{2^\ell} - x$$

downto the leaf polynomials $(x - \omega^i)$.

The DFT polynomial evaluation can actually be still improved by optimizing the ordering of the leafs ω^i so that the intermediate remainder polynomials $u_{i,j}(x)$ have a special form that enables fast modular reduction. It is shown in [28, page 573] that there exists an ordering $(\beta_0, \beta_1, \ldots, \beta_{2^\ell - 1})$ of all the elements of $GF(2^\ell)$, such that if $u_{0j}(x) := x - \beta_j$, and for $1 \leq i \leq \ell$,

$$u_{ij}(x) = u_{i-1,\,2j}(x) \cdot u_{i-1,\,2j+1}(x), \qquad 0 \leq j < 2^{\ell - i},$$

then each polynomial $u_{i0}(x)$ is an i^{th}-order linearized polynomial:

$$u_{i0}(x) = \sum_{k=0}^{i} v_{ik} \cdot x^{2^k},$$

and each polynomial $u_{ij}(x)$, $j \neq 0$, is an affine polynomial and is related to $u_{i0}(x)$ by $u_{ij}(x) = u_{i0}(x) + c_{ij}$, for some constants $c_{ij} \in GF(2^\ell)$. Since each polynomial $u_{ij}(x)$ has at most $i + 2$ non-zero terms (instead of at most $2^i + 1$ for a polynomial of degree 2^i), modular reduction can be done in time quasi-linear in the degree of $u_{ij}(x)$, instead of quadratic time (see below).

Formally, the DFT computation is defined as follows:

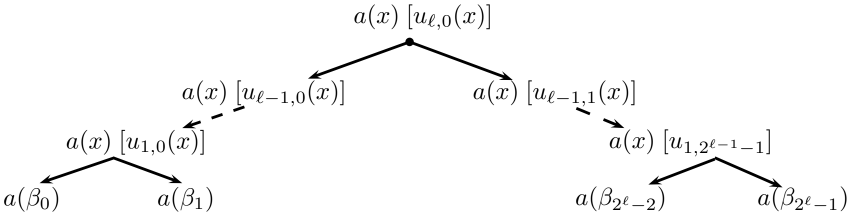


Fig. 1. Remainder tree for the computation of $a(\beta_i)$, for $0 \leq i < 2^\ell$

Algorithm 3. DFT Computation

INPUT: a polynomial $a(x) = \sum_{i=0}^{n-1} a_i \cdot x^i$ over $GF(2^\ell)$, where $n = 2^\ell - 1$.
OUTPUT: the field elements $a(\beta_i)$, for $0 \leq i < 2^\ell$
PUBLIC: the public polynomials $u_{ij}(x)$

```
1. Let $a_{\ell 0}(x) \leftarrow a(x)$
2. for $\lambda$ from $\ell - 1$ to $0$
3.     for $j = 0$ to $2^{\ell-\lambda} - 1$
4.         for $k = 0$ to $1$
5.             do $a_{\lambda,\,2j+k}(x) \leftarrow a_{\lambda+1,\,j}(x) \bmod u_{\lambda,\,2j+k}(x)$
6. Return $a_{0,\,j}$ for $0 \leq j \leq 2^\ell$
```

When applied to process the fourth step of Algorithm 1, the polynomial $a(x)$ in (8) corresponds to $Q_i(x)$ (associating each coefficient of $a(x)$ to the corresponding coefficients in $Q_i(x)$ and setting $a_j = 0$ for all the indices $j \in [d+1; n-1]$) and the n public points α_j are assumed to be chosen equal to ω^j. In such a setting, it can then be checked that the leaf polynomials computed by the fast evaluation method described here correspond to $a(\omega^j) = Q_i(\alpha_j)$ as expected (with the special case $a(0)$ which gives the already known value W_i at Step 4 of Algorithm 1).

4.2 Complexity Analysis and Parameters' Choice

We note that the degree of the polynomials $u_{\lambda,j}$ is 2^λ, which implies that the degree of the polynomials $a_{\lambda,j}$ is at most $2^\lambda - 1$. Since the polynomials $u_{\lambda,j}$ contain at most $\lambda + 2$ non-zero terms, every modular polynomial reduction at Step 5 has complexity $\mathcal{O}(\lambda 2^\lambda) = \mathcal{O}(\ell 2^\lambda)$. For a fixed level λ there are $2 \cdot 2^{\ell-\lambda}$ such reductions, which gives a total complexity $\mathcal{O}(\ell 2^\ell)$ for a given level λ. Since there are ℓ levels the full complexity of Algorithm 3 is $\mathcal{O}(\ell^2 \cdot 2^\ell)$. With $n = 2^\ell - 1$, we obtain a complexity $\mathcal{O}(n \cdot \log^2 n) = \tilde{\mathcal{O}}(n)$ as required.[3] Note that smaller values of n are possible; namely it suffices that $n | 2^\ell - 1$. For example for AES with $GF(2^8)$, since $2^8 - 1 = 3 \cdot 5 \cdot 17$, we can take $n = 3, 5, 15, 17, 51, 85, 255$.

To select a larger value of n for a fixed field size $GF(2^\ell)$, it suffices to work in an extension field $GF(2^{\ell \cdot s})$ of $GF(2^\ell)$ for $s > 1$; then one can take $n = 2^{\ell \cdot s} - 1$. The complexity of Algorithm 3 is still $\mathcal{O}(n \cdot \log^2 n)$ operations in the extension field $GF(2^{\ell \cdot s})$. Each operation in $GF(2^{\ell \cdot s})$ can be computed with $\mathcal{O}(s^2) = \mathcal{O}(\log^2 n)$ operations in $GF(2^\ell)$. Therefore the complexity of Algorithm 3 becomes $\mathcal{O}(n \cdot \log^4 n)$, which is still $\tilde{\mathcal{O}}(n)$.

4.3 Security of the Improved Multiplication Algorithm

Since in Algorithm 1 this polynomial evaluation step is performed n times, the full complexity becomes $\mathcal{O}(n^2 \cdot \log^4 n) = \tilde{\mathcal{O}}(n^2)$ instead of $\mathcal{O}(n^3)$. In the multiparty computation setting, the new algorithm is still secure against a coalition of up to $t < n/2$ players; namely the polynomial evaluation step at Step 4 in Algorithm 1 is performed *locally* by each player; therefore changing the polynomial evaluation algorithm does not modify the security property of the algorithm. In the context of side-channel analysis, with $n = 2d + 1$, the algorithm is therefore still secure against a d-th order attack.

5 Conclusion

Several works argued on the importance of identifying new sharing techniques that minimize the amount of sensitive information extractable from the family of shares in a SCA context. This is indeed of particular importance since such a sharing, combined with noise, would be able to resist to any higher-order side-channel attack in practice, even when parametrized with small sharing orders (*e.g.* 2 or 3). The polynomial sharing introduced by Shamir is a promising candidate. However, it remains to define efficient algorithms able to operate on data shared with this technique without introducing key-dependent leakages of order lower than the sharing order. Until this work, there were essentially two algorithm proposals to securely perform a multiplication between two shared data: one proposed by Goubin and Martinelli at CHES 2011 and one adapted from an algorithm by Ben-Or *et al.* at STOC in 1988. In the present paper, we showed that the first proposal is flawed (more precisely is always broken by a first-order SCA) and we improved the complexity of the second proposal from $\mathcal{O}(d^3)$ to $\tilde{\mathcal{O}}(d^2)$, where d is the number of shares per data. We think that those results are a first promising step toward efficient methods to process on data shared with Shamir's secret sharing in embedded systems.

References

1. Ben-Or, M., Goldwasser, S., Wigderson, A.: Completeness theorems for non-cryptographic fault-tolerant distributed computation (extended abstract). In: Simon, J. (ed.) STOC, pp. 1–10. ACM (1988)

[3] We write $f(\lambda) = \tilde{\mathcal{O}}(g(\lambda))$ if $f(\lambda) = \mathcal{O}(g(\lambda) \log^k g(\lambda))$ for some $k \in \mathbb{N}$.

2. Blömer, J., Guajardo, J., Krummel, V.: Provably Secure Masking of AES. In: Handschuh, H., Hasan, M.A. (eds.) SAC 2004. LNCS, vol. 3357, pp. 69–83. Springer, Heidelberg (2004)
3. Carlet, C., Goubin, L., Prouff, E., Quisquater, M., Rivain, M.: Higher-order masking schemes for S-boxes. In: Canteaut, A. (ed.) FSE 2012. LNCS, vol. 7549, pp. 366–384. Springer, Heidelberg (2012)
4. Chari, S., Jutla, C., Rao, J., Rohatgi, P.: A Cautionary Note Regarding Evaluation of AES Candidates on Smart-Cards. In: Second AES Candidate Conference – AES 2 (March 1999)
5. Coron, J.-S., Prouff, E., Rivain, M.: Side Channel Cryptanalysis of a Higher Order Masking Scheme. In: Paillier, P., Verbauwhede, I. (eds.) CHES 2007. LNCS, vol. 4727, pp. 28–44. Springer, Heidelberg (2007)
6. Courtois, N., Goubin, L.: An Algebraic Masking Method to Protect AES against Power Attacks. In: Won, D., Kim, S. (eds.) ICISC 2005. LNCS, vol. 3935, pp. 199–209. Springer, Heidelberg (2006)
7. Damgård, I., Ishai, Y., Krøigaard, M., Nielsen, J.B., Smith, A.: Scalable multiparty computation with nearly optimal work and resilience. In: Wagner, D. (ed.) CRYPTO 2008. LNCS, vol. 5157, pp. 241–261. Springer, Heidelberg (2008)
8. Fumaroli, G., Martinelli, A., Prouff, E., Rivain, M.: Affine masking against higher-order side channel analysis. In: Biryukov, A., Gong, G., Stinson, D.R. (eds.) SAC 2010. LNCS, vol. 6544, pp. 262–280. Springer, Heidelberg (2011)
9. Genelle, L., Prouff, E., Quisquater, M.: Thwarting Higher-Order Side Channel Analysis with Additive and Multiplicative Maskings. In: Preneel, Takagi (eds.) [19], pp. 240–255
10. Gennaro, R., Rabin, M.O., Rabin, T.: Simplified vss and fact-track multiparty computations with applications to threshold cryptography. In: PODC, pp. 101–111 (1998)
11. Goubin, L., Martinelli, A.: Protecting aes with shamir's secret sharing scheme. In: Preneel, Takagi (eds.) [19], pp. 79–94
12. Ishai, Y., Sahai, A., Wagner, D.: Private Circuits: Securing Hardware against Probing Attacks. In: Boneh, D. (ed.) CRYPTO 2003. LNCS, vol. 2729, pp. 463–481. Springer, Heidelberg (2003)
13. Kim, H., Hong, S., Lim, J.: A Fast and Provably Secure Higher-Order Masking of AES S-Box. In: Preneel, Takagi (eds.) [19], pp. 95–107
14. Kocher, P., Jaffe, J., Jun, B.: Introduction to Differential Power Analysis and Related Attacks. Technical report, Cryptography Research Inc. (1998)
15. Oswald, E., Mangard, S., Herbst, C., Tillich, S.: Practical Second-order DPA Attacks for Masked Smart Card Implementations of Block Ciphers. In: Pointcheval (ed.) [18], pp. 192–207
16. Oswald, E., Mangard, S., Pramstaller, N., Rijmen, V.: A Side-Channel Analysis Resistant Description of the AES S-box. In: Gilbert, H., Handschuh, H. (eds.) FSE 2005. LNCS, vol. 3557, pp. 413–423. Springer, Heidelberg (2005)
17. Peeters, E., Standaert, F.-X., Donckers, N., Quisquater, J.-J.: Improved Higher-order Side-Channel Attacks with FPGA Experiments. In: Rao, J.R., Sunar, B. (eds.) CHES 2005. LNCS, vol. 3659, pp. 309–323. Springer, Heidelberg (2005)
18. Pointcheval, D. (ed.): CT-RSA 2006. LNCS, vol. 3860. Springer, Heidelberg (2006)
19. Preneel, B., Takagi, T. (eds.): CHES 2011. LNCS, vol. 6917. Springer, Heidelberg (2011)
20. Prouff, E., Roche, T.: Higher-order glitches free implementation of the aes using secure multi-party computation protocols. In: Preneel, Takagi (eds.) [19], pp. 63–78

21. Rivain, M., Prouff, E.: Provably Secure Higher-Order Masking of AES. In: Mangard, S., Standaert, F.-X. (eds.) CHES 2010. LNCS, vol. 6225, pp. 413–427. Springer, Heidelberg (2010)
22. Roche, T., Prouff, E.: Higher-order glitch free implementation of the aes using secure multi-party computation protocols. Journal of Cryptographic Engineering, 1–17 (2012)
23. Satoh, A., Morioka, S., Takano, K., Munetoh, S.: A Compact Rijndael Hardware Architecture with S-Box Optimization. In: Boyd, C. (ed.) ASIACRYPT 2001. LNCS, vol. 2248, pp. 239–254. Springer, Heidelberg (2001)
24. Schramm, K., Paar, C.: Higher Order Masking of the AES. In: Pointcheval (ed.) [18], pp. 208–225
25. Shamir, A.: How to Share a Secret. Commun. ACM 22(11), 612–613 (1979)
26. Standaert, F.-X., Veyrat-Charvillon, N., Oswald, E., Gierlichs, B., Medwed, M., Kasper, M., Mangard, S.: The World Is Not Enough: Another Look on Second-Order DPA. In: Abe, M. (ed.) ASIACRYPT 2010. LNCS, vol. 6477, pp. 112–129. Springer, Heidelberg (2010)
27. Waddle, J., Wagner, D.: Towards Efficient Second-Order Power Analysis. In: Joye, M., Quisquater, J.-J. (eds.) CHES 2004. LNCS, vol. 3156, pp. 1–15. Springer, Heidelberg (2004)
28. Wang, Y., Zhu, X.: A fast algorithm for the fourier transform over finite fields and its vlsi implementation. IEEE Journal on Selected Areas in Communications 6(3), 572–577 (1988)

A Mutual Information Study of Goubin and Martinelli's Scheme 1^{st}-Order Flaw

We have seen in Section 3 that Goubin and Martinelli's proposal possesses a first-order flaw whatever the masking order d of their scheme. For the study of this flaw to be complete, we propose here an information theoretic evaluation of the information leakage with respect to the noise standard deviation and d. We moreover compare the quantity of sensitive information in the flaw with that contained in the observation of the $d+1$ shares build thanks to Shamir's sharing for $d=1$ and $d=2$.

To quantify the amount of leaking information, we modelled the relationship between the physical leakage and the value of the variable processed at the time of the leakage. For such a purpose, we associated each $(d+1)$-tuple of shares $(A_0,\cdots,A_d)$ with a $(d+1)$-tuple of leakages $\mathbf{L}=(L_0,\cdots,L_d)$ s.t. $L_j = \mathrm{HW}(A_j)+\mathcal{N}_j$, with $\mathcal{N}_j$ an independent Gaussian noise with mean 0 and standard deviation σ. We use the notation $\mathbf{L} \hookleftarrow (A_0,\cdots,A_d)$ to refer to this association. In the case of the first-order flaw exhibited in Section 3, the leakage $\mathbf{L}$ is univariate and satisfies $\mathbf{L} \hookleftarrow \mathrm{HW}(C_i)+\mathcal{N}$ with C_i being defined as in (3). In that case, the sensitive information in the product $C=(AB)$.

To evaluate the information revealed by each tuple of shares for the polynomial masking technique, we computed the mutual information[4] $I(A,\mathbf{L})$ between

[4] As shown in [26], the number of measurements required to achieve a given success-rate in a maximum likelihood attack can be expressed as a function of the mutual information evaluation and equals $c \times I(A,\mathbf{L})^{-1}$, where c is a constant related to the chosen success-rate.

the sensitive variable A and $\mathbf{L}$. Similarly, in the case of Goubin and Martinelli's scheme, we computed the mutual information $I(AB, \mathbf{L})$ between the sensitive variable (AB) and $\mathbf{L}$. We list hereafter the leakages we considered and the underlying leaking variables:

$$\text{(2,1)-sharing leakage:} \quad \mathbf{L} \hookleftarrow (P_A(\alpha_1), P_A(\alpha_2)) \ . \tag{9}$$

$$\text{(3,2)-sharing leakage:} \quad \mathbf{L} \hookleftarrow (P_A(\alpha_1), P_A(\alpha_2), P_A(\alpha_3)) \ . \tag{10}$$

$$\text{Flaw in Alg. 2 for } d = 1\text{:} \quad \mathbf{L} \hookleftarrow (C_i = (P_A \cdot P_B)_{|1}(\alpha_i)) \ . \tag{11}$$

$$\text{Flaw in Alg. 2 for } d = 2\text{:} \quad \mathbf{L} \hookleftarrow (C_i = (P_A \cdot P_B)_{|2}(\alpha_i)) \ . \tag{12}$$

Figure A summarizes the information theoretic evaluation for each leakage (9) to (12). For d equal to 1 or 2, it can be observed that the amount of information revealed by the $d+1$ sharing elements is greater than that revealed by the 1^{st}-order flaw up to a certain amount of noise. As a matter of fact, the first-order flaw is less impacted by the noise than the 2^{nd}-order and 3^{rd}-order leakages. Hence, for any choice of input parameter d in Algorithm 2 and for any Shamir's sharing order $d' > d$, there exists a noise standard deviation σ s.t. the first-order flaw leaks more sensitive information than the d'-tuple of Shamir's shares. For example, for $d = 1$, then the first-order flaw in Algorithm 2 leaks more information than any d'^{th}-order sharing with $d' > 1$ as long as $\sigma > 3.7$. We also emphasize that the traces' resynchronization issue and the computational complexity of the processings make higher-order SCA attacks much more difficult to mount in practice than first-order ones. As a consequence, the first-order flaw is even more important from a practical point of view than suggested in Fig. A.

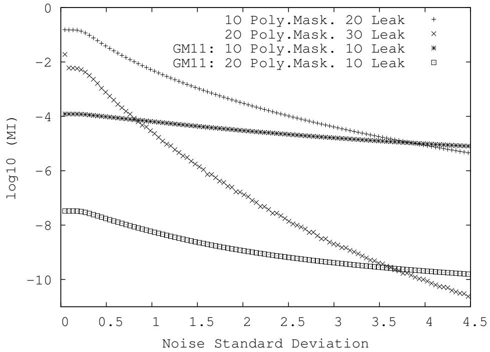


Fig. 2. Mutual information ($\log_{10}$) between the leakage and the sensitive variable over an increasing noise standard deviation (x-axis)

Secure Multiple SBoxes Implementation with Arithmetically Masked Input

Luk Bettale

Oberthur Technologies
71-73 rue des Hautes Pâtures
92726 Nanterre Cedex - France
l.bettale@oberthur.com

Abstract. The building blocks of several block ciphers involve arithmetic operations, bitwise operations and non-linear functions given as SBoxes. In the context of implementations secure against Side Channel Analysis, these operations shall not leak information on secret data. To this end, masking is a widely used protection technique. Propagating the masks through non-linear functions is a necessary task to achieve a sound and secure masked implementation. This paper describes an efficient method to securely access N SBoxes when the N inputs are encoded as a single word arithmetically masked. This problematic arises for instance in a secure implementation of the standard block ciphers GOST or SEED. A method using state of the art algorithms would be to first perform an arithmetic to boolean mask conversion before independently accessing the N SBoxes. Compared to this method, the algorithm proposed in this paper needs less code, less random generation and no extra memory. This makes our algorithm particularly suitable for very constrained devices. As a proof of concept, we compare an implementation in 8051 assembly language of our algorithm to the existing solutions.

Keywords: Side Channel Analysis, Differential Power Analysis, Block Cipher, SBox, Arithmetic Masking, Boolean Masking, Mask Conversion.

1 Introduction

During the execution of a cryptographic algorithm, power consumption, execution timings, or electro-magnetic radiation may give information on secret data. The techniques using these leakages to attack cryptographic primitives are called Side Channel Analysis (SCA). Different techniques appeared in the literature. Among them, Differential Power Analysis (DPA) [15,5,17] turns out to be a powerful tool to attack implementations [3,1,20]. To recover a small part k of a secret, DPA consists in recording the power consumption – the leakage – of d moments where a sensitive variable (i.e. a value depending on k and the input) is involved, for a large number of different inputs. Then, an attacker makes predictions on a combination of these leakages for all possible hypotheses on k according to a well-chosen leakage model. Finally, the attacker outputs the hypothesis $\widehat{k}$ for

S. Mangard (Ed.): CARDIS 2012, LNCS 7771, pp. 91–105, 2013.

which the prediction and the actual leakages are the most "similar". This degree of similarity is usually measured using statistical tools such as the Pearson correlation coefficient [5]. The number d of moments denotes the order of the DPA.

In parallel, countermeasures were soon developed to secure implementations against DPA. In this specific context, a countermeasure aims at preventing all sensitive data from leaking information, or more precisely making their leakages independent of the secret. A commonly used countermeasure is to *mask* – or split – a sensitive variable with one (resp. d) random value(s) [11,6]. This is called first order masking (resp. d-th order masking). With this countermeasure, as the random masks change at each execution, the leakage of the masked variable is statistically independent of the secret. Only the combined leakages of the masked variable and all the masks could reveal some information. Thus, a $d+1$-th order DPA is necessary to attack a d-th order masked implementation.

Sensitive data shall then remain masked through every step of a cryptographic algorithm. The critical task is to adapt the algorithm to keep this state, especially for non-linear parts. In block ciphers, the use of highly non-linear functions (w.r.t. bitwise addition) is mandatory to prevent classical cryptanalysis. One way to represent such functions is to use a so-called *SBox*. SBoxes have been used to design many symmetric encryption schemes such as the DES or the AES. Several masking schemes have been developed for these functions to prevent DPA of the first order [11,2,4,9] or higher order [25,24,23]. In general, for block ciphers built upon SBoxes and linear layers, a masking scheme based on *boolean masking* is chosen and an adapted secure SBox access [18,22] is implemented.

Some algorithms also use modular addition as a non-linear function. The block cipher IDEA [16] is an example. For this kind of algorithm, boolean masking is not always suitable. Indeed, a boolean mask propagates easily through linear functions, but with modular addition, *arithmetic masking* is preferred. To switch from one masking to another, a mask conversion is required. Several methods have been proposed [10,7,19,8], and an application to IDEA is suggested for instance in [19].

Finally, both modular addition and SBoxes may be used together as in the Korean standard block cipher SEED [26], or in the Russian standard block cipher GOST 28147-89 [28]. In these algorithms, multiple small SBoxes are used simultaneously to compute the image of a larger input. More precisely, arithmetic operations are performed on the large input before accessing these SBoxes. It is a natural requirement to protect such algorithms against DPA. This paper focuses on protecting this kind of block ciphers against first order DPA. More specifically, given N SBoxes (ℓ bits $\rightarrow \ell'$ bits), we need to securely access these SBoxes when the N inputs are encoded as a single word of $n = N\,\ell$ bits, arithmetically masked – addition modulo 2^n – with a random n-bit mask. Though, protecting an implementation usually comes with a cost, either in memory or in performances. Often, devices that are more likely to suffer Side Channel Analysis are also very constrained in terms of memory (smart-cards, embedded devices, ...). A method has been proposed in [13] and applied to the SEED algorithm. The method is highly efficient but uses a non-negligible amount of RAM. It is then interesting to have an alternative solution requiring lower memory resources.

Contributions. In this paper, we study the protection of block ciphers mixing arithmetic operations, boolean operations and multiple SBoxes against first order DPA. More specifically, we focus on the secure access to multiple SBoxes when arithmetic masking is used. We propose a novel algorithm to perform this operation. Compared to existing methods, our algorithm needs less memory, less code and less random generation.

Organization of the Paper. This paper is organized as follows. In Sect. 2, we give some notations and review existing algorithms to securely access one SBox. We also give some useful background on masking conversion. In Sect. 3, we present in detail the main concern of this paper, namely the problem of accessing simultaneously N SBoxes with an arithmetically masked input. In this section we also propose a first solution using state of the art techniques. In Sect. 4, we present our new algorithm as well as a careful security analysis. We compare an implementation of both solutions in Sect. 5. Section 6 concludes this article.

2 Background and Related Work

This paper deals with two kinds of masking:

- Boolean masking: a sensitive variable $a \in \mathbb{F}_2^n$ is masked with a random value $m \in \mathbb{F}_2^n$ by computing $\widetilde{a} = a \oplus m$, where $\oplus$ denotes bitwise exclusive or.
- Arithmetic masking: a sensitive variable $a \in \mathbb{F}_2^n$ is masked with a random value $m \in \mathbb{F}_2^n$ by computing $\widetilde{a} = a \boxplus m$, where $\boxplus$ denotes addition modulo 2^n.

When the masking operation is not precised, we may use operators $\star$ and $\diamond$ to denote either $\oplus$ or $\boxplus$. The unmasking operation for $\star$ is denoted by $\star^{-1}$. For instance if $\star$ is $\boxplus$, then $\star^{-1}$ is $\boxminus$. If $\star$ is $\oplus$, then $\star^{-1}$ is also $\oplus$.

Sometimes, a value $a \in \mathbb{F}_2^n$ is viewed as N chunks of ℓ bits where $n = N\,\ell$. The notation a_i, $0 \leqslant i < N$, is used to denote the i-th chunk where a_0 is the least significant chunk. Then, we use the notation $a = a_{N-1}\|\ldots\|a_0$. If a modular operation is performed on a small chunk, the operator is sub-scripted by the corresponding bit-size. For instance, we write $\boxplus_\ell$ to denote the addition modulo 2^ℓ. If the operation is done on n bits, the subscript is omitted to ease reading.

2.1 Substitution Boxes

A non-linear function $f : \mathbb{F}_2^n \to \mathbb{F}_2^p$ is often described as an SBox. An SBox is a table which gives for an n-bit index a the corresponding p-bit output $f(a)$. If S is an SBox, we use the notation $S[a]$ to denote the image of a by the corresponding function f.

In a secure implementation context, it is necessary that a masked input $\widetilde{a}$ remains masked during the SBox access. Moreover, the output of the SBox shall also be masked, possibly with a different mask. The input masking is denoted by the operation $\star$ and the output masking is denoted by the operation $\diamond$.

In [18], Messerges proposes to recompute a table corresponding to a masked version of the SBox, that is, to compute an SBox T such that $T[a \star m] = S[a] \diamond m'$. This method has been chosen in [2] to secure the DES and AES SBoxes. The method is efficient but requires a large amount of memory ($p \cdot 2^n$ bits). In a context where memory is a limited resource, this method could be a bottleneck if n is too large. In [22], the authors proposed an alternative method, more suitable to such environment. The method does not require extra memory space. We describe it in Alg. 1.

Algorithm 1. Secure SBox implementation [22]

```
Inputs: $S : \mathbb{F}_2^n \to \mathbb{F}_2^p,\ \widetilde{a} = (a \star m) \in \mathbb{F}_2^n,\ m \in \mathbb{F}_2^n,\ m' \in \mathbb{F}_2^p,$
    $\star : (\mathbb{F}_2^n \times \mathbb{F}_2^n) \to \mathbb{F}_2^n,\ \diamond : (\mathbb{F}_2^p \times \mathbb{F}_2^p) \to \mathbb{F}_2^p$
Output: $\widetilde{b} = S[a] \diamond m'$
function SecureSBox($S, \widetilde{a}, m, m', \star, \diamond$)
    for $k = 0$ to $2^n - 1$ do
        cmp ← $(k \overset{?}{=} m)$                ▷ if $k$ and $m$ are equal, cmp is 1, else 0
        $t \leftarrow \widetilde{a} \star^{-1} k$                    ▷ unmask $\widetilde{a}$ with the loop index
        $R_{\mathrm{cmp}} \leftarrow S[t] \diamond m'$
    end for
    return $R_1$
end function
```

Using Alg. 1, the real unmasking operation is dissimulated among $2^n - 1$ other dummy unmasking. Note that the masking type may be the same for both the input and the output. In the rest of the paper, this method is preferred to the table re-computation as we focus on implementation on memory-constrained devices.

Remark 1. Contrary to this paper, the method proposed in [13] is built upon the table re-computation algorithm of [18].

2.2 Masking Conversion

A sensitive variable has to remain masked throughout the cryptographic algorithm. A boolean (resp. arithmetic) masking propagates easily through boolean (resp. arithmetic) operations. When a specific algorithm mixes boolean operations (rotations, bitwise and/or, ...) and arithmetic operations (modular addition/subtraction), *mask conversion* is needed. An efficient method to perform boolean to arithmetic masking conversion (BMtoAM) has been proposed by Goubin in [10]. It uses a constant number of operations with respect to the bitsize of the data. The converse operation (AMtoBM) is more costly to achieve.

A first method has also been proposed in [10] but it requires a number of operations linear in the bit-size. Later, Coron and Tchulkine introduced in [7] a more efficient method which pre-computes a conversion table for "small" input and output masks r and s (say λ bits over n), and processes the input by λ-bit chunks. It appears that the method fails for some inputs because of a problem in the carry propagation between the chunks [8]. In 2004, Neiße and Pulkus proposed a sound and efficient method to perform AMtoBM conversion in [19]. Their method is similar to [7], as it also uses a conversion table for a smaller λ-bit mask, but they handle the carries in a different way. A pre-computation step first generates two uniformly distributed λ-bit masks r and s. Then, two tables T and C are initialized. For every possible λ-bit inputs k ($0 \leqslant k < 2^\lambda$) the value $(k \boxminus_\lambda r) \oplus s$ is stored in $T[k]$. The carry resulting from $(k \boxminus_\lambda r)$ is stored in $C[k]$. The whole process is described in Alg. 2.

Algorithm 2. Secure AMtoBM implementation [19]: pre-computation

```
1: procedure AMtoBMPrecomp( )
2:     $r \leftarrow$ Random($\{0, \ldots, 2^\lambda\}$)
3:     $s \leftarrow$ Random($\{0, \ldots, 2^\lambda\}$)
4:     for $k = 0$ to $2^\lambda - 1$ do
5:         $T[k] \leftarrow (k \boxminus_\lambda r) \oplus s$          ▷ subtraction may generate a carry
6:         $C[k] \leftarrow$ carry(step 5)
7:     end for
8: end procedure
```

To convert the masking without leaking information, the authors use the following useful property. For any $u, v \in \mathbb{F}_2^\lambda$, it holds that

$$\neg(u \boxplus_\lambda v) = \neg u \boxplus_\lambda \neg v \boxplus_\lambda 1 \ ,$$

where $\neg x$ denotes the bitwise complement of x. Let (-1) be the value $2^\lambda - 1$ (all λ bits are set to 1). Then, for any $z \in \{0, 1\}$, it holds that

$$(u \boxplus_\lambda v) \oplus (-z) = (u \oplus (-z)) \boxplus_\lambda (v \oplus (-z)) \boxplus_\lambda z \ . \tag{1}$$

If the bit z is chosen uniformly at random for each execution, it can be used to mask the propagating carries. Indeed, whenever $u \boxplus_\lambda v$ generates a carry, $\neg u \boxplus_\lambda \neg v \boxplus_\lambda 1$ does not, and conversely. The pre-computed tables T and C can then be used to convert and propagate the carry by λ-bit chunks. This is the purpose of the conversion algorithm from [19] described in Alg. 3.

Remark 2. The authors of [19] also propose several optimizations to their algorithm such as storing $C[k]$ in place of the least significant bit (LSB) of $T[k]$ after remarking that if the LSB of r and s are the same, the LSB of k and $T[k]$ are also the same (for $0 \leqslant k < 2^\lambda$). Then this bit does not need to be stored when s is chosen accordingly, for instance when $s = r$.

Algorithm 3. Secure AMtoBM implementation [19]: conversion

```
Inputs: $\widetilde{a} = (a \boxplus m) \in \mathbb{F}_2^n$, $m \in \mathbb{F}_2^n$, $m' \in \mathbb{F}_2^n$
Output: $\widetilde{b} = a \oplus m'$
function AMtoBM($\widetilde{a}, m, m'$)
    AMtoBMPrecomp( )                              ▷ only if not already done
    $z \leftarrow$ Random($\{0, 1\}$)
    $f \leftarrow \widetilde{a} \oplus (-z)$                      ▷ complement (or not) the input
    $g \leftarrow (m \oplus (-z)) \boxminus (r\|\ldots\|r)$            ▷ change mask to $r\|\ldots\|r$
    $h \leftarrow ((s\|\ldots\|s) \oplus (-z)) \oplus m'$          ▷ change mask from $s\|\ldots\|s$
    $c \leftarrow z$
    for $i = 0$ to $N - 1$ do
        $f \leftarrow f \boxminus g_i \boxminus c$
        $\widetilde{b}_i \leftarrow T[f_0] \oplus h_i$
        $c \leftarrow C[f_0]$
        $f \leftarrow f \gg \lambda$
    end for
    return $\widetilde{b} = \widetilde{b}_{N-1}\|\ldots\|\widetilde{b}_0$
end function
```

Recently, Debraize proposed an alternative method in [8]. The method is similar to [19], but the carry is protected by computing two sets of tables $T^{(0)}, C^{(0)}$ and $T^{(1)}, C^{(1)}$. These tables can be described using table T and C of Alg. 2 by:

$$\begin{aligned} T^{(\rho)}[i] &= T[i] \ , & T^{(\rho\oplus 1)}[i] &= T[i \boxminus_\lambda 1] \ , \\ C^{(\rho)}[i] &= C[i] \oplus \rho \ , & C^{(\rho\oplus 1)}[i] &= C[i \boxminus_\lambda 1] \oplus \rho \ , \end{aligned}$$

where ρ is a bit randomly chosen at each execution. Whether to access $T^{(0)}$ or $T^{(1)}$ is decided according to the value of the input carry masked by ρ. The output carry is masked again by ρ. This way, the input and output carries are always blinded. Compared to Alg. 3, the resulting algorithm would require twice the amount of memory. To suit our low-memory requirements, in the rest of this paper Alg. 3 is preferred.

3 Multiple SBoxes

As described in Sect. 1, in some algorithms, several "small" SBoxes are used simultaneously to compute a non-linear function on a "big" input. Let $\ell, N \in \mathbb{N}$ and let $n = N\ell$. Let $a \in \mathbb{F}_2^n$ be an SBox input. It can be written as $a = a_{N-1}\|\ldots\|a_0$, where a_i are elements of $\mathbb{F}_2^\ell$, for $0 \leqslant i < N$. Consider N SBoxes $S_0, \ldots, S_{N-1}$ which map ℓ bits to ℓ' bits ($\mathbb{F}_2^\ell \rightarrow \mathbb{F}_2^{\ell'}$). We define S to be the "SBox" mapping n bits to $N\ell'$ bits such that

$$S[a] = S_{N-1}[a_{N-1}]\|\ldots\|S_0[a_0] \ .$$

For instance, the DES has $N = 8$ SBoxes with $\ell = 6$ and $\ell' = 4$. For the sake of clarity, in what follows, we assume that $\ell = \ell'$. Nevertheless, the results below can be extended to the case $\ell \neq \ell'$.

Boolean Masking. When the input a of multiple SBoxes is masked with boolean masking, the application of Alg. 1 to each small SBox is straightforward because each part m_i of a boolean mask m is a boolean mask for the corresponding part a_i of an input a:

$$\widetilde{a} = a \oplus m = a_{N-1} \oplus m_{N-1} \| \ldots \| a_0 \oplus m_0 \ .$$

One may then securely compute $S[a]$ using table re-computation or using Alg. 1 by computing

$$\begin{aligned} &\textsc{SecureSBox}\left(S_{N-1}, \widetilde{a}_{N-1}, m_{N-1}, m'_{N-1}, \oplus, \oplus\right) \| \ldots \\ &\qquad \ldots \| \textsc{SecureSBox}\left(S_0, \widetilde{a}_0, m_0, m'_0, \oplus, \oplus\right) \ , \end{aligned} \tag{2}$$

where the m'_i's are ℓ-bit parts of an output mask m'. This method is the one used in [2] to secure the DES SBoxes with table re-computation. However, an issue appears when the input is arithmetically masked.

3.1 Multiple SBoxes on Arithmetically Masked Input

In some algorithms, it can be necessary that the input of multiple SBoxes is masked with an arithmetic mask. In Fig. 1, we give as an example the round function of a typical block cipher. At round t, the current sub-key $\text{key}^{(t)}$ is added to the state $v^{(t)}$ to constitute the N ℓ-bit input a. The output b of the $N = 4$ SBoxes is then rotated to produce the next state $v^{(t+1)}$.

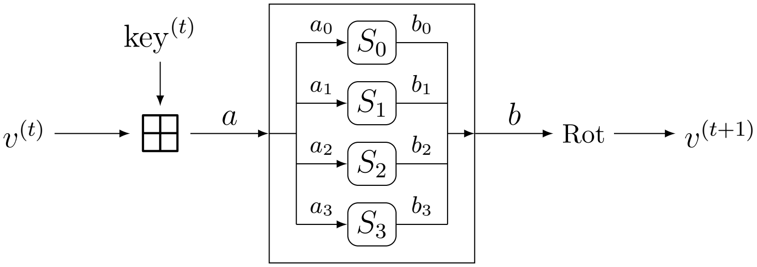


Fig. 1. A block cipher round using an arithmetic operation and multiple SBoxes

This kind of round function is used for instance in SEED where $N = 4$ and $\ell = 8$, or in GOST 28147-89 where $N = 8$ and $\ell = 4$. In both cases, the input a and the output b are 32-bit words.

To protect such a round against DPA, $v^{(t)}$, $v^{(t+1)}$ as well as the intermediate values a and b should be masked. In this example, $v^{(t)}$ should be preferably masked with modular addition to allow the mask to propagate through the operation $v^{(t)} \boxplus \text{key}^{(t)}$. As for b, it should be masked with a boolean mask which easily propagates through the rotation. In these conditions, a simple solution as in the boolean case cannot be achieved.

When the input a is arithmetically masked with a mask m, the following property holds:

$$\widetilde{a} = a \boxplus m = (a_{N-1} \boxplus_\ell m_{N-1} \boxplus_\ell c_{N-1}) \,\|\, \ldots \,\|\, (a_0 \boxplus_\ell m_0 \boxplus_\ell c_0) \quad , \tag{3}$$

where $c_0 = 0$ and for all i, $1 \leqslant i < N$,

$$c_i = \begin{cases} 1 & \text{if} \quad (a_{i-1} + m_{i-1} + c_{i-1}) \geqslant 2^\ell \\ 0 & \text{otherwise} \end{cases} \quad .$$

The value c_i corresponds to the carry propagating through the modular addition $\boxplus$. Because of this carry, the direct application of (2) is not possible. We have then to address the following problem.

Problem 1. Given an arithmetically masked n-bit input $\widetilde{a} = a \boxplus m$, and N ℓ-bit SBoxes $S = (S_0, \ldots, S_{N-1})$ such that $n = N\,\ell$ as defined above, we want to securely compute $\widetilde{b} = b \diamond m'$ such that $b = S[a]$, where m and m' are uniformly distributed n-bit masks, and $\diamond$ is either $\boxplus$ or $\oplus$.

Assume that an algorithm called SecureMulSBox solves Prob. 1 with $\diamond = \oplus$, a secure implementation of the round function of Fig. 1 would be achieved in Fig. 2. Our goal is then to find an implementation of SecureMulSBox.

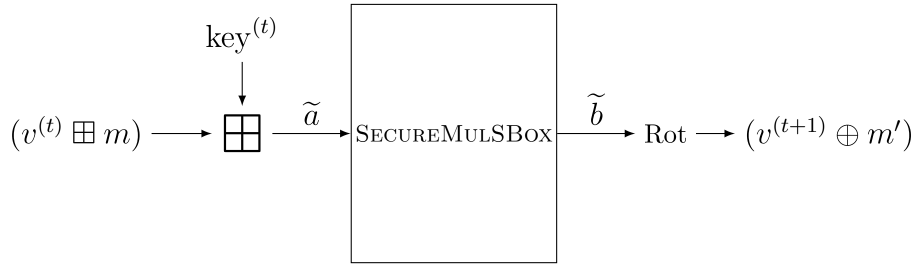


Fig. 2. A masked block cipher round using an arithmetic operation and a secure implementation of multiple SBoxes

3.2 A Solution Using Mask Conversion

We have seen that accessing N SBoxes when boolean masking is used can be done independently. Then, an answer to Prob. 1 could be to first perform an AMtoBM conversion and then apply N times Alg. 1 on each small SBox. This process is described in Alg. 4.

The output of Alg. 4 is masked with boolean masking. To obtain an arithmetic masking instead, a BMtoAM conversion could be added at the end. As mentioned in Sect. 2.2, the conversion in this direction is not very costly. In particular, it does not require extra memory. Thus, in the rest of this paper, we focus on solving Prob. 1 with $\diamond = \oplus$.

Algorithm 4. Secure multiple SBox with AMtoBM implementation

```
Inputs: $S = (S_0, \dots, S_{N-1}) \in (\mathbb{F}_2^\ell \to \mathbb{F}_2^\ell)^N$, $\widetilde{a} = (a \boxplus m) \in \mathbb{F}_2^n$, $m \in \mathbb{F}_2^n$, $m' \in \mathbb{F}_2^n$
Output: $\widetilde{b} = S[a] \oplus m' = (S_{N-1}[a_{N-1}] \oplus m'_{N-1}) \| \dots \| (S_0[a_0] \oplus m'_0)$
function SecureMulSBox($S, \widetilde{a}, m, m'$)
    $\widetilde{t} \leftarrow$ AMtoBM($\widetilde{a}, m, m$)                ▷ keep $m$ as output mask
    for $i = 0$ to $N - 1$ do
        $\widetilde{b}_i \leftarrow$ SecureSBox($S_i, \widetilde{t}_i, m_i, m'_i, \oplus, \oplus$)
    end for
    return $\widetilde{b} = \widetilde{b}_{N-1} \| \dots \| \widetilde{b}_0$
end function
```

Algorithm 4 needs to pre-compute 2^λ elements in a table (to use AMtoBM from [19]). A large value of λ ensures a faster conversion time, but to fit our low memory requirements, λ would have to be quite small. This implies a more expensive conversion. In the next section, we discuss a new solution which integrates the conversion within the secure SBoxes computation.

4 Secure Multiple SBoxes with Arithmetic Masking

4.1 Algorithm

As pointed out in Sect. 2.2, the propagation of the carry coming from the arithmetic masking is an issue. In the solution of Prob. 1 presented in Alg. 4, it is carried out by the masking conversion algorithm. If one could turn an arithmetic mask m on $n = N\,\ell$ bits into N arithmetic masks on ℓ bits for a given masked input, then, similarly to (2), one could compute

$$\begin{aligned}\text{SecureSBox}\left(S_{N-1}, \widetilde{a}_{N-1}, m_{N-1}, m'_{N-1}, \boxplus_\ell, \oplus\right) \| \dots \\ \dots \| \text{SecureSBox}\left(S_0, \widetilde{a}_0, m_0, m'_0, \boxplus_\ell, \oplus\right) \ . \quad (4)\end{aligned}$$

We describe in Alg. 5 a naive, though non-secure algorithm to "remove carries" from an arithmetic mask.

Algorithm 5 is insecure against DPA as it manipulates the sensitive variable a (more precisely, parts a_i of the sensitive variable) unmasked. This is done at step 4 when the i-th input block $\widetilde{a}_i$ is unmasked to compute the carry for the $i\,\ell$-th bit. However if we use (4), in the algorithm SecureSBox presented in Alg. 1 a loop for every possible masks is performed to securely access the i-th SBox. The idea is to use this loop to dissimulate not only the unmasking operation, but also the next carry computation.

Still, each output carry has to be masked. This may be performed using a bit z as in the mask conversion algorithm from [19] described in Alg. 3. Indeed, consider (3) when masked by a random bit z. It holds then that

$$\begin{aligned}\widetilde{a} \oplus (-z) &= (a \boxplus m) \oplus (-z) \\ &= (a_{N-1} \boxplus_\ell m_{N-1} \boxplus_\ell c_{N-1}) \oplus (-z) \| \dots \| (a_0 \boxplus_\ell m_0 \boxplus_\ell c_0) \oplus (-z) \ .\end{aligned}$$

Algorithm 5. Non-secure removing of carries

```
Inputs: $\widetilde{a} = (a \boxplus m) \in \mathbb{F}_2^n,\ m \in \mathbb{F}_2^n$
Output: $\widetilde{b} = (a_{N-1} \boxplus_\ell m_{N-1}) \| \dots \| (a_0 \boxplus_\ell m_0)$
function RemoveCarries($\widetilde{a}, m$)
    $c = 0$                                        ▷ $C$ propagates the carries
    for $i = 0$ to $N-1$ do
        $t \leftarrow \widetilde{a}_i \boxminus_\ell m_i \boxminus_\ell c$
        $c \leftarrow$ carry(step 4)                    ▷ save next carry
        $b_i \leftarrow t \boxplus_\ell m_i$
    end for
    return $\widetilde{b} = \widetilde{b}_{N-1} \| \dots \| \widetilde{b}_0$
end function
```

For each i, $0 \leqslant i < N$, let $a_i^z = a_i \oplus (-z)$ and $m_i^z = m_i \oplus (-z)$. Let $c_i^z = c_i \oplus (-z) = c_i \oplus z$, the i-th carry masked by the bit z. Then, according to (1), we obtain that

$$\widetilde{a} \oplus (-z) = \left(a_{N-1}^z \boxplus_\ell m_{N-1}^z \boxplus_\ell c_{N-1}^z\right) \| \dots \| \left(a_0^z \boxplus_\ell m_0^z \boxplus_\ell c_0^z\right) \ . \tag{5}$$

Equation (5) together with Alg. 5 are used to derive the main algorithm of this paper presented in Alg. 6.

To prove the correctness of Alg. 6, we study the internal variables when $i = 0$. It can be noticed that only when k is equal to the mask chunk m_0^z, step 12 becomes

$$R_1 = S_0[(\widetilde{a}_0^z \boxminus_\ell m_0^z \boxminus_\ell z) \oplus (-z)] \oplus m_0' \ .$$

Using (5), as $\widetilde{a}_0^z = a_0^z \boxplus_\ell m_0^z \boxplus_\ell z$, it holds that

$$\begin{aligned} R_1 &= S_0[((a_0^z \boxplus_\ell m_0^z \boxplus_\ell z) \boxminus_\ell m_0^z \boxminus_\ell z) \oplus (-z)] \oplus m_0' \\ &= S_0[a_0^z \oplus (-z)] \oplus m_0' \\ &= S_0[a_0] \oplus m_0' \ . \end{aligned}$$

At step 11, we also have

$$\begin{aligned} B_1 &= \text{carry(step 10)} \\ &= \begin{cases} 1 & \text{if} \quad (a_0^z + m_0^z + z) \geqslant 2^\ell \\ 0 & \text{otherwise} \end{cases} \ . \end{aligned}$$

This corresponds to the next carry masked by the bit z:

$$B_1 = c_1 \oplus z \ .$$

For $i \geqslant 0$, assume that B_1 contains the current masked carry at the beginning of step 8. For each i, $0 < i < N$, it holds then that

$$\begin{aligned} R_1 &= S_i[(\widetilde{a}_i^z \boxminus_\ell m_i^z \boxminus_\ell c_i^z) \oplus (-z)] \oplus m_i' \\ &= S_i[((a_i^z \boxplus_\ell m_i^z \boxplus_\ell c_i^z) \boxminus_\ell m_i^z \boxminus_\ell c_i^z) \oplus (-z)] \oplus m_i' \\ &= S_i[a_i^z \oplus (-z)] \oplus m_i' \\ &= S_i[a_i] \oplus m_i' \ . \end{aligned}$$

Algorithm 6. Secure multiple SBox with arithmetically masked input

Inputs: $S = (S_0, \dots, S_{N-1}) \in (\mathbb{F}_2^\ell \to \mathbb{F}_2^\ell)^N$, $\widetilde{a} = (a \boxplus m) \in \mathbb{F}_2^n$, $m \in \mathbb{F}_2^n$, $m' \in \mathbb{F}_2^n$

Output: $\widetilde{b} = S[a] \oplus m' = (S_{N-1}[a_{N-1}] \oplus m'_{N-1}) \| \dots \| (S_0[a_0] \oplus m'_0)$

```
 1: function SecureMulSBox($S, \widetilde{a}, m, m'$)
 2:     $z \leftarrow$ Random($\{0, 1\}$)
 3:     $\widetilde{a}^z \leftarrow \widetilde{a} \oplus (-z)$
 4:     $m^z \leftarrow m \oplus (-z)$
 5:     $B_1 \leftarrow z$                                  ▷ $B_1$ propagates the correct carries
 6:     for $i = 0$ to $N - 1$ do
 7:         $c \leftarrow B_1$
 8:         for $k = 0$ to $2^\ell - 1$ do
 9:             cmp $\leftarrow (k \overset{?}{=} m_i^z)$        ▷ if $k$ and $m$ are equal, cmp is 1, else 0
10:             $t \leftarrow \widetilde{a}_i^z \boxminus_\ell k \boxminus_\ell c$
11:             $B_{\text{cmp}} \leftarrow$ carry(step10)                  ▷ save next carry
12:             $R_{\text{cmp}} \leftarrow S_i[t \oplus (-z)] \oplus m'_i$
13:         end for
14:         $\widetilde{b}_i \leftarrow R_1$
15:     end for
16:     return $\widetilde{b} = \widetilde{b}_{N-1} \| \dots \| \widetilde{b}_0$
17: end function
```

Consequently, we also obtain

$$B_1 = c_{i+1}^z = c_{i+1} \oplus z \ \ ,$$

which is the next masked carry. At the end of Alg. 6, $\widetilde{b}_i = S_i[a_i] \oplus m'_i$ for each i, $0 \leqslant i < N$, which is the expected output.

4.2 Security Analysis

To achieve security against first order DPA, each step of our algorithm shall not depend on any unmasked secret data. Before discussing the security in details, we need to precisely define the attacker model.

Attacker: We assume that the attacker has full access to a device performing an encryption using a block-cipher implementing Alg. 6 as a countermeasure in the same context as Fig. 2. The attacker can choose any input to the block-cipher, and we assume that she is able to predict the value of the input a of Alg. 6 for any hypothesis on the secret key$^{(t)}$. The attacker can only perform first-order DPA, meaning that only one moment of the execution can be targeted at each execution. But, the attacker is able to acquire as many power traces as needed for the targeted moment.

Leakage: We assume that the device is leaking information on the values involved during every operation. This information may be their Hamming weight. Every operation is assumed to leak similarly.

Recall that a and $S[a]$ are sensitive data. Algorithm 6 is secure against the previously defined model only if each operation involved behaves as it was manipulating only random or constant values. No operation should involve both a masked value and the corresponding mask at the same time. We analyze each step of Alg. 6 involving a sensitive data in Table 1. Steps 2, 4, 5 and 9 are omitted as they manipulate only constant or random value. The notation k^z is used to denote $k \oplus (-z)$.

Table 1. Sensitive values manipulated in Alg. 6

Step	Instruction	Manipulated value	Sensitive value	Mask(s)
3.1	$t \leftarrow \widetilde{a}$	$a \boxplus m$	a	m
3.2	$\widetilde{a}^z \leftarrow t \oplus (-z)$	$(a \boxplus m) \oplus (-z)$	a	m, z
7	$c \leftarrow B_1$	c_i^z	c_i	z
10.1	$t \leftarrow \widetilde{a}_i^z$	$(a_i \boxplus_\ell m_i \boxplus_\ell c_i) \oplus (-z)$	$a_i \boxplus_\ell c_i$	m_i, z
10.2	$t \leftarrow t \boxminus_\ell k \boxminus_\ell c$	$a_i \oplus (-z) \boxplus_\ell m_i^z \boxminus_\ell k$	a_i	z, $m_i^z \boxminus_\ell k$
11	$B_{\text{cmp}} \leftarrow$ CARRY(step10)	c_{i+1}^z	c_{i+1}	z
12.1	$t \leftarrow t \oplus (-z)$	$a_i \boxplus_\ell m_i \boxminus_\ell k^z$	a_i	$m_i \boxminus_\ell k^z$
12.2	$t \leftarrow S_i[t]$	$S_i[a_i \boxplus_\ell m_i \boxminus_\ell k^z]$	a_i	$m_i \boxminus_\ell k^z$
12.3	$R_{\text{cmp}} \leftarrow t \oplus m_i'$	$S_i[a_i \boxplus_\ell m_i \boxminus_\ell k^z] \oplus m_i'$	a_i	$m_i \boxminus_\ell k^z$, m_i'
14	$\widetilde{b}_i \leftarrow R_1$	$S_i[a_i] \oplus m_i'$	$S_i[a_i]$	m_i'

We further detail the steps presented in Table 1:

- Step 3 manipulates a masked variable, and the operation only adds another independent random bit.
- Steps 7 and 14 are assignments of masked data: B_1 is the next carry masked by z and R_1 is $S_i[a_i]$ masked by m_i'.
- At steps 10 and 12, k is constant, then the masks $m_i^z \boxminus_\ell k$ or $m_i \boxminus_\ell k^z$ are uniformly distributed on ℓ bits. These steps are performed once for every possible value of k. Next, the carry read at Step 11 is already masked by the bit z and gives no information on the sensitive variable.

According to Table 1, each sensitive value is masked with a uniformly distributed mask of at least as many bits as the sensitive value.

5 Implementation

We implemented our method (Alg. 6) as well as the method using the mask conversion from [19], then the secure SBox algorithm of [22] (Alg. 4). The implementations were done in 8051 assembly with the same optimizations – favoring small code size. The implemented example is $N = 8$ SBoxes of $\ell = 4$ bits input and output which are the parameters used in the GOST block-cipher. For Alg. 4, we used $\lambda \in \{2, 4, 8\}$ as chunk size for the AMtoBM conversion algorithm. In the cases $\lambda < 8$, each element of precomputed tables T and C has been stored on one byte. In the case $\lambda = 8$, we also used the optimization for Alg. 4 described in Remark 2. We give in Table 5 a comparison between these methods regarding several parameters. Our method has not been compared to methods based on table re-computation because we targeted low memory devices. For these parameters, such methods would require at least 64 extra bytes of memory.

Table 2. Comparison of 8051 implementations of Alg. 4 and Alg. 6

	Alg. 4 ([19] + [22])			Alg. 6
	$\lambda = 2$	$\lambda = 4$	$\lambda = 8$	(this paper)
rand. gen. (in bits)	3	5	9	1
pre-comp. time (in cy)	72	201	3349	$\emptyset$
algorithm time (in cy)	3013	2773	2633	3334
XRAM (in bytes)	4	16	256	0
Code (in bytes)	276	271	256	151

It can be noticed that our new algorithm outperforms the others in terms of memory requirements (RAM and code). This makes it particularly suitable on devices with limited resources. If only one call is needed, the execution time of our algorithm is similar to the algorithms from [19] and [22] with $\lambda = 2$, a bit slower with $\lambda = 4$, and better with $\lambda = 8$. Then, for the implementation of a full block cipher, depending on the number of secure rounds needed, our method may be less efficient than those using pre-computation, but still has the advantage of using no extra memory. Furthermore, the random generation time has not been taken into account in the provided timings. On a device where no fast random is available, the cost of the additional random generation needed by Alg. 4 may make it even slower.

6 Conclusion

We have introduced a new method that answers the problem of accessing so-called multiple SBoxes with arithmetically masked input in the context of a software implementation secure against first order DPA. One advantage of the proposed solution is that no extra RAM is needed to securely compute the output of the SBoxes. Besides, the code size of the proposed method is relatively low, and the execution speed remains competitive with other methods when only few computations are needed. All in all, our method is particularly suitable for extremely constrained devices with tight requirements on memory (RAM and ROM). We have demonstrated the security of our method against first-order DPA. An extension of our algorithm to second-order masking as in [24] is the next step of this work.

References

1. Akkar, M.-L., Bévan, R., Goubin, L.: Two Power Analysis Attacks against One-Mask Methods. In: Roy, B., Meier, W. (eds.) FSE 2004. LNCS, vol. 3017, pp. 332–347. Springer, Heidelberg (2004)
2. Akkar, M.L., Giraud, C.: An Implementation of DES and AES, Secure against Some Attacks. In: Koç, et al. (eds.) [14], pp. 309–318
3. Biham, E., Shamir, A.: Power Analysis of the Key Scheduling of the AES Candidates. In: Second AES Candidate Conference – AES 2 (March 1999), `http://csrc.nist.gov/encryption/aes/round1/conf2/aes2conf.htm`
4. Blömer, J., Guajardo, J., Krummel, V.: Provably Secure Masking of AES. In: Handschuh, H., Hasan, M.A. (eds.) SAC 2004. LNCS, vol. 3357, pp. 69–83. Springer, Heidelberg (2004)
5. Brier, E., Clavier, C., Olivier, F.: Correlation Power Analysis with a Leakage Model. In: Joye, Quisquater (eds.) [12], pp. 16–29
6. Chari, S., Jutla, C., Rao, J., Rohatgi, P.: Towards Sound Approaches to Counteract Power-Analysis Attacks. In: Wiener (ed.) [27], pp. 398–412
7. Coron, J.-S., Tchulkine, A.: A New Algorithm for Switching from Arithmetic to Boolean Masking. In: Walter, C.D., Koç, Ç.K., Paar, C. (eds.) CHES 2003. LNCS, vol. 2779, pp. 89–97. Springer, Heidelberg (2003)
8. Debraize, B.: Efficient and provably secure methods for switching from arithmetic to boolean masking. In: Prouff, E., Schaumont, P. (eds.) CHES 2012. LNCS, vol. 7428, pp. 107–121. Springer, Heidelberg (2012)
9. Genelle, L., Prouff, E., Quisquater, M.: Secure multiplicative masking of power functions. In: Zhou, J., Yung, M. (eds.) ACNS 2010. LNCS, vol. 6123, pp. 200–217. Springer, Heidelberg (2010)
10. Goubin, L.: A Sound Method for Switching between Boolean and Arithmetic Masking. In: Koç, et al. (eds.) [14], pp. 3–15
11. Goubin, L., Patarin, J.: DES and Differential Power Analysis – The "Duplication" Method. In: Koç, Ç.K., Paar, C. (eds.) CHES 1999. LNCS, vol. 1717, pp. 158–172. Springer, Heidelberg (1999)
12. Joye, M., Quisquater, J.-J. (eds.): CHES 2004. LNCS, vol. 3156. Springer, Heidelberg (2004)
13. Kim, H., Cho, Y.I., Choi, D., Han, D.G., Hong, S.: Efficient masked implementation for SEED based on combined masking. ETRI Journal 33(2), 267–274 (2011)
14. Koç, Ç.K., Naccache, D., Paar, C. (eds.): CHES 2001. LNCS, vol. 2162. Springer, Heidelberg (2001)
15. Kocher, P., Jaffe, J., Jun, B.: Differential Power Analysis. In: Wiener (ed.) [27], pp. 388–397
16. Lai, X., Massey, J.L.: A proposal for a new block encryption standard. In: Damgård, I.B. (ed.) EUROCRYPT 1990. LNCS, vol. 473, pp. 389–404. Springer, Heidelberg (1991)
17. Mangard, S., Oswald, E., Popp, T.: Power Analysis Attacks – Revealing the Secrets of Smartcards. Springer (2007)
18. Messerges, T.S.: Securing the AES Finalists Against Power Analysis Attacks. In: Schneier, B. (ed.) FSE 2000. LNCS, vol. 1978, pp. 150–164. Springer, Heidelberg (2001)
19. Neiße, O., Pulkus, J.: Switching Blindings with a View Towards IDEA. In: Joye, Quisquater (eds.) [12], pp. 230–239

20. Oswald, E., Mangard, S., Herbst, C., Tillich, S.: Practical Second-order DPA Attacks for Masked Smart Card Implementations of Block Ciphers. In: Pointcheval (ed.) [21], pp. 192–207
21. Pointcheval, D. (ed.): CT-RSA 2006. LNCS, vol. 3860. Springer, Heidelberg (2006)
22. Prouff, E., Rivain, M.: A Generic Method for Secure SBox Implementation. In: Kim, S., Yung, M., Lee, H.-W. (eds.) WISA 2007. LNCS, vol. 4867, pp. 227–244. Springer, Heidelberg (2008)
23. Rivain, M., Prouff, E.: Provably Secure Higher-Order Masking of AES. In: Mangard, S., Standaert, F.-X. (eds.) CHES 2010. LNCS, vol. 6225, pp. 413–427. Springer, Heidelberg (2010)
24. Rivain, M., Dottax, E., Prouff, E.: Block Ciphers Implementations Provably Secure Against Second Order Side Channel Analysis. In: Nyberg, K. (ed.) FSE 2008. LNCS, vol. 5086, pp. 127–143. Springer, Heidelberg (2008)
25. Schramm, K., Paar, C.: Higher Order Masking of the AES. In: Pointcheval (ed.) [21], pp. 208–225
26. Telecommunications Technology Association: 128-bit symmetric block cipher (SEED), Seoul, Korea (1998)
27. Wiener, M. (ed.): CRYPTO 1999. LNCS, vol. 1666. Springer, Heidelberg (1999)
28. Zabotin, I.A., Glazkov, G.P., Isaeva, V.B.: Cryptographic protection for information processing systems, government standard of the USSR, GOST 28147-89. Government Committee of the USSR for Standards (1989)

Low-Cost Countermeasure against RPA

Jean-Luc Danger[1,2], Sylvain Guilley[1,2], Philippe Hoogvorst[2],
Cédric Murdica[1,2], and David Naccache[3]

[1] Secure-IC S.A.S., 80 avenue des Buttes de Coësmes,
F-35700 Rennes, France
{jean-luc.danger,sylvain.guilley,cedric.murdica}@secure-ic.com
[2] Département COMELEC, Institut TELECOM,
TELECOM ParisTech, CNRS LTCI, Paris, France
{jean-luc.danger,sylvain.guilley,philippe.hoogvorst,
cedric.murdica}@telecom-paristech.fr
[3] École normale supérieure, Département d'informatique
45, rue d'Ulm, F-75230, Paris Cedex 05, France
david.naccache@ens.fr

Abstract. On smart-cards, Elliptic Curve Cryptosystems (ECC) can be vulnerable to Side Channel Attacks such as the Refined Power Analysis (RPA). This attack takes advantage of the apparition of special points of the form $(0, y)$. In this paper, we propose a new countermeasure based on co-Z formulæ and an extension of the curve isomorphism countermeasure. It permits to transform the base point $P = (x, y)$ into a base point $P' = (0, y')$, which, with $-P'$, are the only points with a zero X-coordinate. In such case, the RPA cannot be applied. Moreover, the cost of this countermeasure is very low compared to other countermeasures against RPA.

Keywords: Elliptic Curve Cryptosystem, Co-Z formulæ, Differential Power Analysis, Refined Power Analysis.

1 Introduction

The use of elliptic curves for cryptographic applications has been introduced by Koblitz [17] and Miller [23]. Elliptic Curve Cryptosystems (ECC) have gained much importance in smart-cards devices because of their better speed and low memory constraints compared to other asymmetric cryptosystems such as RSA.

The main operation on ECC is the computation of an elliptic curve scalar multiplication (ECSM), that is the computation of $[d]P$ for an integer d and a point P on an elliptic curve. The cryptographic security of ECC is based on the elliptic curve discrete logarithm problem (ECDLP), which asks to compute d given $Q = [d]P$ and P.

An ECSM is generally based on addition and doubling formulæ of points. Meloni points that addition formulæ of two points of an elliptic curve is more efficient if they share the same Z-coordinate [21]. He brought new formulæ, called

S. Mangard (Ed.): CARDIS 2012, LNCS 7771, pp. 106–122, 2013.

co-Z formulæ, that can be used to perform an ECSM with addition chains and Zeckendorf representation.

Meloni's formulæ were adapted in [8,10,9] so that they might be usable with traditional ECSM algorithms such as the right-to-left signed-digit method, the Montgomery Powering Ladder [16], or the Joye's double-add method [14].

In this paper, we are interested on the security against Side Channel Attacks. We present alternative co-Z formulæ using an extension of the curve isomorphism countermeasure [15]. The new co-Z formulæ allow to perform an ECSM that can be secured against SPA [18], DPA [19] and RPA [7] attacks. We give a comparison of different countermeasures against RPA. Our countermeasure has a very low cost compared to the other countermeasures.

The rest of the paper is structured as follows. In Section 2, we describe some properties on elliptic curves arithmetic and ECSM algorithms. In Section 3, we recall on the different side channel attacks, especially the RPA. Section 3 also gives countermeasures against RPA. Section 4 describes our countermeasure based on modified co-Z formulæ and an extension of the curve isomorphism countermeasure. We give a comparison of different countermeasures against the RPA in Section 5. Finally, we conclude in Section 6.

2 Elliptic Curve Arithmetic

In this paper, we are interested in elliptic curves based on field with characteristic greater than 3, and the given elliptic curves are in the reduced Weierstraß form.

However, our proposed countermeasure transforms the curve given into another one that is not in its short Weierstraß form. This is why we also give the formulæ for elliptic curve in the general Weierstraß form to understand our modified formulæ of Section 4.

2.1 Elliptic Curve Arithmetic in the Affine Coordinates System

In a finite field $\mathbb{K}$, an elliptic curve can be described by its Weierstraß form:

$$E: \ y^2 + a_1xy + a_3y = x^3 + a_2x^2 + a_4x + a_6 \ .$$

We denote by $E(\mathbb{K})$ the set of points $(x, y) \in \mathbb{K}^2$ satisfying the equation, plus the point at infinity $\mathcal{O}$. $E(\mathbb{K})$ has an Abelian group structure. Let $P = (x_1, y_1) \neq \mathcal{O}$ and $Q = (x_2, y_2) \notin \{\mathcal{O}, -P\}$ two points in $E(\mathbb{K})$. The point $R = (x_3, y_3) = P+Q$ can be computed as:

$$\begin{aligned} x_3 &= \lambda^2 + a_1\lambda - a_2 - x_1 - x_2 \\ y_3 &= \lambda(x_1 - x_3) - y_1 - a_1x_3 - a_3 \end{aligned} \qquad \text{where } \lambda = \begin{cases} \frac{y_1 - y_2}{x_1 - x_2} & \text{if } P \neq Q, \\ \frac{3x_1^2 + 2a_2x_1 + a_4 - a_1y_1}{2y_1 + a_1x_1 + a_3} & \text{if } P = Q. \end{cases}$$

The inverse of the point P is $-P = (x_1, -y_1 - a_1x_1 - a_3)$.

In a finite field $\mathbb{F}_p$, with p a prime such that $p > 3$, an elliptic curve can be described by its short Weierstraß form:

$$E : \ y^2 = x^3 + ax + b \ .$$

The point $R = (x_3, y_3) = P + Q$ can be computed as:

$$\begin{aligned} x_3 &= \lambda^2 - x_1 - x_2 \\ y_3 &= \lambda(x_1 - x_3) - y_1 \end{aligned} \qquad \text{where } \lambda = \begin{cases} \frac{y_1 - y_2}{x_1 - x_2} & \text{if } P \neq Q, \\ \frac{3x_1^2 + a}{2y_1} & \text{if } P = Q. \end{cases}$$

The inverse of the point P is $-P = (x_1, -y_1)$.

2.2 Elliptic Curve Arithmetic in the Jacobian Projective Coordinates System

To avoid costly inversions, one can use the Jacobian projective coordinates system. The equation of an elliptic curve in the Jacobian projective coordinates system in the reduced Weierstraß form is:

$$E^{\mathcal{J}} : \ Y^2 = X^3 + aXZ^4 + bZ^6 \ .$$

The point (X, Y, Z) corresponds to the affine point $(X/Z^2, Y/Z^3)$.

We give addition (ECADD) and doubling (ECDBL) formulæ in the Jacobian projective coordinates system. The formulæ are from [3].

Algorithm 1. ecdbl

```
Input: $P = (X_1, Y_1, Z_1) \in E^{\mathcal{J}}(\mathbb{F}_p)$
Output: $2P$

  $A \leftarrow X_1^2;\ B \leftarrow Y_1^2$
  $C \leftarrow B^2;\ D \leftarrow Z_1^2$
  $S \leftarrow 2((X_1 + B)^2 - A - C)$
  $M \leftarrow 3A + aD^2$
  $X_3 \leftarrow M^2 - 2S$
  $Y_3 \leftarrow M(S - X_3) - 8C$
  $Z_3 \leftarrow (Y_1 + Z_1)^2 - B - D$

  return $(X_3, Y_3, Z_3)$
```

Algorithm 2. ecadd

```
Input: $P = (X_1, Y_1, Z_1), Q = (X_2, Y_2, Z_2) \in E^{\mathcal{J}}(\mathbb{F}_p)$
Output: $P + Q$
  $A \leftarrow Z_1^2;\ B \leftarrow Z_2^2$
  $U_1 \leftarrow X_1 B;\ U_2 \leftarrow X_2 A$
  $S_1 \leftarrow Y_1 Z_2 B;\ S_2 \leftarrow Y_2 Z_1 A$
  $H \leftarrow U_2 - U1$
  $I \leftarrow (2H)^2$
  $J \leftarrow HI;\ K \leftarrow 2(S_2 - S_1);\ V \leftarrow U_1 I$
  $X_3 \leftarrow K^2 - J - 2V$
  $Y_3 \leftarrow K(V - X_3) - 2S_1 J$
  $Z_3 \leftarrow ((Z_1 + Z_2)^2 - A - B)H$
  return $(X_3, Y_3, Z_3)$
```

We denote by M, S the cost of field multiplication and field squaring respectively. We neglect the cost of additions and subtractions. ECDBL can be performed in $2M + 8S$ and ECADD can be performed in $11M + 5S$. Mixed addition (mECADD) is the addition of a point in Jacobian coordinates with a point in affine coordinates ($Z_2 = 1$). mECADD can be performed in $7M + 4S$ [3].

2.3 Elliptic Curve Arithmetic Using co-Z Formulæ

We describe here addition formulæ with points sharing the same Z-coordinate. Two procedures are presented. Addition and update in co-Z (ZADDU) is the procedure to compute $P+Q$ and update the point P to get the same Z-coordinate. It was introduced in [21]. Conjugate addition in co-Z (ZADDC) is the procedure to compute $P+Q$ and $P-Q$. It was introduced in [8].

Algorithm 3. co-Z addition and update (ZADDU)

```
Input: $P=(X_1,Y_1,Z), Q=(X_2,Y_2,Z) \in E^{\mathcal{J}}(\mathbb{F}_p)$
Output: $(R,S)$ with $R = P+Q$ and $S = (\lambda^2 X_1, \lambda^3 Y_1, \lambda Z)$ with $\lambda = X_1 - X_2$

  $C \leftarrow (X_1 - X_2)^2$
  $W_1 \leftarrow X_1 C$; $W_2 \leftarrow X_2 C$; $Z_3 \leftarrow Z(X_1 - X_2)$
  $D \leftarrow (Y_1 - Y_2)^2$; $A_1 \leftarrow Y_1(W_1 - W_2)$
  $X_3 \leftarrow D - W_1 - W_2$
  $Y_3 \leftarrow (Y_1 - Y_2)(W_1 - X_3) - A_1$
  $X_4 \leftarrow W_1$
  $Y_4 \leftarrow A_1$
  return $((X_3,Y_3,Z_3),(X_4,Y_4,Z_3))$
```

Algorithm 4. conjugate co-Z addition (ZADDC)

```
Input: $P=(X_1,Y_1,Z), Q=(X_2,Y_2,Z) \in E^{\mathcal{J}}(\mathbb{F}_p)$
Output: $(R,S)$ with $R = P+Q$, $S = P-Q$

  $C \leftarrow (X_1 - X_2)^2$
  $W_1 \leftarrow X_1 C$; $W_2 \leftarrow X_2 C$; $Z_3 \leftarrow Z(X_1 - X_2)$
  $D_1 \leftarrow (Y_1 - Y_2)^2$; $A_1 \leftarrow Y_1(W_1 - W_2)$
  $X_3 \leftarrow D_1 - W_1 - W_2$
  $Y_3 \leftarrow (Y_1 - Y_2)(W_1 - X_3) - A_1$
  $D_2 \leftarrow (Y_1 + Y_2)^2$
  $X_4 \leftarrow D_2 - W_1 - W_2$
  $Y_4 \leftarrow (Y_1 + Y_2)(W_1 - X_4) - A_1$
  return $((X_3,Y_3,Z_3),(X_4,Y_4,Z_3))$
```

Goundar et al. proposed in [9] an optimisation by removing the useless computation of the Z-coordinate. The formulæ are called the (X,Y)-only co-Z formulæ (ZACAU')[1]. ZACAU' (algorithm 15 in appendix) is a procedure computing the point $2P$ and $P+Q$. It can be performed in $8M+6S$ [9].

2.4 Elliptic Curve Scalar Multiplication

In elliptic curve cryptography, one has to compute scalar multiplications, *i.e.* compute $[d]P$, given the point P and a positive integer d.

The Montgomery Ladder is regular since the same operations are performed at each iteration independently of the current bit. Therefore it can be used to prevent the SPA. The Montgomery Ladder can be adapted with co-Z formulæ.

Algorithm 5. Montgomery Ladder

```
Input: $P \in E^{\mathcal{J}}(\mathbb{F}_p), d=(d_{n-1},\ldots,d_0)_2, d_{n-1}=1$
Output: $[d]P$
  $R_0 \leftarrow P, R_1 \leftarrow 2P$
  for $i = n-2$ downto $0$ do
      $R_{1-d_i} \leftarrow$ ECADD$(R_{1-d_i}, R_{1-d_i})$
      $R_{d_i} \leftarrow$ ECDBL$(R_{d_i})$
  end for
  return $R_0$
```

Algorithm 6. add only Montgomery Ladder using co-Z formulæ [8]

```
Input: $P \in E^{\mathcal{J}}(\mathbb{F}_p), d=(d_{n-1},\ldots,d_0)_2, d_{n-1}=1$
Output: $[d]P$
  $R_0 \leftarrow P, R_1 \leftarrow 2P$
  for $i = n-2$ downto $0$ do
      $(R_{1-d_i}, R_{d_i}) \leftarrow$ ZADDC$(R_{d_i}, R_{1-d_i})$
      $(R_{d_i}, R_{1-d_i}) \leftarrow$ ZADDU$(R_{1-d_i}, R_{d_i})$
  end for
  return $R_0$
```

Remark 1. In algorithm 6, the output point R_{d_i} of ZADDC is always equal to $\pm P$. Indeed, at the end of each iteration of the algorithm, R_0 and R_1 verify $R_1 = R_0 + P$.

[1] We use the same notation as in [9]: (') stands for formulæ that does not involve the Z-coordinate.

Using ZACAU', ZADDC' and ZADDU', the add only Montgomery Ladder using co-Z formulæ can be improved. See [9] for the justification to recover the Z coordinate.

Algorithm 7. Montgomery Ladder with (X, Y)-only co-Z formulæ [9]

```
Input: $P \in E^{\mathcal{J}}(\mathbb{F}_p), d = (d_{n-1}, \dots, d_0)_2, d_{n-1} = 1$
Output: $[d]P$

  $R_0 \leftarrow P, R_1 \leftarrow 2P$
  $C \leftarrow (X_{R_0} - X_{R_1})^2$
  for $i = n - 2$ downto $1$ do
      $(R_{d_i}, R_{1-d_i}, C) \leftarrow$ ZACAU'$(R_{d_i}, R_{1-d_i}, C)$
  end for
  $b \leftarrow d_0$; $(R_{1-b}, R_b) \leftarrow$ ZADDC'$(R_b, R_{1-b})$
  $Z \leftarrow x_P Y_{R_b}(X_{R_0} - X_{R_1})$; $\lambda \leftarrow y_P X_{R_b}$
  $(R_b, R_{1-b}) \leftarrow$ ZADDU'$(R_{1-b}, R_b)$

  return $\left((\frac{\lambda}{Z})^2 X_{R_0}, (\frac{\lambda}{Z})^3 Y_{R_0}\right)$
```

3 Side Channel Attacks

We describe in this section passive attacks such as DPA, RPA and ZPA.

3.1 DPA Attack and Countermeasures

If the same scalar d is used several times, the implementation can be vulnerable to the DPA [19]. The attacker recursively guesses the bits of the scalar and simulates the computation.

The countermeasures given below can be used to prevent the DPA.

Random Projective Coordinates [6]. A point $P = (X, Y, Z)$ in Jacobian coordinates is equivalent to any point (r^2X, r^3Y, rZ), with $r \in \mathbb{F}_p^*$. One can randomize the base point at the beginning of the ECSM by choosing a random r.

Random Curve Isomorphism [15]. A curve E defined by $E : y^2 = x^3 + ax + b$ in affine coordinates is isomorphic to the curve E' defined by $E' : y^2 = x^3 + a'x + b'$ if and only if there exists $u \in \mathbb{F}_p^*$ such that $u^4 a' = a$ and $u^6 b' = b$. The isomorphism φ is defined as:

$$\varphi : E \xrightarrow{\sim} E', \begin{cases} \mathcal{O} \rightarrow \mathcal{O} \\ (x, y) \rightarrow (u^{-2}x, u^{-3}y) \end{cases}$$

The countermeasure consists of computing the ECSM on a random curve E' instead of E.

Scalar Randomization [6]. Randomization of the scalar using $d' = d + r\sharp E$ is effective against DPA. r must be at least 32 bits, because attacks have been pointed out in [24] if r is small. For this reason, the ECSM is 32 iterations longer.

Random Scalar Split [5]. Random scalar splitting, such as computing $Q = [d_1]P + [d_2]P$ with $d = d_1 + d_2$, is effective against DPA. One can also use the euclidian splitting method [5]: compute $Q = [d_1]P + [d_2]S$ with $d_1 = d \bmod r$, $d_2 = \lfloor d/r \rfloor$ and $S = [r]P$ with r a random integer a half size of d. Ciet and Joye proposed in [5] to compute the point Q using a variant of Shamir's trick for efficiency (algorithm 13 in appendix). However, since they use four temporary points, they cannot use the co-Z formulæ. ECDBL and mECADD should be used instead. For the computation of $S = [r]P$, one can also use the variant of Shamir's trick with one of the scalar being zero.

Point Blinding [6]. Computing $Q = [d](P + R)$ instead of $[d]P$, with R a pseudo-random point is effective against DPA. The chip returns $Q - [d]R$. R and $[d]R$ are computed from R_0 and $[d]R_0$ precomputed and stored in the chip, with R_0 a random point.

This countermeasure was improved in [11] and later in [20]. The authors proposed to modify the ECSM for gradually subtract the random point R. The Binary Expansion with Random Initial Point (BRIP) can be found in appendix (algorithm 14). However, since they use three temporary points, they cannot use the co-Z formulæ. ECDBL and mECADD should be used instead.

3.2 RPA Attack

The RPA [7] is based on the apparition of a special point of the form $(0, y)$ during the ECSM[2].

Let $P_0 = (0, y)$ for some y. Suppose that the Montgomery Ladder (algorihm 5) is used to compute an ECSM. Suppose that the attacker already knows the $n - i - 1$ leftmost bits of the fixed scalar $d = (d_{n-1}, d_{n-2} \ldots, d_{i+1}, d_i, d_{i-1}, \ldots, d_0)_2$. He tries to recover the unknown bit d_i.

The attacker computes the point $P = [(d_{n-1}, d_{n-2}, \ldots, d_{i+1}, 0)_2^{-1} \bmod \sharp E]P_0$ and gives P to the targeted chip that computes $[d]P$. If $d_i = 0$, then the point P_0 will appear during the ECSM. If the attacker is able to recognize a zero value in a register, he can then conclude whether his hypothesis ($d_i = 0$) was correct or not.

3.3 ZPA Attack

The Zero-Value Point Attack (ZPA) [1] uses the same approach than the RPA, except that the attack is not only interested in zero values in coordinates but in intermediate registers when computing the double of a point, or during the addition of two points. Such points are defined as zero-value points.

[2] the point $(x, 0)$ can also be used but a point of this form is of order 2. In ECC, the order of the provided base point is checked and points of order 2 never appear during an ECSM.

Finding zero-value points for doubling formulæ consists in resolving polynomial equations in x, y with low degree (less than 4).

Finding zero-value points for addition is more difficult. For the Montgomery Ladder algorithm, suppose the attacker already knows the $n - i - 1$ leftmost bits of the fixed scalar $d = (d_{n-1}, d_{n-2}, \ldots, d_0)_2$ and try to recover d_i. With $c = (d_{n-1}, d_{n-2} \ldots, d_{i+1}, 0)_2$, he has to find a point P_0 such that $[c]P_0$ and $[c + 1]P_0$ are zero-value points. The only known procedure is using division polynomials and solve equations in two variables with degree of order $\mathcal{O}(c^2)$ [1]. At this day, when c is large, it is a hard problem. This problem was discussed in [12] and [1].

Remark 2. The random projective coordinates and the random curve isomorphism countermeasures described in the previous subsection fail against RPA and ZPA.

Some countermeasures to prevent RPA and ZPA are given below.

Isogeny Defence [25,2]. Computing an ECSM on a curve E' isogenous to E such that E' does not contain any non-trivial zero-value point is effective against the RPA and the ZPA.

Randomized Linearly Transformed Coordinates (RLC) [11]. This countermeasure consists in modifying the addition and doubling formulæ such that a zero value can never show up. A point $P = (X, Y, Z)$ is transformed into (X_μ, Y, Z, μ) with $X_\mu = X + \mu$, with μ a random field element. The potential zero value X is never manipulated alone. The countermeasure was given in [11] with classical doubling and addition formulæ. We adapted the countermeasure with the co-Z formulæ, because it is more efficient. We only modified the formulæ to prevent RPA because of the remark of the difficulty of the ZPA on addition formulæ. The countermeasure adapted with co-Z formulæ can be found in appendix (algorithm 16).

Remark 3. The scalar randomisation, random scalar split and point blinding countermeasures described in the previous subsection are also effective against RPA and ZPA.

4 Our Proposed Countermeasure

We describe in this section our new co-Z formulæ that we can use to perform a secured ECSM against the DPA and the RPA.

The main idea is to perform the ECSM with a base point $P' = (0, y')$. This point and its opposite $-P'$ are the only points with a zero X-coordinate. Using the ECSM add only Montgomery Ladder using co-Z formulæ (algorithm 6) with P' as the base point, we will show that the inputs of ZADDC are never $\pm P'$ whatever the value of the scalar. So the RPA cannot be performed.

The output point R_{d_i} of ZADDC is always equal to $\pm P'$ (see remark 1). So $\pm P' = (0, \pm y')$ appears at the end of ZADDC and therefore appears at the beginning of ZADDU. Algorithms 3 and 4 can be modified by removing the useless multiplications and additions with the zero-value.

Algorithm 8. co-Z addition and update with a zero value (ZADDUzero)

```
Input: $P = (X_1, Y_1, Z), Q = (0, Y_2, Z) \in E^{\mathcal{J}}(\mathbb{F}_p)$
Output: $(R, S)$ with $R = P + Q$ and $S = (\lambda^2 X_1, \lambda^3 Y_1, \lambda Z)$ with $\lambda = X_1$

  $C \leftarrow X_1^2$
  $W_1 \leftarrow X_1 C;\ Z_3 = Z X_1$
  $D \leftarrow (Y_1 - Y_2)^2;\ A_1 \leftarrow Y_1 W_1$
  $X_3 \leftarrow D - W_1$
  $Y_3 \leftarrow (Y_1 - Y_2)(W_1 - X_3) - A_1$
  $X_4 \leftarrow W_1;\ Y_4 \leftarrow A_1$

  return $((X_3, Y_3, Z_3), (X_4, Y_4, Z_3))$
```

Algorithm 9. conjugate co-Z addition with a zero value (ZADDCzero)

```
Input: $P = (X_1, Y_1, Z), Q = (X_2, Y_2, Z) \in E^{\mathcal{J}}(\mathbb{F}_p)$,
  such that $x_{P-Q} = 0$
Output: $(R, S)$ with $R = P + Q$, $S = P - Q$

  $C \leftarrow (X_1 - X_2)^2$
  $W_1 \leftarrow X_1 C;\ W_2 \leftarrow X_2 C;\ Z_3 \leftarrow Z(X_1 - X_2)$
  $D \leftarrow (Y_1 - Y_2)^2;\ A_1 = Y_1(W_1 - W_2)$
  $X_3 \leftarrow D - W_1 - W_2$
  $Y_3 \leftarrow (Y_1 - Y_2)(W_1 - X_3) - A_1$
  $Y_4 \leftarrow (Y_1 + Y_2) W_1 - A_1$

  return $((X_3, Y_3, Z_3), (0, Y_4, Z_3))$
```

ZADDUzero and ZADDCzero can be combined without the Z-coordinate: ZACAUzero'. The algorithm is given in appendix (algorithm 17). ZACAUzero' requires one multiplication and one square less compared to ZACAU' (algorithm 15).

The main step is to find a method to transform any base point $P = (x, y)$ of any curve into a base point $P' = (0, y')$.

We present in this paper two methods of such a transformation. The first method uses isogenies and was proposed in [2]. The second method is an extension of the random curve isomorphism countermeasure.

4.1 Transformation of the Base Point Using Isogenies

An isogeny between two elliptic curves E and E' defined over $\mathbb{F}_p$ is a nonconstant morphism $\phi : E \to E'$. Every isogeny has a finite kernel and the size of this kernel is called the degree of isogeny.

Brier and Joye introduced in [4] the use of isogenies for efficiency: they transform an elliptic curve $E : y^2 = x^3 + ax + b$ into an elliptic curve $E' : y^2 = x^3 - 3x + b'$. The parameter $a' = -3$ brings better performance for doubling formulæ.

Smart proposed in [25] to use isogenies as a countermeasure against the RPA [7]. ECSMs are performed on an isogenous elliptic curve that does not contain any special point of the form $(0, y)$. Akishita and Takagi extended the isogeny defense in [2] so it can also prevent the ZPA. They also use isogenies for efficiency for binary ECSM methods. The given elliptic curve is transformed into an isogenous curve where the base point G' has the particular form $G' = (0, y')$, for that the addition with the point G' is more efficient because of the zero value.

Finding isogenies for a given elliptic curve is not trivial. Isogenies of standardized curves are precomputed and stored in the chip. The base point given also needs to be mapped in the isogenous curve. This transformation has a non negligible cost which is discussed in [25].

4.2 Transformation of the Base Point Using Isomorphism

We propose here a more practical method to transform the base point that can work to any arbitrary curve. We extend the isomorphic curve countermeasure proposed in [15]. We need the following corollary of the theorem [22, Theorem 2.2].

Corollary 1. *Let $\mathbb{F}_p$ be a finite field with a prime $p > 3$. The elliptic curves given by the Weierstraß equations*

$$\begin{aligned} E\ &: y^2 = x^3 + a_4 x + a_6 \\ E'\ &: y^2 = x^3 + a'_2 x^2 + a'_4 x + a'_6 \end{aligned}$$

are isomorphic over $\mathbb{F}_p$ if and only if there exist $u \in \mathbb{F}_p^*$ and $r \in \mathbb{F}_p$ such that the change of variables

$$(x, y) \rightarrow (u^{-2}(x - r), u^{-3}y)$$

transforms equation E into equation E'. Such a transformation is referred to as an admissible change of variables. Furthermore,

$$\begin{cases} u^2 a'_2 = 3r \\ u^4 a'_4 = a_4 + 3r^2 \\ u^6 a'_6 = a_6 + r a_4 + r^3 \ . \end{cases}$$

Proof. The corollary is simply a particular case of the theorem [22, Theorem 2.2] with $s = t = a_1 = a_2 = a_3 = a'_1 = a'_3 = 0$. □

Remark 4. In the isomorphic curve countermeasure [15], the isomorphic curve E' is also in its short short Weierstraß form for efficiency reason. Therefore $a'_2 = 0$. This implies $r = 0$. Only u is randomly chosen for the countermeasure.

The ECSM add only Montgomery Ladder using co-Z formulæ (algorithm 6) has the following properties:

- the base point P or its opposite $-P$ appears at each iteration
- a point doubling is never performed in the main loop

The goal is to perform an ECSM with a base point of the form $P' = (0, y_{P'})$. If the base point given is $P = (x_P, y_P)$ on the elliptic curve E, one can choose $r = x_P$ and a random u so that the isomorphism:

$$\varphi : E \xrightarrow{\sim} E', \begin{cases} \mathcal{O} \rightarrow \mathcal{O} \\ (x, y) \rightarrow (u^{-2}(x - r), u^{-3}y) \end{cases}$$

maps the point $P = (x_P, y_P)$ into the point $P' = (0, u^{-3}y_P)$. However, the elliptic curve E' is not in the short Weierstraß form: the parameter a'_2 is non-zero. Thanks to the add only Montgomery Ladder using co-Z formulæ (algorithm 6), a doubling is never performed. Only the addition has to be modified. Using Section 2 to see the modifications due to the non-zero a'_2 parameter on co-Z formulæ, we can give the modified co-Z formulæ.

Algorithm 10. co-Z addition and update with a zero value and a'_2 parameter (ZADDUazero)

```
Input: $P = (X_1, Y_1, Z), Q = (0, Y_2, Z) \in E'^{\mathcal{J}}(\mathbb{F}_p)$ and $T_a = a'_2 Z^2$
Output: $(R, S, T_a)$ with $R = P + Q$, $S = (\lambda^2 X_1, \lambda^3 Y_1, \lambda Z)$ with $\lambda = X_1$ and $T_a = a'_2 Z_3^2$

  $C \leftarrow X_1^2$
  $W_1 \leftarrow X_1 C$;  $Z_3 \leftarrow Z X_1$;  $T_a \leftarrow T_a C$
  $D \leftarrow (Y_1 - Y_2)^2$;  $A_1 \leftarrow Y_1 W_1$
  $X_3 \leftarrow D - W_1 - T_a$
  $Y_3 \leftarrow (Y_1 - Y_2)(W_1 - X_3) - A_1$
  $X_4 \leftarrow W_1$;  $Y_4 \leftarrow A_1$
  return $((X_3, Y_3, Z_3), (X_4, Y_4, Z_3), T_a)$
```

Algorithm 11. conjugate co-Z addition with a zero value and a'_2 parameter (ZADDCazero)

```
Input: $P = (X_1, Y_1, Z), Q = (X_2, Y_2, Z) \in E'^{\mathcal{J}}(\mathbb{F}_p)$, such that $x_{P-Q} = 0$ and $T_a = a'_2 Z^2$
Output: $(R, S, T_a)$ with $R = P + Q$, $S = P - Q$ and $T_a = a'_2 Z_3^2$

  $C \leftarrow (X_1 - X_2)^2$
  $W_1 \leftarrow X_1 C$;  $W_2 \leftarrow X_2 C$;  $Z_3 \leftarrow Z(X_1 - X_2)$
  $T_a \leftarrow T_a C$
  $D \leftarrow (Y_1 - Y_2)^2$;  $A_1 \leftarrow Y_1(W_1 - W_2)$
  $X_3 \leftarrow D - W_1 - W_2 - T_a$
  $Y_3 \leftarrow (Y_1 - Y_2)(W_1 - X_3) - A_1$
  $Y_4 \leftarrow (Y_1 + Y_2)W_1 - A_1$
  return $((X_3, Y_3, Z_3), (0, Y_4, Z_3), T_a)$
```

The combination of the two formulæ without the Z-coordinate is given in appendix (algorithm 18). We called it ZACAUazero'. ZACAUazero' requires $9M + 5S$. That is one multiplication more and one square less than ZACAU'.

The complete ECSM using the countermeasure and the modified co-Z formulæ is given below.

Algorithm 12. (X, Y)-only add only Montgomery Ladder using modified co-Z formulæ

```
Input: $P \in E^{\mathcal{J}}(\mathbb{F}_p), d = (d_{n-1}, \ldots, d_0)_2, d_{n-1} = 1$
Output: $[d]P$

  $u \overset{R}{\leftarrow} \mathbb{F}_p^*$
  $P' \leftarrow (0, u^{-3} y_P, 1)$                                              ▷ isomorphism
  $T_a \leftarrow 3 x_P u^{-2}$                      ▷ $T_a$ is the parameter $a'_2$ of the isomorphic curve $E'$
  $R_0 \leftarrow P', R_1 \leftarrow 2P'$                  ▷ $R_0$ and $R_1$ must share the same $Z$-coordinate
  $C \leftarrow (X_{R_0} - X_{R_1})^2 = X_{R_1}^2$
  for $i = n - 2$ downto $1$ do
      $(R_{d_i}, R_{1-d_i}, C, T_a) \leftarrow$ ZACAUazero'$(R_{d_i}, R_{1-d_i}, C, T_a)$
  end for
  $b \leftarrow d_0$;  $(R_{1-b}, R_b, T_a) \leftarrow$ ZADDCazero'$(R_b, R_{1-b}, T_a)$
  $Z \leftarrow 3 x_P u^{-2} Y_{R_b}(X_{R_0} - X_{R_1})$;  $\lambda \leftarrow u^{-3} y_P T_a$     ▷ $Z = a'_2 Y_{R_b}(X_{R_0} - X_{R_1}),\ \lambda = u^{-3} y_P a'_2 Z_{R_0}^2$
  $(R_b, R_{1-b}, T_a) \leftarrow$ ZADDUazero'$(R_{1-b}, R_b, T_a)$
  $(x', y') \leftarrow \left(\left(\frac{\lambda}{Z}\right)^2 X_{R_0}, \left(\frac{\lambda}{Z}\right)^3 Y_{R_0}\right)$
  $(x, y) \leftarrow (u^2 x' + x_P, u^3 y)$                                      ▷ isomorphism inverse

  return $(x, y)$
```

4.3 Security Analysis of our Countermeasure

The security against RPA is based on the fact that the base point P' has a x-zero coordinate. The only possible points having a x-zero coordinate are P' and $-P'$. These two points can never appear as inputs of ZACAUazero'. Joye was the first to propose in [13] an extension of the random curve isomorphism countermeasure to prevent the RPA. His countermeasure can be applied for elliptic curves on binary fields of the form $y^2 + xy = x^3 + a_2x^2 + a_6$, so choosing a random r for the isomorphism of theorem [22, Theorem 2.2] does not affect the efficiency. In this paper, we introduced the extension of the random isomorphic curve countermeasure on elliptic curve over field of large characteristic without any efficiency loss.

SPA Security. Our ECSM is regular: the same operation ZACAUazero' is performed whatever the value of the current bit. The classical SPA where an attacker is able to distinguish different patterns depending on the value of the current bit cannot be applied.

DPA Security. The random parameter u gives the security against DPA. All values and intermediates values in ZACAUazero' (algorithm 18) are multiplicatively randomized by u.

RPA Security. The RPA security is provided by the following lemma.

Lemma 1. *Suppose d satisfies $1 < d < ord(P)$. The points R_0 and R_1 at the beginning of each iteration $1 \leq i \leq n-3$ in algorithm 12 cannot take the values $\pm P'$.*

Proof. Suppose the ECSM is performed with the scalar $d = (d_{n-1}, d_{n-2}, \ldots, d_0)_2$ and the base point P'. Let $c_i = (d_{n-1}, d_{n-2} \ldots, d_{i+1})_2$. At the beginning of iteration i with $1 \leq i \leq n-3$, the points R_0, R_1 verify $R_0 = c_iP'$ and $R_1 = [c_i + 1]P'$.

- if $R_0 = [c_i]P' = P'$, then $[c_i - 1]P' = \mathcal{O}$ so the order of P' and P is $(c_i - 1)$, which is impossible by the condition of d.
- if $R_0 = [c_i]P' = -P'$, then $[c_i + 1]P' = \mathcal{O}$ so the order of P' and P is $(c_i + 1)$, which is impossible by the condition of d.
- if $R_1 = [c_i + 1]P' = P'$, then $[c_i]P' = \mathcal{O}$ so the order of P' and P is c_i, which is impossible by the condition of d.
- if $R_1 = [c_i + 1]P' = -P'$, then $[c_i + 2]P' = \mathcal{O}$ so the order of P' and P is $(c_i + 2)$, which is impossible by the condition of d.

By contradiction, we prove that the points R_0, R_1 cannot take the values P' or $-P'$. □

An elliptic curve $E' : y^2 = x^3 + a'_2x^2 + a'_4x + a'_6$ contains at most two points of the form $(0, y')$. Those points are $P' = (0, \sqrt{a'_6})$ and $-P' = (0, -\sqrt{a'_6})$. The points P' and $-P'$ are the only points with a zero x-coordinate. With the lemma, we can state that an attacker is not able to perform a RPA attack because the zero-value points never appear in outputs of ZACAUazero'.

ZPA Security. The ZPA security is not guaranteed. However, the ZPA remains difficult because no doubling is performed during the ECSM. The attacker has to find zero-value points for addition which is a difficult problem [12,1]. This is discussed in Section 3.

5 Comparison with Prior RPA Countermeasures

In this section, we compare different countermeasures against RPA described in Section 3.

We can see that if the cost of a multiplication and a square is the same, our countermeasure does not bring any additional cost. The isogeny defence does not bring any additional cost as well but it does not work to any curve: the isogenous curves have to be precomputed and stored in the chip. Moreover, the base point given has to be mapped to the isogenous curve, so the countermeasure has an extra cost. The cost of isogeny is approximatively $3l$ multiplications with l the degree of isogeny [25].

ECSM	Countermeasure	Cost per bit	Cost of ECSM	works to any curve
(X,Y)-only co-Z Montgomery Ladder	none (reference) [9]	$8M + 6S$	$n(8M + 6S)$	✓
(X,Y)-only co-Z Montgomery Ladder	$d' = d + r\sharp E$ [6]	$8M + 6S$	$(n + 32)(8M + 6S)$	✓
Regular Shamir's trick	random scalar split [5]	$8M + 12S$	$n(8M + 12S)$	✓
(X,Y)-only co-Z Montgomery Ladder	isogeny defense [25,2]	$8M + 6S$	$n(8M + 6S)$	× (isogenous curves precomputed)
BRIP algorithm 14	BRIP [20]	$8M + 12S$	$n(8M + 12S)$	× (initial random point precomputed)
(X,Y)-only co-Z Montgomery Ladder with RLC	Random Linear Coordinates [11]	$10M + 6S$	$n(10M + 6S)$	✓
(X,Y)-only co-Z Montgomery Ladder with ZACAUazero’	this paper	$9M + 5S$	$n(9M + 5S)$	✓

6 Conclusion

We presented in this paper a secured ECSM where the base point is of the form $P' = (0, y')$. The base point P given is transformed into P' using an extension of the isomorphic curve countermeasure [15]. The ECSM is secured against DPA [19] and RPA [7]. Moreover, thanks to co-Z formulæ, a doubling is never performed during the main loop, so the ZPA [1] remains a hard problem. A comparison of different countermeasure against RPA is also given. Using modified co-Z formulæ, the loss of efficiency is negligible, and our countermeasure is the most efficient.

Further work is to guarantee the security against the ZPA with either finding formulæ with no zero-value point or calculating the cost of finding zero-value points for addition. Also, a comparison of the memory cost of countermeasures against RPA is missing.

References

1. Akishita, T., Takagi, T.: Zero-Value Point Attacks on Elliptic Curve Cryptosystem. In: Boyd, C., Mao, W. (eds.) ISC 2003. LNCS, vol. 2851, pp. 218–233. Springer, Heidelberg (2003)
2. Akishita, T., Takagi, T.: On the Optimal Parameter Choice for Elliptic Curve Cryptosystems Using Isogeny. In: Bao, F., Deng, R., Zhou, J. (eds.) PKC 2004. LNCS, vol. 2947, pp. 346–359. Springer, Heidelberg (2004)
3. Bernstein, D.J., Lange, T.: Explicit-formulas database (2004), `http://hyperelliptic.org/EFD`
4. Brier, E., Joye, M.: Fast Point Multiplication on Elliptic Curves through Isogenies. In: Fossorier, M.P.C., Høholdt, T., Poli, A. (eds.) AAECC 2003. LNCS, vol. 2643, pp. 43–50. Springer, Heidelberg (2003)
5. Ciet, M., Joye, M.: (Virtually) Free Randomization Techniques for Elliptic Curve Cryptography. In: Qing, S., Gollmann, D., Zhou, J. (eds.) ICICS 2003. LNCS, vol. 2836, pp. 348–359. Springer, Heidelberg (2003)
6. Coron, J.-S.: Resistance against Differential Power Analysis for Elliptic Curve Cryptosystems. In: Koç, Ç.K., Paar, C. (eds.) CHES 1999. LNCS, vol. 1717, pp. 292–302. Springer, Heidelberg (1999)
7. Goubin, L.: A Refined Power-Analysis Attack on Elliptic Curve Cryptosystems. In: Desmedt, Y.G. (ed.) PKC 2003. LNCS, vol. 2567, pp. 199–211. Springer, Heidelberg (2003)
8. Goundar, R.R., Joye, M., Miyaji, A.: Co-Z Addition Formulæ and Binary Ladders on Elliptic Curves - (Extended Abstract). In: Mangard, S., Standaert, F.-X. (eds.) CHES 2010. LNCS, vol. 6225, pp. 65–79. Springer, Heidelberg (2010)
9. Goundar, R.R., Joye, M., Miyaji, A., Rivain, M., Venelli, A.: Scalar multiplication on Weierstraß elliptic curves from Co-Z arithmetic. Journal of Cryptographic Engineering 1, 161–176 (2011)
10. Hutter, M., Joye, M., Sierra, Y.: Memory-Constrained Implementations of Elliptic Curve Cryptography in Co-Z Coordinate Representation. In: Nitaj, A., Pointcheval, D. (eds.) AFRICACRYPT 2011. LNCS, vol. 6737, pp. 170–187. Springer, Heidelberg (2011)
11. Itoh, K., Izu, T., Takenaka, M.: Efficient Countermeasures against Power Analysis for Elliptic Curve Cryptosystems. In: Proceedings of CARDIS 2004, pp. 99–114. Kluwer Academic Publishers (2004)
12. Izu, T., Takagi, T.: Exceptional Procedure Attackon Elliptic Curve Cryptosystems. In: Desmedt, Y.G. (ed.) PKC 2003. LNCS, vol. 2567, pp. 224–239. Springer, Heidelberg (2003)
13. Joye, M.: Smart-Card Implementation of Elliptic Curve Cryptography and DPA-type Attacks. In: Proceedings of CARDIS 2004, pp. 115–126. Kluwer Academic Publisher (2004)
14. Joye, M.: Highly Regular Right-to-Left Algorithms for Scalar Multiplication. In: Paillier, P., Verbauwhede, I. (eds.) CHES 2007. LNCS, vol. 4727, pp. 135–147. Springer, Heidelberg (2007)

15. Joye, M., Tymen, C.: Protections against Differential Analysis for Elliptic Curve Cryptography. In: Koç, Ç.K., Naccache, D., Paar, C. (eds.) CHES 2001. LNCS, vol. 2162, pp. 377–390. Springer, Heidelberg (2001)
16. Joye, M., Yen, S.-M.: The Montgomery Powering Ladder. In: Kaliski Jr., B.S., Koç, Ç.K., Paar, C. (eds.) CHES 2002. LNCS, vol. 2523, pp. 291–302. Springer, Heidelberg (2003)
17. Koblitz, N.: Elliptic Curve Cryptosystems. J. Mathematics of Computation 48, 203–209 (1987)
18. Kocher, P.C.: Timing Attacks on Implementations of Diffie-Hellman, RSA, DSS, and Other Systems. In: Koblitz, N. (ed.) CRYPTO 1996. LNCS, vol. 1109, pp. 104–113. Springer, Heidelberg (1996)
19. Kocher, P.C., Jaffe, J., Jun, B.: Differential Power Analysis. In: Wiener, M. (ed.) CRYPTO 1999. LNCS, vol. 1666, pp. 388–397. Springer, Heidelberg (1999)
20. Mamiya, H., Miyaji, A., Morimoto, H.: Efficient Countermeasures against RPA, DPA, and SPA. In: Joye, M., Quisquater, J.-J. (eds.) CHES 2004. LNCS, vol. 3156, pp. 343–356. Springer, Heidelberg (2004)
21. Meloni, N.: New Point Addition Formulae for ECC Applications. In: Carlet, C., Sunar, B. (eds.) WAIFI 2007. LNCS, vol. 4547, pp. 189–201. Springer, Heidelberg (2007)
22. Menezes, A.J.: Elliptic Curve Public Key Cryptosystems. Kluwer Academic Publishers (1993)
23. Miller, V.S.: Use of elliptic curves in cryptography. In: Williams, H.C. (ed.) CRYPTO 1985. LNCS, vol. 218, pp. 417–426. Springer, Heidelberg (1986)
24. Okeya, K., Sakurai, K.: Power Analysis Breaks Elliptic Curve Cryptosystems Even Secure against the Timing Attack. In: Roy, B., Okamoto, E. (eds.) INDOCRYPT 2000. LNCS, vol. 1977, pp. 178–190. Springer, Heidelberg (2000)
25. Smart, N.P.: An Analysis of Goubin's Refined Power Analysis Attack. In: Walter, C.D., Koç, Ç.K., Paar, C. (eds.) CHES 2003. LNCS, vol. 2779, pp. 281–290. Springer, Heidelberg (2003)

A Elliptic Curve Scalar Multiplication Algorithms

Algorithm 13. Variant of Shamir's trick [5]

```
Input: $P, S \in E^{\mathcal{J}}(\mathbb{F}_p), k = (k_{n-1}, \ldots, k_0)_2,$
  $d = (d_{n-1}, \ldots, d_0)_2$ with $(k_{n-1}, d_{n-1}) \neq (0, 0)$
Output: $[k]P + [d]S$
  $R_1 \leftarrow P;\ R_2 \leftarrow S;\ R_3 \leftarrow P + S;\ R_4 \leftarrow P + S$
  $c \leftarrow 2d_{n-1} + k_{n-1};\ R_0 \leftarrow R_c$
  for $i = n - 2$ downto 0 do
      $R_0 \leftarrow \textsc{ecdbl}(R_0)$
      $b \leftarrow \neg(k_i \vee d_i);\ c \leftarrow 2d_i + k_i$
      $R_{4b} \leftarrow \textsc{mecadd}(R_{4b}, R_c)$
  end for
  return $R_0$
```

Algorithm 14. BRIP [20]

```
Input: $d = (d_{n-1}, \ldots, d_0)_2, P$
Output: $[d]P$

  $R \leftarrow randompoint()$
  $R_0 \leftarrow R,\ R_1 \leftarrow -R, R_0 = P - R$
  for $i = n - 1$ downto 0 do
      $R_0 \leftarrow \textsc{ecdbl}(R_0)$
      $R_0 \leftarrow \textsc{mecadd}(R_0, R_{d_i+1})$
  end for

  return $R_0 + R_1$
```

B co-Z Formulæ

Algorithm 15. (X, Y)-only co-Z conjugate-addition-addition with update (ZACAU') [9]

Input: $(X_1, Y_1), (X_2, Y_2), C$ with $P = (X_1, Y_1, Z), Q = (X_2, Y_2, Z) \in E^{\mathcal{J}}(\mathbb{F}_p)$ and $C = (X_1 - X_2)^2$
Output: $(X_3, Y_3), (X_4, Y_4), C$ with $R = (X_3, Y_3, Z_3)$, $S = (X_4, Y_4, Z_3)$ such that $R = 2P, S = P + Q$ and $C = (X_3 - X_4)^2$

```
$W_1 \leftarrow X_1 C;\ W_2 \leftarrow X_2 C$
$D_1 \leftarrow (Y_1 - Y_2)^2;\ A_1 \leftarrow Y_1(W_1 - W_2)$
$X_1' \leftarrow D_1 - W_1 - W_2;\ Y_1' \leftarrow (Y_1 - Y_2)(W_1 - X_1') - A_1$
$D_2 \leftarrow (Y_1 + Y_2)^2$
$X_2' \leftarrow D_2 - W_1 - W_2;\ Y_2' \leftarrow (Y_1 + Y_2)(W_1 - X_2') - A_1$
$C' \leftarrow (X_1' - X_2')^2$
$X_4 \leftarrow X_1' C';\ W_2' \leftarrow X_2' C'$
$D' \leftarrow (Y_1' - Y_2')^2;\ Y_4 \leftarrow Y_1'(X_4 - W_2')$
$X_3 \leftarrow D' - X_4 - W_2'$
$C \leftarrow (X_3 - X_4)^2$
$Y_3 \leftarrow (Y_1' - Y_2' + X_4 - X_3)^2 - D' - C - 2Y_4$
$X_3 \leftarrow 4X_3;\ Y_3 \leftarrow 4Y_3;\ X_4 \leftarrow 4X_4$
$Y_4 \leftarrow 8Y_4;\ C \leftarrow 16C$
return $(X_3, Y_3), (X_4, Y_4), C$
```

Algorithm 16. (X, Y)-only co-Z conjugate-addition-addition with update using RLC

Input: $(X_{1,\mu}, Y_1), (X_{2,\mu}, Y_2), C, \mu$ with $P = (X_1, Y_1, Z), Q = (X_2, Y_2, Z) \in E^{\mathcal{J}}(\mathbb{F}_p)$ with $X_{1,\mu} = X_1 + \mu, X_{2,\mu} = X_2 + \mu$ and $C = (X_1 - X_2)^2$
Output: $(X_{3,\mu}, Y_3), (X_{4,\mu}, Y_4), C$ with $R = (X_3, Y_3, Z_3)$, $S = (X_4, Y_4, Z_3)$ such that $R = 2P, S = P + Q$ with $X_{3,\mu} = X_3 + \mu, X_{4,\mu} = X_4 + \mu$ and $C = (X_3 - X_4)^2$

```
$W_1 \leftarrow X_{1,\mu} C;\ W_2 \leftarrow X_{2,\mu} C$
$C_\mu = \mu C$
$D \leftarrow (Y_1 - Y_2)^2;\ A_1 \leftarrow Y_1(W_1 - W_2)$
$X_{1,\mu}' \leftarrow D - W_1 - W_2 + 2C_\mu + \mu;\ Y_1' \leftarrow (Y_1 - Y_2)(W_1 - X_{1,\mu}' + \mu - 2C_\mu) - A_1$
$\bar{D} \leftarrow (Y_1 + Y_2)^2$
$X_{2,\mu}' \leftarrow \bar{D} - W_1 - W_2 + 2C_\mu + \mu;\ Y_2' \leftarrow (Y_1 + Y_2)(W_1 - X_{2,\mu}' + \mu - 2C_\mu) - A_1$
$C' \leftarrow (X_{1,\mu}' - X_{2,\mu}')^2$
$C_\mu' = \mu C'$
$X_{4,\mu} \leftarrow X_{1,\mu}' C';\ W_2' \leftarrow X_{2,\mu}' C'$
$D' \leftarrow (Y_1' - Y_2')^2;\ Y_4 \leftarrow Y_1'(X_{4,\mu} - W_2' + \mu - 2C_\mu')$
$X_{3,\mu} \leftarrow D' - X_{4,\mu} - W_2' + 2C_\mu' + \mu$
$X_{4,\mu} \leftarrow X_{4,\mu} - C_\mu' + \mu$
$C \leftarrow (X_{3,\mu} - X_{4,\mu})^2$
$Y_3 \leftarrow (Y_1' - Y_2' + X_{4,\mu} - X_{3,\mu})^2 - D' - C - 2Y_4$
$X_{3,\mu} \leftarrow 4X_{3,\mu} - 3\mu;\ Y_3 \leftarrow 4Y_3;\ X_{4,\mu} \leftarrow 4X_{4,\mu} - 3\mu$
$Y_4 \leftarrow 8Y_4;\ C \leftarrow 16C$
return $(X_{3,\mu}, Y_3), (X_{4,\mu}, Y_4), C$
```

Algorithm 17. (X, Y)-only co-Z conjugate-addition-addition with a zero value (ZACAUzero’)

Input: $(X_1, Y_1), (X_2, Y_2), C$ with $P = (X_1, Y_1, Z), Q = (X_2, Y_2, Z) \in E'^{\mathcal{J}}(\mathbb{F}_p)$ such that $x_{P-Q} = 0$ and $C = (X_1 - X_2)^2$
Output: $(X_3, Y_3), (X_4, Y_4), C$ with $R = (X_3, Y_3, Z_3)$, $S = (X_4, Y_4, Z_3)$ such that $R = 2P, S = P + Q$ and $C = (X_3 - X_4)^2$

```
$W_1 \leftarrow X_1 C;\ W_2 \leftarrow X_2 C$
$D \leftarrow (Y_1 - Y_2)^2;\ A_1 \leftarrow Y_1(W_1 - W_2)$
$X_1' \leftarrow D - W_1 - W_2;\ Y_1' \leftarrow (Y_1 - Y_2)(W_1 - X_1') - A_1$
$Y_2' \leftarrow (Y_1 + Y_2)W_1 - A_1$
$C' \leftarrow (X_1' - X_2')^2$
$X_4 \leftarrow X_1' C'$
$D' \leftarrow (Y_1' - Y_2')^2;\ Y_4 \leftarrow Y_1' X_4$
$X_3 \leftarrow D' - X_4$
$C \leftarrow (X_3 - X_4)^2$
$Y_3 \leftarrow (Y_1' - Y_2' + X_4 - X_3)^2 - D' - C - 2Y_4$
$X_3 \leftarrow 4X_3;\ Y_3 \leftarrow 4Y_3;\ X_4 \leftarrow 4X_4$
$Y_4 \leftarrow 8Y_4;\ C \leftarrow 16C$
return $(X_3, Y_3), (X_4, Y_4), C$
```

Algorithm 18. (X, Y)-only co-Z conjugate-add-add with a zero value and a_2' (ZACAUazero’)

Input: X_1, Y_1, X_2, T_a, C with $P = (X_1, Y_1, Z), Q = (X_2, Y_2, Z) \in E'^{\mathcal{J}}(\mathbb{F}_p)$ such that $x_{P-Q} = 0$, $T_a = a_2' Z^2$ and $C = (X_1 - X_2)^2$
Output: $(X_3, Y_3), (X_4, Y_4), T_a, C$ with $R = (X_3, Y_3, Z_3)$, $S = (X_4, Y_4, Z_3)$ such that $R = 2P, S = P + Q$, $T_a = a_2' Z_3^2$ and $C = (X_3 - X_4)^2$

```
$W_1 \leftarrow X_1 C;\ W_2 \leftarrow X_2 C;\ T_a \leftarrow T_a C$
$D \leftarrow (Y_1 - Y_2)^2;\ A_1 \leftarrow Y_1(W_1 - W_2)$
$X_1' \leftarrow D - W_1 - W_2 - T_a$
$Y_1' \leftarrow (Y_1 - Y_2)(W_1 - X_1') - A_1;\ Y_2' \leftarrow (Y_1 + Y_2)W_1 - A_1$
$C' \leftarrow (X_1' - X_2')^2$
$X_4 \leftarrow X_1' C';\ T_a \leftarrow T_a C'$
$D' \leftarrow (Y_1' - Y_2')^2;\ Y_4 \leftarrow Y_1' X_4$
$X_3 \leftarrow D' - X_4 - T_a$
$C \leftarrow (X_3 - X_4)^2$
$Y_3 \leftarrow (Y_1' - Y_2' + X_4 - X_3)^2 - D - C - 2Y_4$
$X_3 \leftarrow 4X_3;\ Y_3 \leftarrow 4Y_3;\ X_4 \leftarrow 4X_4;\ T_a \leftarrow 4T_a;\ Y_4 \leftarrow 8Y_4;\ C \leftarrow 16C$
return $(X_3, Y_3), (X_4, Y_4), T_a, C$
```

Algorithm 19. (X, Y)-only co-Z conjugate-add-add with a zero value and a_2' (ZACAUzero') (register allocation)

```
Input: $(X_1, Y_1), (X_2, Y_2), T_a, C$ with $P = (X_1, Y_1, Z), Q = (X_2, Y_2, Z)$ such that $x_{P-Q} = 0$, $T_a = a_2'Z^2$ and $C = (X_1 - X_2)^2$
Output: $(X_3, Y_3), (X_4, Y_4), T_a, C$ with $R = (X_3, Y_3, Z_3)$, $S = (X_4, Y_4, Z_3)$ such that $R = 2P, S = P + Q$, $T_a = a_2'Z_3^2$ and $C = (X_3 - X_4)^2$

$T_1 \leftarrow X_1, T_2 \leftarrow Y_1, T_3 \leftarrow C, T_4 \leftarrow X_2, T_5 \leftarrow Y_2$

$T_a \leftarrow T_a \times T_3$        $\{a_2'Z^2_{P+Q}\}$
$T_6 \leftarrow T_3 \times T_4$        $\{W_2\}$
$T_3 \leftarrow T_3 \times T_1$        $\{W_1\}$
$T_1 \leftarrow T_2 - T_5$        $\{Y_1 - Y_2\}$
$T_1 \leftarrow T_1^2$        $\{D\}$
$T_1 \leftarrow T_1 - T_a$        $\{D - a_2'Z'^2\}$
$T_1 \leftarrow T_1 - T_3$        $\{D - a_2'Z'^2 - W_1\}$
$T_1 \leftarrow T_1 - T_6$        $\{X_1'\}$
$T_6 \leftarrow T_6 - T_3$        $\{W_2 - W_1\}$
$T_6 \leftarrow T_6 \times T_2$        $\{-A_1\}$
$T_2 \leftarrow T_2 - T_5$        $\{Y_1 - Y_2\}$
$T_5 \leftarrow 2T_5$        $\{2Y_2\}$
$T_5 \leftarrow T_5 + T_2$        $\{Y_1 + Y_2\}$
$T_5 \leftarrow T_5 \times T_3$        $\{Y_2' + A_1\}$
$T_5 \leftarrow T_5 + T_6$        $\{Y_2'\}$
$T_3 \leftarrow T_3 - T_1$        $\{W_1 - X_1'\}$
$T_2 \leftarrow T_2 \times T_3$        $\{Y_1' + A_1\}$
$T_2 \leftarrow T_2 + T_6$        $\{Y_1'\}$
$T_3 \leftarrow T_1^2$        $\{C'\}$
$T_a \leftarrow T_a \times T_3$        $\{a_2'Z_R^2\}$
$T_4 \leftarrow T_1 \times T_3$        $\{X_4\}$
$T_3 \leftarrow T_2 - T_5$        $\{Y_1' - Y_2'\}$
$T_5 \leftarrow T_2 \times T_4$        $\{Y_4\}$
$T_2 \leftarrow T_3^2$        $\{D'\}$
$T_1 \leftarrow T_2 - T_a$        $\{D' - a_2'Z'^2\}$
$T_1 \leftarrow T_1 - T_4$        $\{X_3\}$
$T_6 \leftarrow T_1 - T_4$        $\{X_3 - X_4\}$
$T_3 \leftarrow T_3 - T_6$        $\{Y_1' - Y_2' + X_4 - X_3\}$
$T_3 \leftarrow T_3^2$        $\{(Y_1' - Y_2' + X_4 - X_3)^2\}$
$T_2 \leftarrow T_3 - T_2$        $\{(Y_1' - Y_2' + X_4 - X_3)^2 - D'\}$
$T_3 \leftarrow T_6^2$        $\{C\}$
$T_2 \leftarrow T_2 - T_3$        $\{(Y_1' - Y_2' + X_4 - X_3)^2 - D' - C\}$
$T_5 \leftarrow 2T_5$        $\{2Y_4\}$
$T_2 \leftarrow T_2 - T_5$        $\{Y_3\}$
$T_1 \leftarrow 4T_1$        $\{4X_3\}$
$T_2 \leftarrow 4T_2$        $\{4Y_3\}$
$T_3 \leftarrow 16T_3$        $\{16C\}$
$T_4 \leftarrow 4T_4$        $\{4X_4\}$
$T_5 \leftarrow 4T_5$        $\{8Y_3\}$
$T_a \leftarrow 4T_a$        $\{4a_2'Z_3^2\}$

return $((T_1, T_2), (T_4, T_5), T_a, T_3)$
```

Efficient Removal of Random Delays from Embedded Software Implementations Using Hidden Markov Models

François Durvaux, Mathieu Renauld, François-Xavier Standaert*, Loic van Oldeneel tot Oldenzeel, and Nicolas Veyrat-Charvillon

Université catholique de Louvain, ICTEAM/ELEN/Crypto Group,
B-1348 Louvain-la-Neuve, Belgium

Abstract. Inserting random delays in cryptographic implementations is often used as a countermeasure against side-channel attacks. Most previous works on the topic focus on improving the statistical distribution of these delays. For example, efficient random delay generation algorithms have been proposed at CHES 2009/2010. These solutions increase security against attacks that solve the lack of synchronization between different leakage traces by integrating them. In this paper, we demonstrate that integration may not be the best tool to evaluate random delay insertions. For this purpose, we first describe different attacks exploiting pattern-recognition techniques and Hidden Markov Models. Using these tools and as a case study, we perform successful key recoveries against an implementation of the CHES 2009/2010 proposal in an Atmel microcontroller, with the same data complexity as against an unprotected implementation of the AES Rijndael. In other words, we completely cancel the countermeasure in this case. Next, we show that our cryptanalysis tools are remarkably robust to attack improved variants of the countermeasure, e.g. with additional noise or irregular dummy operations. We also exhibit that the attacks remain applicable in a non-profiled adversarial scenario. These results suggest that the use of random delays may not be effective for protecting small embedded devices against side-channel leakage. They highlight the strength of Viterbi decoding against such time-randomization countermeasures, in particular when combined with a precise description of the target implementations, using large lattices.

1 Introduction

Protecting small embedded devices against side-channel attacks is a challenging task. Following the DPA book [17], masking and hiding are two popular solutions to achieve this goal. Masking can be viewed as a type of data randomization technique, in which the sensitive (key dependent) intermediate values in an implementation are split into different shares. One of its important outcomes is that, under certain physical assumptions (e.g. that the leakage of the different

* Associate researcher of the Belgian fund for scientific research (FNRS-F.R.S.).

S. Mangard (Ed.): CARDIS 2012, LNCS 7771, pp. 123–140, 2013.

shares can be considered as independent), the security of a masked implementation against side-channel attacks increases exponentially with the number of shares [3]. On the drawbacks side, masking usually implies significant performance overheads. In addition, the exponential security increase it theoretically guarantees is only effective when the amount of noise in the measurements is sufficient [24]. Hence, it is hardly useful as a standalone countermeasure for small cryptographic devices, and is usually combined with hiding. Roughly speaking, hiding aims at reducing the side-channel information by adding randomness to the leakage signal (rather than to the data producing it) and can take advantage of different methods. For example, the direct addition of physical noise, or the design of dual-rail logic styles [26], are frequently considered options. Exploiting time-randomization is another alternative, e.g. used to protect smart cards.

Among the different time-randomization techniques proposed in the literature, e.g. [5,14,27], one can generally distinguish the software ones, e.g. based on Random Delay Interrupts (RDIs), from the hardware ones, e.g. based on increasing the clock jitter. Usually, the more hardware-flavored is the countermeasure, the more signal-processing oriented are the solutions to overcome them [11,19,28]. In this paper, we pay a particular attention to the software-based solutions exploiting RDIs. In this setting, it is interesting to notice that most previous evaluations of the countermeasures' impact (e.g. [16]) pre-process the leakage traces by integrating them. Somewhat influenced by this evaluation technique, recent works such as the ones of Coron and Kizhvatov at CHES 2009/2010 [6,7] mainly focused on how to improve the statistical distribution of the random delays, in order for their integration to produce the most noisy traces. However, looking at the source codes provided in these papers that alternate actual cipher computations with dummy operations, a natural question is to ask whether techniques based on pattern-recognition could not be used to directly remove the delays. In other words, could it happen that, at least in certain contexts, this countermeasure can be strongly mitigated, or even completely reversed?

We answer this question positively and show that, when implemented in an Atmel 8-bit microcontroller, designs protected with the CHES 2009/2010 countermeasures can be as easy to attack as unprotected ones. We start by observing that simple tools based on correlation analysis can be used to detect different types of patterns in leakage traces. For example, one can identify the structure of random delays, or block cipher operations such as AddRoundKey, SubBytes or MixColumn in the AES, opening the door to various attacks. On the one hand, this suggests that omitting the possibility of random delay removal (e.g. by only considering attacks that integrate the leakage traces) may not lead to an adequate estimation of the security. On the other hand, heuristic tools based on correlation analysis are inherently limited in more complex situations, where the dummy operations are less regular than in [6,7], or when the random delays exploit the hardware interrupt feature of the underlying microcontroller. For this purpose, we propose to take advantage of Hidden Markov Model (HMM) cryptanalysis, as a generic modeling tool to capture these variants of the RDI countermeasure. As previously observed in [10,15], HMMs provide a very natural tool to deal with implementations in which some operations are randomized. We show experimentally that by adequately modeling a protected AES implementation as a HMM, we are able to produce traces that exactly correspond to the ones of an unprotected implementation, with very high probability. As it remains that the addition of RDIs prevents the use of averaging to improve the quality of an adversary's measurements, we additionally evaluate the amount of noise that should be added to our measurements, in order for the countermeasure to become effective (i.e. to get closer to the security increases predicted using integration of the traces). It turns out the application of HMMs is remarkably robust to noise addition. In particular, and compared to previous works, we show that using a complete lattice of 6000 states to describe our target AES implementation allows a very resilient decoding of the random delays. We then conclude the paper by discussing possible improvements of the countermeasure and their limitations, as well as a non-profiled variant of our HMM-based cryptanalysis.

Related Work. In a recent and independent work, Strobel and Paar investigated the use of pattern matching for removing random delays in embedded software. Their proposal can be viewed as an alternative to our correlation-based techniques in Section 3. Namely, the work in [25] uses pattern matching in order to detect each random delay *independently* and exploits *hard information* made of a string of Hamming weights obtained from power measurements. By contrast, our method in Section 4 models the complete assembly code of a protected AES implementation as a HMM (i.e. considers all the delays *jointly*) and exploits *probabilistic information* from the power traces. As a result, we obtain a better robustness to noise and a more objective evaluation tool. In this respect, we finally note that compared to this previous work, the amount of noise in our measurements can be such that the direct identification of the operations fails with high probability. In other words, it is the Viterbi decoding that allows dealing with scenarios where Simple Power Analysis (SPA) attacks are unsuccessful.

2 Background: The CHES 2009/2010 Countermeasure

Overall, adding random delays in an implementation can be viewed as a tradeoff between the performance overheads (measured in code size and cycle count) and the variance added to the position of a target operation in side-channel measurement traces. In this section, we describe the countermeasure introduced at CHES 2009 (and improved at CHES 2010) by Coron and Kizhvatov, highlight their important characteristics and present our implementation. Note that these references are used for illustration, as they correspond to the state-of-the-art in RDI. However, the techniques we introduce would apply similarly to other variants of such time-randomizations, as will be confirmed in Section 4.5.

Summarizing, both proposals focus on finding a good statistical distribution for the (random) lengths of the delays. First, the CHES 2009 paper analyzes the so-called floating-mean method. Its goal is to decrease the average total length of random delays while increasing the variance they imply for the position of side-channel attack target samples. Next, in the CHES 2010 paper, the authors remark that a bad choice of parameters for the floating-mean method can lead to security weaknesses. As a result, they proposed an improved solution, together with a new criterion to evaluate the security of RDI. In both cases, their implementations ran on an 8-bit Atmel AVR platform, similar to the one we consider in this paper. In practice, the random delays were inserted at 10 places per AES round: once before `AddRoundKey`, once before each 32-bit `SubBytes` block, once before each 32-bit `MixColumn` block and once after the last `MixColumn` block. Our implementation of the RDI countermeasure followed the same guidelines as in [6,7] and was based on the AES-128 "furious" design, available as open source in [1]. Note that, as our goal is to identify and remove the delays from the traces, the actual distribution of their lengths has no incidence on our results. In other words, our focus is on *how* the delays are inserted in the normal flow of the AES instructions, not on *how much* delay is inserted at each step.

Algorithm 1. Random delay insertion function

```
randomdelay:
    rcall randombyte                                3 cycles
    tst RND                                          1 cycle
    breq zero                             1 cycle (2 if true)
    nop                                              1 cycle
    nop                                              1 cycle
dummyloop:
    dec RND                                          1 cycle
    brne dummyloop                     2 cycles (1 if false)
zero:
    ret                                             4 cycles
randombyte:
    ld RND, X+                                      2 cycles
    ret                                             4 cycles
```

More precisely, the code we used in our experimental evaluations is described in Algorithm 1. It is essentially the same as the one presented in [6], with the simplified `randombyte` function that only fetches some random numbers from a register, and can be read as follows. Whenever a random delay needs to be inserted, the `randomdelay` function is called. This function in turn calls (`rcall`) the `randombyte` function that provides a value `RND` (that has to be carefully chosen in order to get good statistical distribution for the delay lengths). Depending on the value of `RND`, there are two possible cases: either $\texttt{RND} = 0$ and the function directly terminates by calling `zero` and returning (`ret`) to the normal flow of the AES instructions; or $\texttt{RND} \neq 0$ and we enter the `dummyloop`. This loop simply consists in decrementing (`dec`) `RND` until it reaches 0: the function terminates afterwards. The right part of Algorithm 1 indicates the number of clock cycles required by the different operations in our Atmel device. Summarizing, and not considering the case where `RND` starts at 0, each delay is essentially constituted of a header of 16 cycles, $(\texttt{RND} - 1)$ dummy loops of 3 cycles, a final dummy loop of 2 cycles, and at last a return (`ret`) instruction of 4 cycles.

3 Pinpointing Useful Operation Leakages

The previous section described the RDI countermeasure and the source code that we ran in an Atmel microcontroller. In this section, we show that the different operations in this target device produce significantly different leakages that can be detected with simple tools based on correlation analysis. Beforehand, we briefly describe the setup we used to perform our experiments.

3.1 Measurement Setup and Pre-processing

Our target device is an Atmel ATMega644P. Its power consumption has been measured at maximum clock frequency (i.e. 20MHz) and taken over a shunt resistor inserted in the supply circuit of the microcontroller. Sampled data was acquired with a Tektronix TDS7k oscilloscope. In order to facilitate our attack, we applied a simple pre-processing step. Namely, we first split the traces into consecutive clock cycles, using the Fast Fourier Transform to recover the rising edges of the clock signal. This was achieved by filtering the frequency spectrum around the clock frequency and its harmonics, then applying the inverse transform on the filtered signal. This pre-processing provides a sequence of peaks indicating where the rising edges of the clock signal are. Following, we were able to work on a sequence of clock cycles instead of raw side-channel traces. It reduced both the difficulty of the attack and its computational cost.

3.2 Correlation-Based Attacks

Let us first have a look at the power traces of an AES implementation protected with the RDI countermeasure. As illustrated in Figure 1, a simple visual inspection allows determining the different parts of the code. In other words, there are significant operation leakages that can be detected with SPA in this case. Two main approaches can be considered for this purpose. On the one hand, one can target the dummy operations, i.e. find the clock cycles during which the `dummyloop` is executed. For example, random delays present a very distinctive

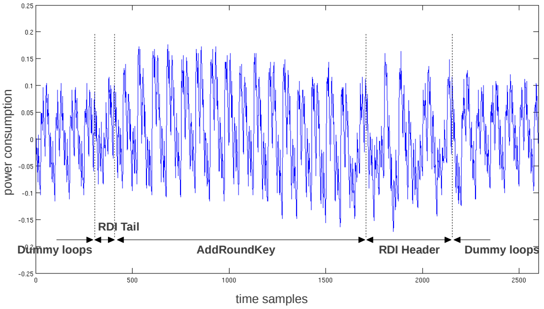


Fig. 1. AddRoundKey operation protected with the random delay countermeasure

outlook, since they contain short repetitive patterns, each loop lasting only three clock cycles. As a result, given that one can extract the pattern of this loop (including its header or tail), it is possible to match this pattern to clock cycles in the side-channel traces, by computing the cross-correlation between them. Eventually, the adversary can filter the traces by removing any cycle which is highly correlated with the delay pattern and attack the filtered traces, e.g. with Correlation Power Analysis (CPA) [2]. On the other hand, the adversary can also target the actual AES operations, instead of the delays. Indeed, while the inserted delays are of variable length and their shape can be changed, the AES operations are fixed and *have* to be executed by the program. It turns out that this detection is specially easy when large sequences of operations are executed without dummy operations. For example, AddRoundKey is rather distinctive in the proposal of [6,7], as it consists in sixteen three-cycle loops surrounded by RDI delays. Hence, if the part of the power trace that corresponds to this sequence of operations can be extracted in a preliminary profiling, it can be compared with other side-channel traces using cross-correlation, just as for the detection of dummy operations. An exemplary result of this detection technique is given in Figure 2. It highlights that significant information on the executed operations is available in our measurements, and that integrating traces is not the best approach for analyzing RDIs when such an information is available.

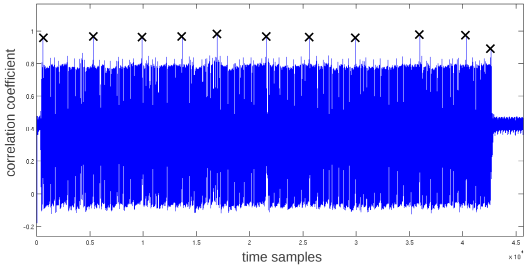


Fig. 2. Cross-correlation between an AddRoundKey pattern and one protected trace

These preliminary results are worth a few general words of discussion.

1. On improving the countermeasure. In the first place, the previous figures admittedly target the direct application of the CHES 2009/2010 countermeasures. However, while the choice of dummy operations to execute has little or no impact on "integrating" attacks, it is critical when playing with pattern matching as we undertake in this paper. In particular, two simple improvements could be implemented. First, one could use AES operations in the dummy cycles. Second, the hardware interrupts of the target microcontroller could be exploited, in order for the RDIs to occur at less predictable places (e.g. the guarantee that AddRoundKey is executed as a single block would vanish in this case).

2. On the heuristic nature of the correlation-based approach. Second, it is worth emphasizing that the previous exploitation of cross-correlation is essentially heuristic. While it is intuitively useful to put forward a risk of attacks, it is also limited and hardly generic. In particular, if the aforementioned improvements were implemented, cross-correlation-based attacks would become ineffective. This justifies the introduction of HMM cryptanalysis in the next section. As will be discussed in Section 4.5, it allows exploiting the leakages of improved countermeasures, for which correlation-based attacks become ineffective.

3. On the meaning of Kerckhoffs' principles in implementation attacks. Eventually, the analysis of RDIs raises the question of what the adversary exactly knows about his target implementation. In general, cryptographers like to consider that most potential information (e.g. about the algorithms) is public and that security only relies on the secrecy of a key. Straightforwardly translating this principle in the physical world would imply that source codes are given to the adversary, a condition that may not always be found in the field though. Such a question directly relates to the question of profiled vs. non-profiled attacks as well. For example, in the previous discussion of correlation-based attacks, is it realistic that the adversary can build an approximate pattern for the dummy operations or AES operations? In the following, we will first investigate the case where the answer is yes and justify this choice with three main reasons.

1. In practice, the gap between profiled and non-profiled attacks and the lack of knowledge about the underlying hardware and implementation can usually be overcome in the long term. Examples of solutions to reduce this gap include the use of non-profiled stochastic models [9,21], or techniques inspired from side-channel attack reverse-engineering, e.g. discussed in [8,12,20].
2. In general in cryptography, security evaluation are looking for worst cases, and this also applies to implementation attacks [23]. Hence, regardless the practical relevance of certain adversarial scenarios, it is essential to consider them as they provide a bound on what an actual adversary could achieve, and a fair understanding of the security level provided by cryptographic implementations. Security against specific attacks can of course be higher than what is lower-bounded by worst-case evaluations.
3. Sound countermeasures should provide additional security even in the worst cases. For example, if an adversary is given a masked implementation with an accurate description of its design and source code (including the exact time instants when the shares are manipulated), the analysis of Chari et al. [3] still holds and the data complexity of an attack against this implementation does still increase exponentially with the amount of shares.

Note that as an illustration of the first point (i.e. the sometimes small gap between profiled and non-profiled attacks), we additionally provide a non-profiled version of our proposed HMM cryptanalysis in Section 4.6.

4 Hidden Markov Model Cryptanalysis

Having justified the need of optimal evaluation tools, this section investigates a new cryptanalysis of RDIs based on HMMs. We argue that it constitutes an interesting generic tool to capture our problem. In particular, and contrary to the correlation-based techniques, it can easily deal with various types of dummy operations and interrupt processes. As mentioned in introduction, this approach follows previous works in the field of reverse-engineering and cryptanalysis of randomized implementations, where similar principles have been used. For example, the work of Karlov and Wagner is close to ours as it exploits HMMs to break randomized exponentiation algorithms [15]. Yet, one important difference is the size of the lattices that we consider to represent our AES implementation: while this previous work exploits a state machine with 3 states, we build a complete lattice of 6000 states to model the protected AES, which allows a very resilient decoding. To a certain extent, removing random delays in side-channel traces can also be seen as a simplified reverse-engineering problem. That is, Eisenbarth et al. intend to build a disassembler, in order to extract an exact sequence of unknown instructions being executed by a device [10]. We follow the same goal in the case where the instructions are known, but the number of dummy loops that are executed for each delay is unknown. In the following, we first explain how to translate the RDI detection as a HMM problem. Next, we describe how to actually remove the delays from side-channel leakage traces and present results of experimental attacks. Eventually, we discuss possible improvements of the countermeasure as well as a non-profiled variant of the attack.

4.1 Building the HMM

A Markov model is a (memoryless) system with a finite number of states, for which the probabilities of transition to the next state only depend on the current state. It is thus constituted of a set of states π_i's and a transition probability matrix T. $T(i,j)$ is the (*a priori*) probability that the next state is π_j if the current state is π_i. If we denote with s_t the current state of the system at time t, $T(i,j) = \Pr(s_{t+1} = \pi_j | s_t = \pi_i)$. In the case of a Hidden Markov model, the sequence $\mathbf{s} = (s_0, s_1, ..., s_n)$ of the states occupied by the system is not known. However, the adversary has access to (at least) one observable that gives partial information about this sequence. Namely, at each time step t, a random vector $\mathbf{l}_t$ is observed by the adversary. In addition to the transition probability matrix, the HMM is then characterized by the emission probability functions associated to each state π_i, namely: $e_i(\mathbf{l}_t) = \Pr(\mathbf{l}_t | s_t = \pi_i)$. For our protected AES implementation, the Markov model describes the encryption process, with each state π_i associated to an instruction (e.g. `NOP`, `RET`, ...). Some instructions take only one clock cycle, but others require several clock cycles to be completed. As each state should correspond to the same number of clock cycles, longer instructions are split into different states, e.g. `RCALL` is split into three states associated to `RCALL` 0, `RCALL` 1 and `RCALL` 2. We call *instruction cycle* the (part of an) instruction associated with a state π_i. The list of all the 26 instruction cycles appearing in the code of the protected AES can be found in Table 1. Note that the same instruction cycle can be used at multiple places in the AES code, corresponding to different running states, e.g. $\pi_0 \leftrightarrow$ `MOV` 0, $\pi_1 \leftrightarrow$ `EOR` 0, $\pi_2 \leftrightarrow$ `MOV` 0, . . . More precisely, the protected AES code can be divided into two types of instruction sequences: the deterministic instruction sequences and the

Table 1. List of instruction cycles occurring in the protected AES implementation

Index	Cycle	Index	Cycle	Index	Cycle
0	`NOP` 0	9	`RET` 2	18	`BRNE T` 1
1	`RCALL` 0	10	`RET` 3	19	`MOV` 0
2	`RCALL` 1	11	`TST` 0	20	`EOR` 0
3	`RCALL` 2	12	`BREQ F` 0	21	`LPM` 0
4	`LDI` 0	13	`BREQ T` 0	22	`LPM` 1
5	`LD` 0	14	`BREQ T` 1	23	`LPM` 2
6	`LD` 1	15	`DEC` 0	24	`RJMP` 0
7	`RET` 0	16	`BRNE F` 0	25	`RJMP` 1
8	`RET` 1	17	`BRNE T` 0		

dummy loops. In the deterministic sequences, the instruction cycle associated to a state is fully determined by its position in the sequence. In order to model these deterministic sequences, we can simply use one different state per successive cycle in the sequence, with deterministic transition probabilities: $T(i, i{+}1) = 1$ (see the top of Figure 3). By contrast, the non-deterministic dummy loops are constituted of a branching (`BRNE`) and a decrement (`DEC`) instruction. This translates into 4 instruction cycles: `BRNE T` 0, `BRNE T` 1 and `DEC` 0 in the loop (i.e. while the counter is decremented), and eventually `BRNE F` 0 to end the loop.

There are two main ways to encode these loops in a Markov model. The simplest one consists in using 4 states, as presented in the middle part of Figure 3. The transition probabilities at the output of the `DEC` state are p to restart another loop, and $1-p$ to exit the loop. This representation uses a fixed probability p, that cannot be changed based on the number of loops previously executed. It is thus not possible to render the details of the probability distribution of the delay lengths. Another way is to "unfold" the dummy loops by using $255 \times 3 + 1$ states, as in the lower part of Figure 3 (255 being the maximum number of dummy loops in one delay). It is then possible to fine tune the probabilities p_1, p_2, ..., in order to match the probability distribution of the delay lengths as closely as possible. The first representation is only perfectly accurate to model delay lengths following a geometric distribution (which is not our case), but offers better performances due to the lower number of states. The second representation is more precise and allows modeling more accurately different distributions of random delay lengths, potentially leading to a better robustness against noisy measurements. However, its larger number of states also implies a more complex resolution phase. Experimental results described in the next section showed that the compact representation (with $p = 0.1$, chosen empirically) was sufficient to obtain successful attacks with low data complexity, even with delay lengths not following a geometric distribution. Hence, we focus on this one in the rest of the paper. Eventually, our Markov model for the protected AES implementation has approximately 6000 states, each of them associated to one of the 26 different instruction cycles given in Table 1. The corresponding transition matrix T is very sparse, as the sequence of instructions is deterministic most of the time (in which case $T(i, i + 1) = 1$ is the only non-zero value of the i^{th} line).

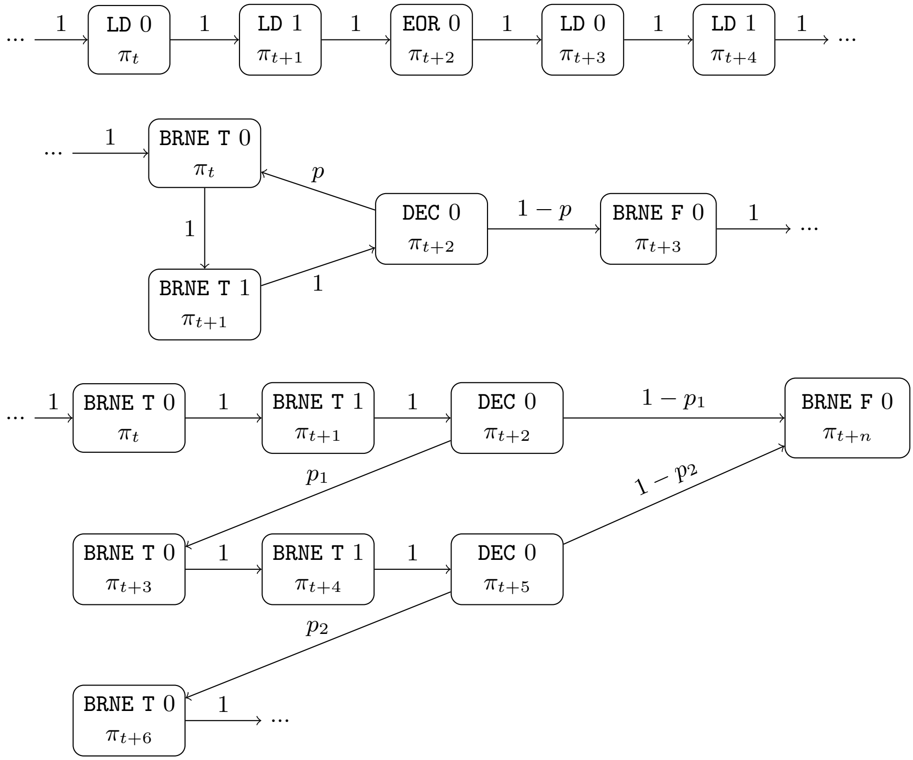


Fig. 3. Parts of the Hidden Markov model. Above: beginning of the `addRoundKey` operation (deterministic sequence). Middle and below: dummy loop.

4.2 Building the Templates

Given a Markov model for the protected AES, we still have to estimate the emission probability functions $e_i(\mathbf{l}_t) = \Pr(\mathbf{l}_t|s_t = \pi_i)$ corresponding to each state π_i. We make two assumptions for this purpose:

1. The power trace $\mathbf{l}$ can be cut cycle by cycle: $\mathbf{l} = (\mathbf{l}_0, \mathbf{l}_1, ...)$. This is possible using the FFT-based technique of Section 3.1. We denote with *trace cycle* each part $\mathbf{l}_i$ of the power trace corresponding to one clock cycle. Trace cycles are measured vectors of 25 values (obtained with the setup of Section 3.1).
2. The emission at time t, denoted as $\mathbf{l}_t$, only depends on the type of the instruction executed at time t. This assumption is admittedly not entirely accurate (because data dependencies, pipeline effects, . . . are neglected). But it again turned out to be sufficiently respected for our attacks to succeed.

Estimating the emission probability functions is equivalent to building a template for each state π_i. But contrarily to standard side-channel template attacks, where templates are built from a single instruction in order to distinguish the data being processed (e.g. [4]), we build templates for the different instruction cycles in order to tell them apart. That is, as our Makov Model contains 6000 states, we could build up to 6000 templates. But thanks to our second assumption, the same template can be used for every state π_i associated to the same instruction cycle from Table 1. Hence only 26 templates are required for the 6000 different states. Next, the emission probability functions are directly determined by these templates: for an instruction cycle i, we have the function $e_i(\mathbf{l}_t) = \mathcal{N}(\mathbf{l}_t|\hat{\boldsymbol{\mu}}_i, \hat{\boldsymbol{\Sigma}}_i)$. For each template, the mean vector $\hat{\boldsymbol{\mu}}_i$ and correlation matrix $\hat{\boldsymbol{\Sigma}}_i$ are estimated from a set of trace cycles $\mathbf{l}_t$ corresponding to the same instruction cycle i. In order to build these 26 templates, we thus need to find sets of trace cycles $\mathbf{l}_t$ corresponding to each possible instruction cycle. However, due to the unknown lengths of the delays in a protected implementation, it is impossible to directly match trace cycles to their corresponding instruction cycles. As a result, we used a *profiling phase* for this purpose. That is, we built the templates from a training device for which the length of the delays was fixed. In addition, we used this profiling phase in order to efficiently reduce the dimensionality of our traces cycles, using the Principal Component Analysis (PCA) technique described in [22]. PCA consists in linearly projecting the data (i.e. the trace cycles) on a lower dimensional subspace, in such a way that the variance between the different instruction cycles is maximized. We kept 3 dimensions per template, which appeared to be a good compromise between the amount of variability we retain and the complexity of the parameters we need to evaluate.

4.3 Removing the Delays

Given properly profiled templates, all the parameters of our HMM are fixed: the state vector π, the transition probability matrix T and the emission probability functions e_i. Hence, it only remains to identify the state sequence $\mathbf{s} = (s_0, s_1, ...)$ that is the best match for a given observation sequence $\mathbf{l} = (\mathbf{l}_0, \mathbf{l}_1, ...)$. The Viterbi algorithm can be launched for this purpose, as described with Algorithm 2 and briefly explained as follows. Let us consider a HMM with N_s states, and a sequence of N_o observations. A probability table $V \in \mathbb{R}^{N_s \times N_o}$ is first built, such that each value $V(i,j)$ is the probability of the most probable sequence of states ending in state π_i, given the sequence of observations $(\mathbf{l}_0, ..., \mathbf{l}_j)$. The first column of V is initialized with the probabilities for the different states to match the first observation $\mathbf{l}_0$: $V(i,0) = p_i \cdot e_i(\mathbf{l}_0)$ (where the p_i's are the initial a priori probabilities that the system starts in state π_i). Next, for each additional observation $\mathbf{l}_t$ in the sequence, the probabilities for the different states are determined by $V(i,t) = e_i(\mathbf{l}_t) \cdot \max_j(V(j,t-1) \cdot T(j,i))$. These probabilities take into account the emission probability of the current observation (i.e. $e_i(\mathbf{l}_t)$), but also the most probable sequences of length $t-1$ (i.e. $V(j,t-1)$), weighted by the transition probabilities $T(j,i)$'s. A matrix $I \in \mathbb{N}^{N_s \times (N_o - 1)}$ is finally used to store, for each step $t \geq 1$ and each state i, the index j of the most probable state sequence of length $t-1$ that can lead to state i: $I(i,t-1) = \operatorname{argmax}_j(V(j,t-1) \cdot T(j,i))$. Once the full sequence of observations is processed, the algorithm selects the most probable ending state, and backtracks to the first observation using the matrix I to select at each step the previous state in the most probable sequence.

Algorithm 2. Viterbi algorithm

```
input: a HMM characterized by a set of $N_s$ states $\pi_i$, initial probabilities $p_i$, a transition probability matrix $T$ and emission probability functions $e_i$.
input: a sequence of $N_o$ observations $\mathbf{l} = (\mathbf{l}_0, \mathbf{l}_1, ...)$.
output: the most probable sequence of states corresponding to the observations.
Define $V$ a new matrix in $\mathbb{R}^{N_s \times N_o}$.
Define $I$ a new matrix in $\mathbb{N}^{N_s \times (N_o - 1)}$.
//Initial probabilities.
for $i = 0$ to $i = N_s - 1$ do
    $V(i, 0) \leftarrow p_i \cdot e_i(\mathbf{l}_0)$
end for
//Computing the probabilities.
for $t = 1$ to $t = N_o - 1$ do
    for $i = 0$ to $i = N_s - 1$ do
        $V(i, t) \leftarrow e_i(\mathbf{l}_t) \cdot \max_j(V(j, t-1) \cdot T(j, i))$
        $I(i, t-1) \leftarrow \mathrm{argmax}_j(V(j, t-1) \cdot T(j, i))$
    end for
end for
//Backtracking to find the most probable path.
Define $\mathbf{s}$ a new vector of size $N_o$.
$\mathbf{s}(N_o - 1) \leftarrow \mathrm{argmax}_j(V(j, N_o))$
for $t = n_o - 2$ to $t = 0$ do
    $\mathbf{s}(t) \leftarrow I(\mathbf{s}(t+1), t)$
end for
return s
```

4.4 Results of the HMM Method and Impact of the Noise

We evaluated the efficiency of the HMM method by using it against power traces measured from a protected implementation of the AES on our 8-bit Atmel microcontroller. For comparison purposes, we also investigated the security of a non-protected implementation running on the same platform, in order to evaluate the security improvement offered by RDIs. Since we are considering first-order (standard) DPA attacks (as defined in [18]), we chose to run CPA to illustrate the insecurity of these implementations. We used a simple Hamming weight model for this purpose. As shown in Figure 4, the success rate of CPA against the protected implementation with actual noise level (corresponding to the curves with no additional simulated noise in the figure, i.e. $\sigma^* = 0$) is nearly the same as for the CPA against the unprotected implementation. Note that for each curve, we estimated the first-order success rate defined in [23], from a set of 100 independent experiments. In fact, the slight difference between protected and unprotected implementations can be explained by the postprocessing of the traces after removing the RDIs, because of slight imperfections of the cycles cut. Otherwise, the actual removing of the delays was close to 100% successful. As a result, both implementations can be broken after approximately 100 measurements. For comparison, the integration attack on the protected implementation in [6] requires approximately 45 000 power traces to succeed. The time complexity due to the Viterbi algorithm amounts to less than 10 minutes per power trace, which is quite high but still acceptable as long as the number of traces to process is not too high (note that the computing time can be optimized, *e.g.* by decoding only the beginning of the trace instead of the whole trace). We conclude that (1) the HMM method is much more efficient than the integration method to attack the RDI countermeasure, and (2) with the actual noise level of an Atmel 8-bit microcontroller, the RDI countermeasure is completely reversible.

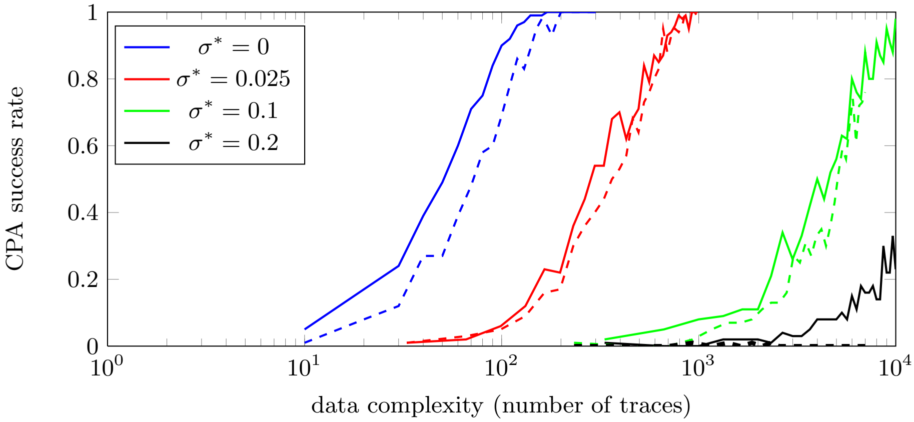


Fig. 4. Success rate of a CPA attack in different scenarios. The plain lines correspond to the attack on an unprotected implementation, the dashed lines correspond to the attack on a protected implementation with RDI removed by the HMM method. σ^* is the standard deviation of the simulated Gaussian noise added to the traces, $\sigma^* = 0$ corresponds to the real noise level. The delays are efficiently removed up to $\sigma^* = 0.1$.

In view of the lack of security improvement of our RDI-protected implementation, we additionally investigated the impact of noise on the efficiency of the HMM-based random delay removal. For this purpose, we added (before the dimensionality reduction step) a simulated random Gaussian noise with standard deviation σ^* to the (already noisy) traces, and we applied the HMM method to the resulting traces. The corresponding signal-to-noise ratios are given in Table 2: they correspond to the maximum ratio between the variance of the mean traces corresponding to the 26 possible instruction cycles, and the noise variance, where the maximum is taken over all the time samples in the traces, as defined in [16]. In addition, we also provided a Correct Classification Rate (CCR), defined as the probability that an instruction cycle among the 26 possible ones in Table 1 is correctly identified from its leakage. It essentially corresponds to the success rate of a template-based SPA performed against the instruction cycles considered independently (the uniform probability without leakage is $1/26 \approx 0.03$). As can be observed, our measurement noise can be increased such that the direct identification of the random delays thanks to an SPA becomes unlikely. By contrast, it turns out that when taking advantage of HMMs, it is still possible to efficiently remove the delays with $\sigma^* = 0.1$, which is one order of magnitude higher than the actual noise we observed in our measurements (i.e. $\sigma = 8 \times 10^{-3}$). In other words, up to high noise levels, the adversary has more trouble estimating the correlation coefficient of a CPA than removing the delays. This suggests that Viterbi decoding allows removing RDIs jointly, even in situations where they are hard to remove individually. More precisely, Figure 4 shows the success rates of unprotected and protected implementations with $\sigma^* = 0.025$, $\sigma^* = 0.1$ and $\sigma^* = 0.2$. It confirms that removing the delays is still pretty accurate at $\sigma^* = 0.1$ (the correct length is found for 95% of the delays in this case). However, when σ^* goes beyond 0.1, it becomes increasingly harder to correctly detect the delay lengths. And by $\sigma^* = 0.2$, the HMM method does not correctly identify the delay lengths anymore. In conclusion, the RDI countermeasure has an impact on security only when the noise level is high enough for a successful CPA attack against the corresponding unprotected implementation to require more than 10 000 traces. This unfortunately goes against the original goal of using RDIs to emulate noise for small embedded (e.g. 8-bit) devices.

Table 2. SNR and CCR of target implementation with additive simulated noise σ^*

	$\sigma^* = 0$	$\sigma^* = 0.025$	$\sigma^* = 0.1$	$\sigma^* = 0.2$
SNR	0.2	0.014	1.34×10^{-3}	2.44×10^{-4}
CCR	0.29	0.19	0.08	0.06

4.5 RDI Improvements

The HMM method is very efficient against the RDI countermeasure presented in [6,7], in part due to the very regular pattern of the delays. It is thus natural to think about possible methods to make the delays and actual AES computations harder to tell apart. In this section, we discuss some proposals that could be considered in order to achieve this goal. One straightforward idea is to use delays with no regular pattern, e.g. delays made of random instructions. However, implementing the RDI countermeasure in this case will still require (1) a specific deterministic header containing the instructions needed to determine the delay length, and (2) a loop ensuring that the delay stops after a given counter has reached 0. Using the hardware interrupts of the Atmel microcontrollers would not help in this respect, as dealing with these interrupts also gives rise to (4 or 5) very distinguishable cycles. In addition, even if the delays had a perfectly random pattern, the AES operations would still have to be processed, and could be identified with the Viterbi algorithm. For illustration, we ran experiments with hypothetical simulated traces, where the delays only consisted in random instructions instead of dummy loops. That is, these simulated traces do not contain any identifiable header or tail. We recall that this context is unrealistic as such headers and tails are needed to implement the countermeasure and generally provide useful information to the adversary. But even this minimum leakage could be efficiently exploited. Namely, while the Viterbi algorithm was not always able to accurately predict the instructions processed during the delays, it was still able to correctly identify the position of the AES instructions. Hence, as illustrated in

Appendix, Figure 5, such an idea is not sufficient to prevent successful attacks. Yet, one can notice that it reduces the noise robustness of the HMM cryptanalysis. Another proposal is to use delays that look like the surrounding AES instructions. But this solution again faces the issue that the headers and tails are still distinguishable, leading to similarly efficient attacks. Besides, an extreme solution is to duplicate every AES computation n times, with one execution on real data and the rest on dummy data. This way, the HMM method provides no information on where are the delays, as the delays are totally indistinguishable from the AES computations. Unfortunately, this countermeasure provides little security at high cost: instead of attacking one time sample, the adversary will have to integrate over n. Eventually, we conjecture that solutions to improve the time-randomization of cryptographic implementations should combine RDIs with other ideas, e.g. shuffling [13] or clock jitter. We leave the analysis of these combined scenarios as an interesting scope for further research.

4.6 A Non-profiled Version of the HMM Method

The previous sections assumed that the adversary can perform a profiling phase on a test device. Although justified in a security evaluation context, this assumption may not always be verified in practice. In the following, we finally show that even in a non-profiled context, it is possible to perform efficient HMM-based attacks against an RDI-protected implementation, by exploiting "on-the-fly" characterization of the target device, with a single leakage trace. The key observation is that the encryption process cannot start with random instructions. Even if a delay is inserted at the beginning of the code, there are always some deterministic instructions in the delay header, where the length of the delay is determined. In our case, the AES encryption starts with 40 deterministic cycles before the first dummy loops take place. These deterministic cycles do not include all of the 26 instruction cycles for which we need a template, but only the first 12 of them (in Table 1). Moreover, these 40 cycles are not enough to estimate very accurate templates for all these 12 instruction cycles. Nevertheless, we can build 12 templates from these first 40 cycles and artificially increase their (noise) variance, as we know that they are not perfectly accurate. For the remaining 14 templates, we use a unique default template, built from the average of the cycles after the first 40. As a result, we have 26 emission probability functions e_i that we can plug into a first HMM: HMM_0. Using the Viterbi algorithm on the leakage trace $\mathbf{l}$ with the model HMM_0 gives the most probable sequence of states, according to our (admittedly bad estimations of the) emission probability functions. For the actual noise level we measured experimentally, between 50% and 70% of the observations are associated to the correct instruction cycle after this first iteration of the Viterbi algorithm. Next, from this predicted sequence of states, we can estimate new, more accurate templates (and emission probability functions) for the 26 instruction cycles. Plugging these new templates into a second HMM (i.e. HMM_1), we can process the same power trace $\mathbf{l}$ again, hence obtaining a better classification of the trace cyles. By iterating this procedure at most 10 times, we get a correct estimation of the templates and a perfect identification of the delay lengths. It is thus possible to perform the profiling phase “on-the-fly”, by using iterations of the Viterbi algorithm on the same power trace, provided that we have some information to start with (e.g. the 12 not so accurate templates in our experiment). As illustrated in Appendix, Figure 5, this non-profiled version of the attack is robust to noise addition, up to quite low SNRs.

5 Conclusions

The investigations in this paper confirm that protections against side-channel attacks based on time-randomizations are challenging to evaluate, as they may easily provide a false sense of security, whenever they are reversible with the appropriate signal processing / statistical / modeling tools. The case of RDIs is a good illustration of this concern: we show that their implementation in a low-cost microcontroller can be completely reversed and that making them effective is a non-trivial task (e.g. software-based improvements are unlikely to be sufficient). Hence, the study of other time-randomization techniques such as shuffling [13], or the combination of RDIs with other hiding countermeasures against side-channel attacks, are interesting research problems. Besides, RDIs are also considered as a solution to prevent fault attacks. In this setting, it is an interesting open question to determine whether the HMM-based cryptanalysis could be applied “on-the-fly” during a fault attack, in order to help adversaries to insert faults precisely, e.g. in the sensitive computations of randomized implementations.

Acknowledgements. F. Durvaux is a PhD student funded by the Walloon region MIPSs project. M. Renauld is a PhD student funded by the Walloon region SCEPTIC project. F.-X. Standaert is an associate researcher of the Belgian fund for scientific research (FNRS-F.R.S.). L. van Oldeneel is a PhD student funded by the 7th framework European project TAMPRES. N. Veyrat-Charvillon is a postdoctoral researcher funded by the ERC project 280141 (acronym CRASH).

References

1. `http://point-at-infinity.org/avraes/`
2. Brier, E., Clavier, C., Olivier, F.: Correlation Power Analysis with a Leakage Model. In: Joye, M., Quisquater, J.-J. (eds.) CHES 2004. LNCS, vol. 3156, pp. 16–29. Springer, Heidelberg (2004)
3. Chari, S., Jutla, C.S., Rao, J.R., Rohatgi, P.: Towards Sound Approaches to Counteract Power-Analysis Attacks. In: Wiener, M. (ed.) CRYPTO 1999. LNCS, vol. 1666, pp. 398–412. Springer, Heidelberg (1999)
4. Chari, S., Rao, J.R., Rohatgi, P.: Template Attacks. In: Kaliski Jr., B.S., Koç, Ç.K., Paar, C. (eds.) CHES 2002. LNCS, vol. 2523, pp. 13–28. Springer, Heidelberg (2003)
5. Clavier, C., Coron, J.-S., Dabbous, N.: Differential Power Analysis in the Presence of Hardware Countermeasures. In: Koç, Ç.K., Paar, C. (eds.) CHES 2000. LNCS, vol. 1965, pp. 252–263. Springer, Heidelberg (2000)

6. Coron, J.-S., Kizhvatov, I.: An Efficient Method for Random Delay Generation in Embedded Software. In: Clavier, C., Gaj, K. (eds.) CHES 2009. LNCS, vol. 5747, pp. 156–170. Springer, Heidelberg (2009)
7. Coron, J.-S., Kizhvatov, I.: Analysis and Improvement of the Random Delay Countermeasure of CHES 2009. In: Mangard, S., Standaert, F.-X. (eds.) CHES 2010. LNCS, vol. 6225, pp. 95–109. Springer, Heidelberg (2010)
8. Daudigny, R., Ledig, H., Muller, F., Valette, F.: SCARE of the DES. In: Ioannidis, J., Keromytis, A.D., Yung, M. (eds.) ACNS 2005. LNCS, vol. 3531, pp. 393–406. Springer, Heidelberg (2005)
9. Doget, J., Prouff, E., Rivain, M., Standaert, F.-X.: Univariate side channel attacks and leakage modeling. J. Cryptographic Engineering 1(2), 123–144 (2011)
10. Eisenbarth, T., Paar, C., Weghenkel, B.: Building a Side Channel Based Disassembler. Transactions on Computational Science 10, 78–99 (2010)
11. Guilley, S., Khalfallah, K., Lomne, V., Danger, J.-L.: Formal Framework for the Evaluation of Waveform Resynchronization Algorithms. In: Ardagna, C.A., Zhou, J. (eds.) WISTP 2011. LNCS, vol. 6633, pp. 100–115. Springer, Heidelberg (2011)
12. Guilley, S., Sauvage, L., Micolod, J., Réal, D., Valette, F.: Defeating Any Secret Cryptography with SCARE Attacks. In: Abdalla, M., Barreto, P.S.L.M. (eds.) LATINCRYPT 2010. LNCS, vol. 6212, pp. 273–293. Springer, Heidelberg (2010)
13. Herbst, C., Oswald, E., Mangard, S.: An AES Smart Card Implementation Resistant to Power Analysis Attacks. In: Zhou, J., Yung, M., Bao, F. (eds.) ACNS 2006. LNCS, vol. 3989, pp. 239–252. Springer, Heidelberg (2006)
14. Irwin, J., Page, D., Smart, N.P.: Instruction Stream Mutation for Non-Deterministic Processors. In: ASAP, pp. 286–295. IEEE (2002)
15. Karlof, C., Wagner, D.: Hidden Markov Model Cryptoanalysis. In: Walter, et al. (eds.) [29], pp. 17–34
16. Mangard, S.: Hardware Countermeasures against DPA – A Statistical Analysis of Their Effectiveness. In: Okamoto, T. (ed.) CT-RSA 2004. LNCS, vol. 2964, pp. 222–235. Springer, Heidelberg (2004)
17. Mangard, S., Oswald, E., Popp, T.: Power analysis attacks - revealing the secrets of smart cards. Springer (2007)
18. Mangard, S., Oswald, E., Standaert, F.-X.: One for All - All for One: Unifying Standard DPA Attacks. IET Information Security 5(2), 100–110 (2011)
19. Nagashima, S., Homma, N., Imai, Y., Aoki, T., Satoh, A.: DPA Using Phase-Based Waveform Matching against Random-Delay Countermeasure. In: ISCAS, pp. 1807–1810. IEEE (2007)
20. Réal, D., Dubois, V., Guilloux, A.-M., Valette, F., Drissi, M.: SCARE of an Unknown Hardware Feistel Implementation. In: Grimaud, G., Standaert, F.-X. (eds.) CARDIS 2008. LNCS, vol. 5189, pp. 218–227. Springer, Heidelberg (2008)
21. Schindler, W., Lemke, K., Paar, C.: A Stochastic Model for Differential Side Channel Cryptanalysis. In: Rao, J.R., Sunar, B. (eds.) CHES 2005. LNCS, vol. 3659, pp. 30–46. Springer, Heidelberg (2005)
22. Standaert, F.-X., Archambeau, C.: Using Subspace-Based Template Attacks to Compare and Combine Power and Electromagnetic Information Leakages. In: Oswald, E., Rohatgi, P. (eds.) CHES 2008. LNCS, vol. 5154, pp. 411–425. Springer, Heidelberg (2008)
23. Standaert, F.-X., Malkin, T.G., Yung, M.: A Unified Framework for the Analysis of Side-Channel Key Recovery Attacks. In: Joux, A. (ed.) EUROCRYPT 2009. LNCS, vol. 5479, pp. 443–461. Springer, Heidelberg (2009)

24. Standaert, F.-X., Veyrat-Charvillon, N., Oswald, E., Gierlichs, B., Medwed, M., Kasper, M., Mangard, S.: The World Is Not Enough: Another Look on Second-Order DPA. In: Abe, M. (ed.) ASIACRYPT 2010. LNCS, vol. 6477, pp. 112–129. Springer, Heidelberg (2010)
25. Strobel, D., Paar, C.: An Efficient Method for Eliminating Random Delays in Power Traces of Embedded Software. In: Kim, H. (ed.) ICISC 2011. LNCS, vol. 7259, pp. 48–60. Springer, Heidelberg (2012)
26. Tiri, K., Verbauwhede, I.: Securing Encryption Algorithms against DPA at the Logic Level: Next Generation Smart Card Technology. In: Walter, et al. (eds.) [29], pp. 125–136
27. Tunstall, M., Benoit, O.: Efficient Use of Random Delays in Embedded Software. In: Sauveron, D., Markantonakis, K., Bilas, A., Quisquater, J.-J. (eds.) WISTP 2007. LNCS, vol. 4462, pp. 27–38. Springer, Heidelberg (2007)
28. van Woudenberg, J.G.J., Witteman, M.F., Bakker, B.: Improving Differential Power Analysis by Elastic Alignment. In: Kiayias, A. (ed.) CT-RSA 2011. LNCS, vol. 6558, pp. 104–119. Springer, Heidelberg (2011)
29. Walter, C.D., Koç, Ç.K., Paar, C. (eds.): CHES 2003. LNCS, vol. 2779. Springer, Heidelberg (2003)

A Additional Figures

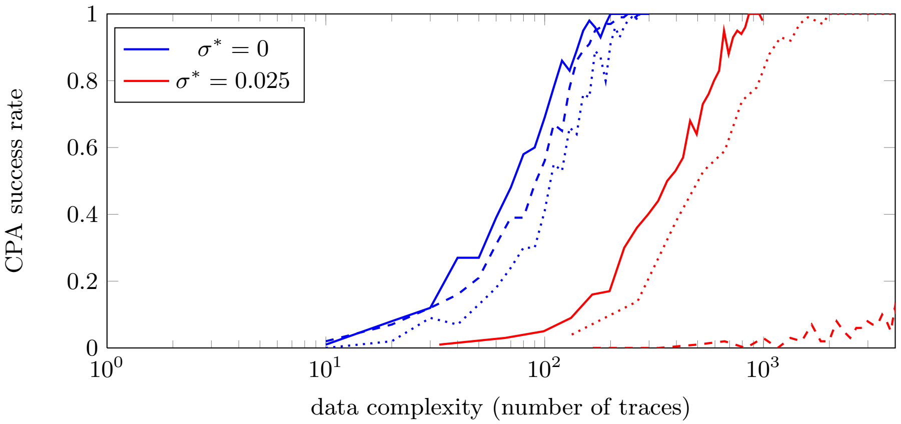


Fig. 5. Success rate of a CPA attack against traces processed using the HMM method in different scenarios. The plain lines correspond to the attack on a protected implementation with the CHES version of the RDI. The dashed lines correspond to the attack against the hypothetical (simulated) implementation of Section 4.5. The dotted lines correspond to the non-profiled version of the attack described in Section 4.6.

On the Implementation Aspects of Sponge-Based Authenticated Encryption for Pervasive Devices

Tolga Yalçın and Elif Bilge Kavun

Horst Görtz Institute for IT-Security,
Ruhr-Universität Bochum, Germany
{tolga.yalcin,elif.kavun}@rub.de

Abstract. Widespread use of pervasive devices has resulted in security problems which can not be solved by conventional algorithms and approaches. These devices are not only extremely resource-constrained, but most of them also require high performance – with respect to available resources – in terms of security, speed and latency. Especially for authenticated encryption, such performance can not be achieved with a standard encryption-hash algorithm pair or even a "block cipher mode of operation" approach. New ideas such as permutation-based authenticated encryption have to be explored. This scheme has been made possible by the introduction of sponge functions. Implementation feasibility of such an approach has yet to be explored. In this study, we make such an attempt by implementing the new SpongeWrap authenticated encryption schemes on all existing sponge functions and show that it is possible to realize a low-latency scheme in less than $6K$ gate equivalents at a throughput of 5 Gbps with a 128-bit claimed security level.

Keywords: Pervasive computing, data security, authenticated encryption, sponge functions, Keccak, Photon, Quark, Spongent.

1 Introduction

Pervasive computing is everywhere. We have, for quite sometime, got used to the idea of using all sorts of computing devices almost in every moment of our lives. These devices range from smart phones to smart cards with varying levels of computing power and usability [1]. A smart card on an ATM card may be used only as a means to draw cash from an ATM. On the other hand, a smart phone may be used to perform complex image processing in unprecedented speeds compared to the most powerful desktop computers of a decade ago. While more pervasive devices are introduced for new applications, these devices themselves also introduce more and more new application areas autonomously. However, it is not just new application areas they introduce, but unforeseen problems as well. Among these problems, security, perhaps, is the most important one, not just from an engineering point of view, but also from the user perspective.

Security of data has to be guaranteed in various levels: During computation from outside observers (also known as side-channel attackers [2]), during communication from third parties, and during storage from unauthorized users. Looking

S. Mangard (Ed.): CARDIS 2012, LNCS 7771, pp. 141–157, 2013.

at the specific example of smart cards, one may consider them to be the most pervasive computing platform. We carry several cards in our wallets for various applications such as banking, identification, access control, mobile phones, loyalty schemes [3]. All of these applications rely on personal and sometimes even critical data of the user (health information, pin codes, biometric information, etc.), which is not only a security nightmare to the user but even worse for the designers of such systems.

Data stored on such systems has to be secured via an encryption algorithm. There are several algorithms and standards designed and extensively analyzed for this purpose. The internationally accepted Advanced Encryption Standard, AES, is perhaps the most widely used encryption algorithm. It is very well-analyzed, tested, and proved to be secure – not just for today, but for the coming decades as well. Moreover, it has been implemented on countless number of platforms for almost all imaginable performance targets. While a high performance version of AES can process 55.5 Gbps of data on tens of thousands of ASIC gates [4], a lightweight version of it can fit into only about 2 thousands gates at reduced performance of a few tens of Kbps.

However, encryption alone is not sufficient. Another important operation that has to be performed on the secure data is authentication. As in the case of encryption, authentication is also very well-established and standardized as Secure Hash Algorithms, SHA-1 and SHA-2 by NIST. Competition for a third standard, SHA-3, is underway.

Authenticated encryption is a technique, which combines both authentication and encryption in order to provide confidentiality, integrity and authenticity of the data, simultaneously. While, the same functionality can be achieved via an encryption algorithm and an authentication algorithm running in parallel, it is not always the preferred solution, especially on resource-limited devices. Therefore, authenticated encryption is introduced as a block cipher mode of operation, where the same cipher block performs both functionalities. The most commonly used (also standardized) modes are CCM, CWC, OCB, EAX and GCM.

More recently, use of sponge-based hash functions as authenticated encryption primitives has been proposed [5]. With its arbitrarily long input and output sizes, the sponge construction allows building various cryptographic primitives such as a hash function, a stream cipher or a MAC [6], which, if properly combined, can lead to a "Do-It-All-Cipher". Although, sponge functions are still quite young, already a few number of sponge-based hash functions have been introduced. Among these, PHOTON, QUARK and SPONGENT are mostly targeted for lightweight applications, which KECCAK, a SHA-3 finalist, though not specifically designed so, can be tweaked to operate in lightweight mode. Furthermore, SPONGEWRAP modes of KECCAK and QUARK have also been presented to be used as authenticated encryption primitives [7]. Especially, the MONKEYDUPLEX and DONKEYSPONGE constructions show a lot of promise to be used in lightweight applications.

With its single round streaming mode, DonkeySponge construction can be effectively used in data storage applications as a low-latency cipher at the penalty of using a nonce in addition to the key, while MonkeyDuplex can perform the same functionality without nonce at the expense of more rounds per encryption. Even in that case, low-latency can be achieved by means of an unfolded design, in a similar fashion as presented in [8]. But there are open questions: From a mathematical point of view, these modes are yet to be extensively studied. The number of rounds per each sponge function in the proposed modes have to be determined together with their security claims, as done for Keccak. On the other hand, from a hardware point of view, efficiency of each sponge function in the target modes have to be determined. A new sponge-based proposal is only acceptable if it offers more (or less – in terms of gate count and power consumption) than the existing block cipher based systems.

In our study, we try to bring some answers, or more literally performance figures to the second question. We implement both DonkeySponge and MonkeyDuplex constructions on the existing sponge functions – Keccak, Photon, Quark and Spongent. In our implementations, we target low-latency data encryption, which is a realistic design target for pervasive applications, especially for data storage security. Furthermore, we choose data wordlength of 32 bits, which also is a realistic figure, considering data storage solutions on pervasive devices. We then select variants of sponge functions that can provide this data rate. These are Keccak-200, Photon-196, Quark-176 and Spongent-176, all of which provide around 80 bits of generic security with a target key length of 128 bits. Since the round numbers for MonkeyDuplex and DonkeySponge duplex mode are given only for Keccak, we obtain round numbers for other sponge functions by simple proportioning (i.e. with respect to the original proposed round numbers). In all fairness, we present the performance figures, for both these proportional round numbers and ones identical to those of Keccak.

The rest of the paper is organized as follows. In the next section, we give a brief introduction about sponge functions, which also includes specific details of each of the target sponge functions. It is followed by MonkeyDuplex and DonkeySponge constructions. In the following section, we present our implementations for both constructions together with performance figures on all hash functions. In the last section, we summarize our results and propose future directions for research.

2 Sponge Functions

Sponge functions can be used to generalize cryptographic hash functions to more general functions with arbitrary output lengths. They are based on the sponge construction, which is a repetitive construction to build a function F with variable-length input and arbitrary-length output based on a fixed-length permutation f operating on a fixed number of b bits, which is called the width. The sponge construction operates on a state of $b = r + c$ bits. r is called the bit rate and c is called the capacity. In the first step, the bits of the state are all initialized to zero. Then, the input message is padded and cut into blocks of r-bit. The construction consists of two phases, namely the absorbing phase and the squeezing phase.

- In the absorbing phase, the r-bit input message blocks are XORed with the first r-bit of the state, then interleaved with the function f. After processing all of the message blocks, the squeezing phase begins.
- In the squeezing phase, the first r-bit of the state is returned as output blocks, and then interleaved with the function f. The number of output blocks is chosen by the user. The block diagram of the sponge construction is shown in Figure 1.

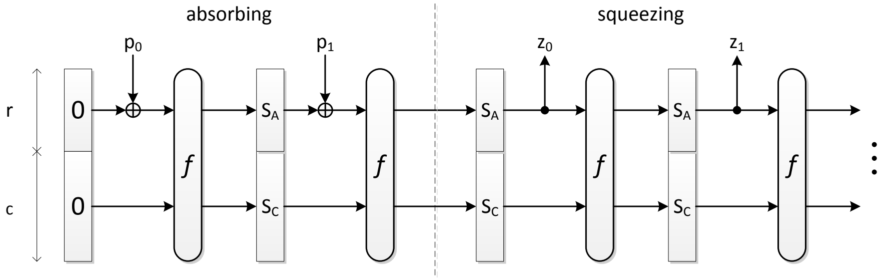


Fig. 1. Sponge construction

The existing sponge functions are summarized in the following subsections.

2.1 Keccak Sponge Function

Keccak [9] is a cryptographic hash function submitted to the NIST SHA-3 hash function competition. It is a family of hash functions based on the sponge construction that is used as a building block of a permutation from a set of seven permutations denoted by Keccak-$f[b]$, where $b \in 25, 50, 100, 200, 400, 800, 1600$ is the width of the permutation. The width b of the permutation is also the width of the state in the sponge construction. The state is organized as an array of 5×5 lanes, each of length w bits, where $w \in 1, 2, 4, 8, 16, 32, 64$ $(b = 25w)$. Depending on the selected permutation width, the Keccak-f permutation consists of a number of simple rounds with logical operations and bit permutations. The number of rounds n_r depends on the permutation width which is calculated by $n_r = 12 + 2l$, where $2^l = w$. This yields 12, 14, 16, 18, 20, 22, 24 rounds for Keccak-$f[25]$, Keccak-$f[50]$, Keccak-$f[100]$, Keccak-$f[200]$, Keccak-$f[400]$, Keccak-$f[800]$, Keccak-$f[1600]$, respectively. The Keccak-$[r, c, d]$ sponge function can be obtained by applying the sponge construction to Keccak-$f[r + c]$ with the parameters capacity c, bit rate r and diversifier d and also padding the message input specifically.

The KECCAK iterated round function is explained in Algorithm 1, where all of the operations on the indices are done in *modulo* 5. A denotes the complete permutation state array, and $A[x, y]$ denotes a particular lane in that state. $B[x, y]$, $C[x]$, $D[x]$ are intermediate variables, the constants $r[x, y]$ are the rotation offsets and $RC[i]$ are the round constants. $ROT(w, r)$ is the bitwise cyclic shift operation which moves the bit from position i into position $i + r$, in the modulo lane size.

Algorithm 1. Pseudo-code of KECCAK-f

```
KECCAK-$f[b](A)$

– $for\ i\ in\ 0 \ldots n_r - 1$
  $A = Round[b](A, RC[i])$
– $return\ A$


$Round[b](A, RC)$

– $\theta$ step:
  $C[x] = A[x, 0] \oplus A[x, 1] \oplus A[x, 2] \oplus A[x, 3] \oplus A[x, 4], \forall x\ in\ 0 \ldots 4$
  $D[x] = C[x - 1] \oplus ROT(C[x + 1], 1), \forall x\ in\ 0 \ldots 4$
  $A[x, y] = A[x, y] \oplus D[x], \forall (x, y)\ in\ (0 \ldots 4, 0 \ldots 4)$
– $\rho$ and $\pi$ steps:
  $B[y, 2x + 3y] = ROT(A[x, y], r[x, y]), \forall (x, y)\ in\ (0 \ldots 4, 0 \ldots 4)$
– $\chi$ step:
  $A[x, y] = B[x, y] \oplus ((NOT B[x + 1, y] AND B[x + 2, y], \forall (x, y)\ in\ (0 \ldots 4, 0 \ldots 4)$
– $\chi$ step:
  $A[0, 0] = A[0, 0] \oplus RC$
– $return\ A$
```

2.2 PHOTON Sponge Function

PHOTON [10] is a sponge construction with an AES-like permutation. The internal state size $t = (c + r)$ depends on the hash output size and can take five distinct values. Therefore, internal permutation P_t is defined for each internal state size. PHOTON starts with the initialization phase where the message is padded and cut into blocks of t bits. Then the t-bit state is processed by P_t permutation in absorption phase. Finally, in the squeezing phase, the n-bit hash value is returned.

The internal permutation P_t is an N_r-round transform of the t-bit state. Note that the state organization is defined according to the hash output size. However, the number of rounds is the same for all t values and the round function is iterated as the number of rounds. The round function is similar to AES round function as shown in Algorithm 2. It starts with an *AddConstants* step instead of the key addition of AES. Here, round constants and internal constants are XORed to the state. Round constants are defined for each round and the internal constants depend on the state organization. It is followed by *SubCells* and *ShiftRows* steps. In the substitution layer, the PRESENT Sbox is used in the case of 4-bit cells and the AES Sbox is used in the case of 8-bit cells. In *MixColumnsSerial* step, the final mixing layer is applied to each of the columns of the internal state. The coefficients and the irreducible polynomials used in this step again depend on the permutation type.

Algorithm 2. Pseudo-code of PHOTON

```
– for $i = 1$ to $R$ do
    $State \leftarrow AddConstant(State)$
    $State \leftarrow SubCells(State)$
    $State \leftarrow ShiftRows(State)$
    $State \leftarrow MixColumnsSerial(State)$
– end for
```

2.3 QUARK Sponge Function

QUARK [11] uses the sponge construction and a b-bit permutation P. Three different instances of QUARK are specified. Each instance is parameterized by a rate r, capacity c, and hash length n. The size of the internal state is b bits ($b = r + c$). The QUARK sponge construction processes a message in three steps: Initialization, absorption and squeezing. As in the case of KECCAK, the message is first padded and cut into r-bit blocks. These blocks are then XORed with the last r bits of the state and interleaved with permutation (P) applications. In the end, the last r bits of the state are returned as output, interleaved with permutation applications, until n bits are returned.

The permutation P is inspired by the stream cipher Grain [12] and the block cipher KATAN [13] (see Figure 2). The internal state of P is viewed as three feedback shift registers – two nonlinear and one linear. P proceeds in three stages for a given b-bit input: Initialization of the internal state, status update with f, g, and h functions, and finally the computation of the output similar to initialization. Note that functions f, g, and h are defined separately for each instance.

2.4 SPONGENT Sponge Function

SPONGENT [14] is a sponge construction based on a wide PRESENT-type [15] permutation. SPONGENT produces an n-bit hash value for a given finite number of input bits – it is a simple iterated design that takes a variable-length input and can produce an output of an arbitrary length based on a permutation π_b

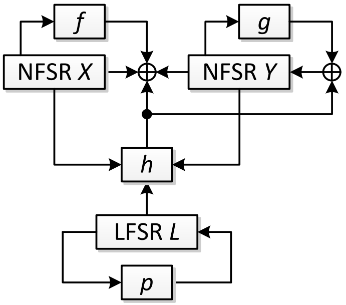


Fig. 2. Permutation of QUARK

operating on a state of b bits (where $b = r + c \geq n$, r rate and c capacity). Hashing starts with an initialization phase where the message is padded and cut into blocks of r bits. In the following phase (absorption), the r-bit message blocks are XORed with the first r bits of the state and then processed in π_b permutation. Finally, in the squeezing phase, the first r bits of the state are returned as output, interleaved with applications of π_b, until n bits are returned.

The permutation π_b is an R-round transform of the b-bit state. The round function is iterated as the number of rounds (R), which depends on the SPONGENT variant used. It is similar to the PRESENT round function, but a wider version. Also, instead of key addition, a counter value depending on an LFSR is added. The substitution and permutation layers are the same; however, they are defined for larger states. Algorithm 3 shows the π_b permutation.

Algorithm 3. Pseudo-code of SPONGENT

```
– $for\ i = 1\ to\ R\ do$
    $State \leftarrow lCounter_{reversed\ bit-order} \oplus State \oplus lCounter$
    $State \leftarrow sBoxLayer(State)$
    $State \leftarrow pLayer(State)$
– $end\ for$
```

3 Permutation-Based Authenticated Encryption

Mainstream symmetric cryptography has been dominated by block ciphers, which offer inverse function capability. However, an inverse cipher is only needed in specific modes such as electronic codebook (ECB), cipher block chaining (CBC) and offset codebook mode (OCB) in authenticated encryption. There are several modes of operation where the cipher block is used in forward (encryption) mode only. From a designer's point of view, an n-bit block cipher is nothing but a

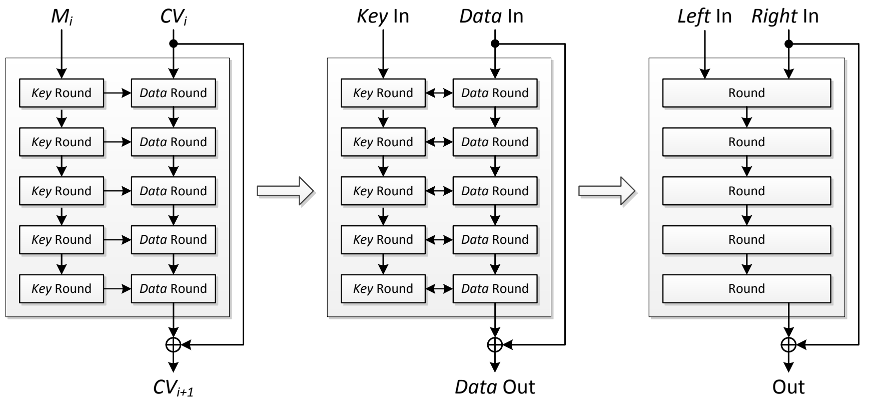


Fig. 3. Evolution from Davies-Meyer construction (left) to iterated permutation mode (right)

b-bit permutation ($b = n + |K|$, $|K|$ being the key size) with no diffusion from data state to key state. A simple example of use of a block cipher in this mode is the hashing via Davies-Meyer compression function. Since for hashing there is no need to limit diffusion, one can use a block cipher in iterated permutation mode, where the internal state is composed of both the left and right states as shown in Figure 3.

This construction vanishes the need for separate key schedule and replaces the n-bit block cipher by a b-bit permutation. This is, in fact, a block cipher without inverse, which has the capability to perform not only encryption but also message authentication – or simply, authenticated encryption (AE). In the following subsections, we will see how sponge functions can be used to perform resource-efficient and secure permutation-based authenticated encryption.

3.1 Authenticated Encryption Mode SpongeWrap

SpongeWrap [5] construction realizes authenticated encryption as shown in Figure 4. Upon initialization, key, K, is loaded into the state. Next, padded header A (also referred to as additional authentication data – AAD) is *absorbed* into the state. This is followed by the encryption (or decryption) phase, which is run in duplex mode, i.e. for each input data block (plaintext or ciphertext), an output data block (ciphertext or plaintext, respectively) is generated. The output (in case of encryption) or input (in case of decryption) ciphertext is also *absorbed* into the state, thereby running hashing in parallel with encryption. Upon completion of processing of all input data blocks, the sponge is run (with zero input data) until all the l-bit tag, T is *squeezed* from internal the state. In decryption mode, the *squeezed* tag is compared with the received tag in order to check if the received tag is valid. In SpongeWrap mode, every key, header and plain/cipher-text block is extended with a so-called *frame bit.*

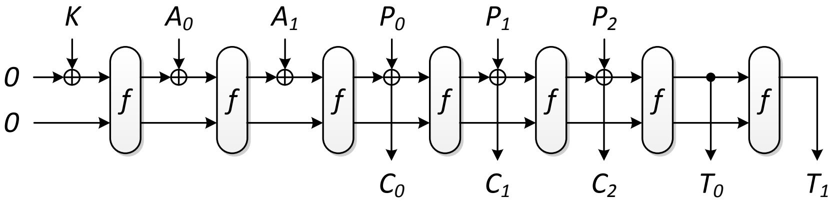


Fig. 4. SpongeWrap authenticated encryption

It is proven that the sponge and duplex constructions are secure against generic attacks with complexity below $2^{c/2}$ [16]. However, when a sponge function or duplex object is used in conjunction with a key, more refined bounds can be defined taking into account the data complexity. If the data complexity is limited to 2^a r-bit blocks, the keyed mode withstands generic attacks with time complexity up to 2^{c-a} calls of the underlying permutation. If $a < c/2$, this results in an increase of the security strength from $c/2$ to $c-a$. It should be noted that when the memory requirements of a pervasive system are considered, a is usually limited to 32 or less, which is in perfect agreement with the requirement of $a < c/2$.

3.2 DonkeySponge Construction

DonkeySponge mode of operation [17] (shown in Figure 5) can be summarized as follows:

- The b-bit state is initialized with the key and run through the round function, f, n_{init} time, resulting in the secret state. The number of rounds, n_{init} must be chosen such that all bits of the secret state depend on the MAC key.
- The b-bit blocks of the message are XORed into the secret state, interleaved with n_{absorb}-round permutations. The number n_{absorb} must be chosen to make the success probability of generating inner collisions negligible.

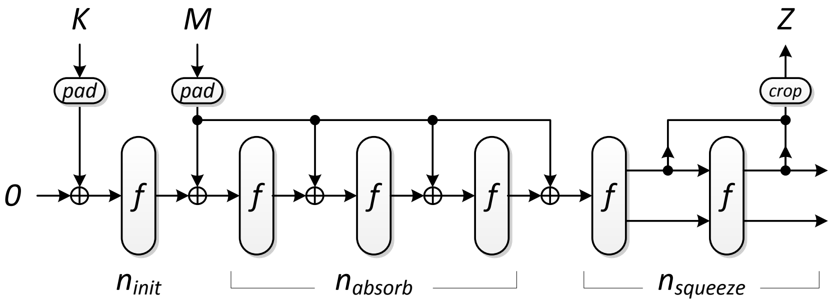


Fig. 5. DonkeySponge construction

- The tag is obtained by applying an $n_{squeeze}$-round permutation to the secret state and truncating the result to l bits. The number of rounds $n_{squeeze}$ should be high enough to prevent an adversary in reconstructing the inner state from outputs observed for chosen inputs. The number $b - l$ must be large enough to prevent state reconstruction by exhaustive search, namely, $b - l \geq k$.

In [17], $n_{init} = 3$, $n_{absorb} = 6$ and $n_{squeeze} = 12$ are chosen for Keccak-f[200]. This choice of parameters are mainly influenced by propagation experiments. Therefore, a realistic selection of these parameters for all other sponge functions can only be possible after similar experiments and/or analyses.

3.3 MonkeyDuplex Construction

MonkeyDuplex mode of operation [17] (shown in Figure 6) is a modified version of the authenticated encryption with associated data (AEAD) mode SpongeWrap based on the duplex construction in [5]. The original version can guarantee confidentiality if for the same key and different messages the associated data is unique. In other words, the associated data should behave as a nonce.

MonkeyDuplex mode removes this restriction by using a unique nonce, which makes it more fragile. However, it results in a considerable security gain. Furthermore, it allows data encryption in stream mode with $n_{duplex} = 1$, resulting in extremely high rates.

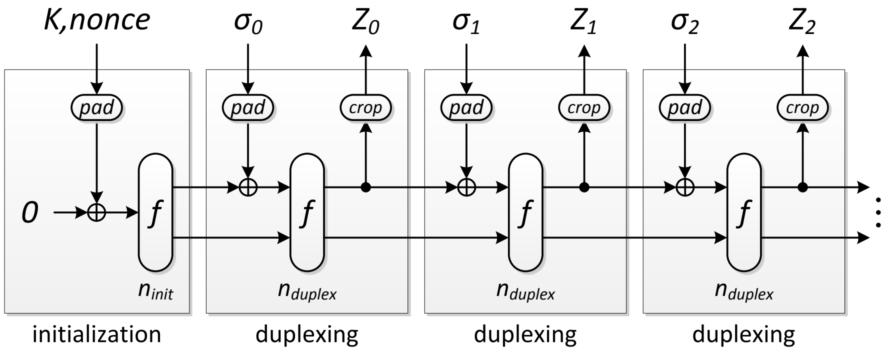


Fig. 6. MonkeyDuplex construction

4 Implementation Aspects

In this section, we summarize our implementation of the DonkeySponge and MonkeyDuplex cores on which we evaluate our performance figures. We start

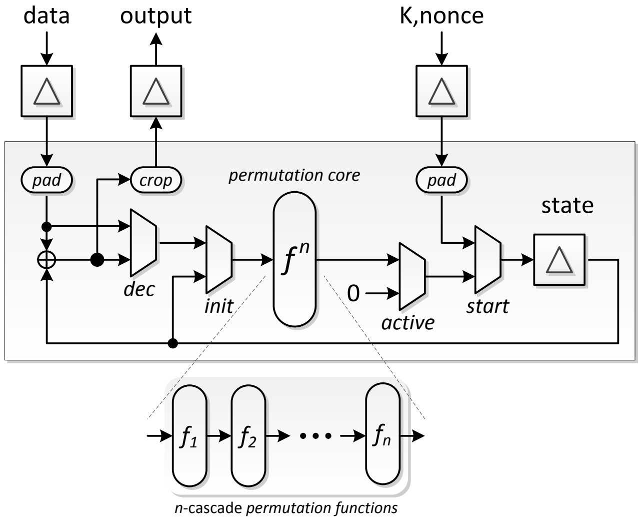


Fig. 7. DonkeySponge and MonkeyDuplex wrapper

by building a generic wrapper, into which any permutation function of a sponge, can be embedded. As shown in Figure 7, the wrapper is build up of a key register, data register, length registers (for header/AAD and data), and a state register. In agreement with our initial design target, data register is a 32-bit register, and the key register is a 192-bit register designed to collect 128-bit key and 64-bit nonce read in 32-bit blocks from input. The length registers are also 32-bit registers in order to support header and input data of up to $2^{32} \times 32$ bits each. The size of the state register depends on the state size of the chosen sponge (permutation) function. The core module is the sponge permutation function, f, module. Its input is determined depending on the phase of authenticated encryption – key/nonce absorption (initialization), header absorption, duplex, tag extraction.

The wrapper has two variants for DonkeySponge and MonkeyDuplex modes, respectively. The main differences between the two variants are the padding circuitry (which has negligible effect on the gate count) and the input data size. The DonkeySponge wrapper is designed to support input data width of up to 64 bits. However, in our designs, we run it in 32-bit mode. Both wrappers are fully functional blocks including control circuitry, and their functionalities have been verified by Modelsim SE v6.5b.

The operation of the authenticated encryption wrappers can be summarized in five phases:

- **Length/Key/Nonce Loading Phase.** Upon reception of a start command, the wrapper requests for initialization data – lengths, key, and optionally nonce – from the input interface. It then loads the corresponding registers with the received data, and instructs the internal sponge core to start operation.

- **Initialization.** This is the phase where key/nonce is hashed to generate the secret internal state of the sponge. This phase takes n_{init} rounds.
- **Header Absorption.** In this phase, the wrapper requests header data from the input interface. Each received data word is padded with and loaded into the data register, whose output is then processed by the internal sponge core (absorbed into the state). In case, the header length is zero, this phase is bypassed.
- **Data Absorption.** In this phase, the wrapper requests plaintext/ciphertext data from the input interface. Again, each received data word is padded with and loaded into the data register, whose output is then processed by the internal sponge core (absorbed into the state). Additionally the leading b bits of the new state is XORed with the input data word to generate the corresponding ciphertext/plaintext, respectively. This mode of operation is known as the duplex mode. The output word is written into the output register which flags the readiness of the output with an "output enable" flag. Processing of each word in this phase and the previous phase takes n_{absorb} rounds, which is realized in a single clock cycle (via unfolding) in order to guarantee low-latency operation.
- **Tag Extraction.** This is the final phase, where the sponge core is run with zero input data in order to generate l-bit tag (where l can be 64 to 128 bits wide). Depending on the target tag length, the internal sponge core can be run several times, squeezing 32 bits of tag at every run. In DonkeySponge wrapper, each run takes $n_{squeeze}$ rounds, where number of clock cycles is determined by the ratio of $n_{squeeze}$ to n_{absorb}. For the MonkeyDuplex, the core is run in duplex mode with zero input data; therefore $n_{squeeze} = n_{absorb}$. As in the data absorption phase, the output tag words are loaded into the output register and signaled with an "output enable" flag.

In the implementation of f-module, we refer to our initially set design targets. We want to primarily achieve low-latency encryption of 32-bit words with (about) 80-bit generic security. Therefore, the rate, r, of the sponge function should be at least $(32 + 2)$ bits (including the padding); and the capacity, c, around 160 bits, resulting in total minimum state size of 194 bits. With these parameters in mind, we choose Keccak-200, Photon-196, Quark-176 and Spongent-176 variants, with the parameters summarized in Table 1. Although not all the variants provide equal design parameters, we have to make a choice in order to limit our design space. However, since we provide the performance figures for different unfolding options, it is still possible to make realistic guesses with the chosen variants.

4.1 Performance Comparison

For the performance evaluation, we run our syntheses using Cadence RTL Synthesizer v08.10-s222 with Nangate 45 nm generic and UMC 90 nm low-leakage

Table 1. Parameters for the chosen sponge functions

	b	c	r		Security (bits)		Generic security (bits)
Function	(bits)	(bits)	(bits)	preimage	2nd preimage	collision	(keyed mode, $a = 32$)
				$(c - r)$	$(c/2)$	$(c/2)$	$(c - a)$
KECCAK-200	200	164	36	128	82	82	132
PHOTON-196	196	160	36	124	80	80	128
QUARK-176	176	160	16	144	80	80	128
SPONGENT-176	176	160	16	144	80	80	128

cell libraries. The gate counts are provided for both constrained and unconstrained syntheses, while power figures are only provided for 50 MHz constrained syntheses.

Figure 8 shows the area comparison for 50 MHz constrained (left) and unconstrained (right) DONKEYSPONGE wrapper designs, respectively. n in the figures corresponds to the number of unfolded levels of permutations within the f-block. In the case of KECCAK, this corresponds to n_{absorb}. However, for all other sponge functions, n's are chosen arbitrarily in order to present illustrative numbers. For example, in the C-QUARK proposal [7] (a QUARK based SPONGEWRAP instance), n is chosen to be 64. As a reference value, gate counts for KECCAK are 13.6 KGE and 15.4 KGE for 90 nm and 45 nm libraries, respectively.

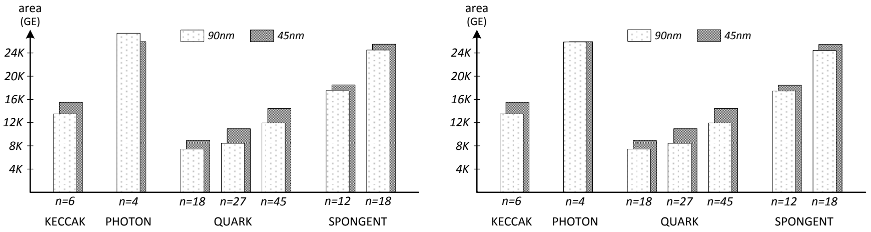


Fig. 8. Area comparison of DONKEYSPONGE wrappers for 50 MHz constrained (left) and unconstrained (right) cases

An important observation we can make from the figure is how close the gate counts are for constrained and unconstrained cases. This means timing targets are met even in the presence of several unfolded rounds, which is a major advantage of permutation based encryption schemes over conventional block ciphers.

Figure 9 shows the power comparison for 50 MHz constrained wrapper designs for the same values of n as in the area comparison. Power figures are provided only for 90 nm library. It should also be noted that these are synthesized power figures, and are *not* as much realistic as simulated figures. As a reference value, average power consumption for KECCAK is 13.6 mW.

Figure 10 shows the area comparison for 50 MHz constrained (left) and unconstrained (right) MONKEYDUPLEX wrapper designs, respectively. Again, n in the figures corresponds to the number of unfolded levels of permutations within

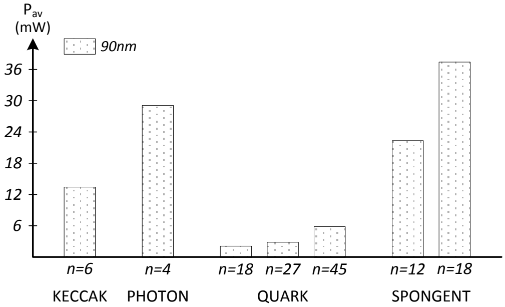


Fig. 9. Power comparison of DonkeySponge wrappers (50 MHz constrained)

the f-block. In the case of Keccak, this corresponds to $n_{duplex} = 1$. However, for all other sponge functions, n's are chosen arbitrarily in order to present illustrative numbers. As a reference value, gate counts for Keccak are 5.9 KGE and 7.4 KGE for 90 nm and 45 nm libraries (constrained), respectively.

In the case of MonkeyDuplex wrappers, Quark loses its area advantage coming from its simple internal structure, since registers dominate the overall area.

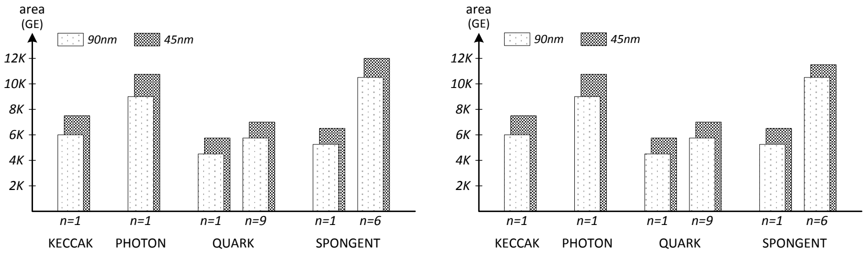


Fig. 10. Area comparison of MonkeyDuplex wrappers for 50 MHz constrained (left) and unconstrained (right) cases

Figure 11 shows the power comparison for 50 MHz constrained wrapper designs for the same values of n as in the area comparison. Again, power figures are provided only for 90 nm library, and they are synthesized power figures. As a reference value, average power consumption for Keccak is 2.1 mW.

We furthermore present the gate counts corresponding to different values of n for each design in separate graphs in Figure 12. For our synthesis ranges, gate counts do not vary in the existence of timing constraint for 45 nm designs, while it is not the case for other designs (except Quark). The gate counts are presented for only MonkeyDuplex constructions. The differences between DonkeySponge and MonkeyDuplex areas are negligible.

We have also checked the highest clock frequencies for all the designs in 90 nm (for $n = 1$). These are 154.2, 154.1, 83.4, and 154.1 MHz for Keccak, Quark,

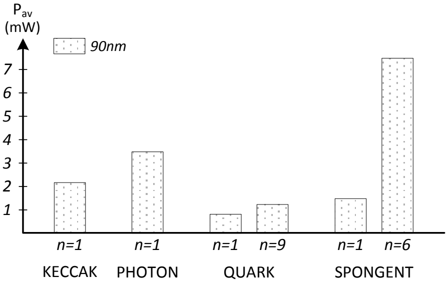


Fig. 11. Power comparison of MonkeyDuplex wrappers (50 MHz constrained)

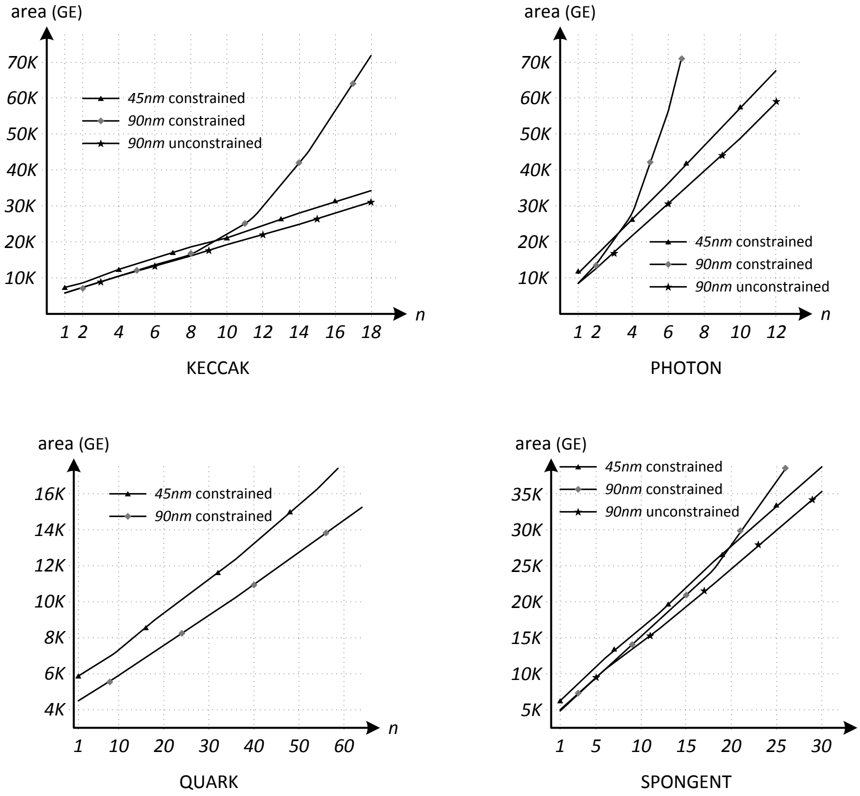


Fig. 12. Area figures for Keccak (upper left), Photon (upper right), Quark (lower left) and Spongent (lower right) MonkeyDuplex wrappers

Photon and Spongent, respectively. This means, except for Photon, sponge permutations do not determine the critical delay path. Instead, the wrapper around determines the overall speed of the design.

With a 32-bit datapath (as in the current implementations), MonkeyDuplex scheme can achieve authenticated encryption throughput of 4.9 Gbps at a gate count of 5.9 KGE, in other words almost 1 Mbps/GE. This number is much higher than any reported figure for existing block cipher based schemes.

5 Conclusion

In this study, we have implemented the newly proposed sponge-based DonkeySponge and MonkeyDuplex authenticated encryption schemes for all known sponge functions in the literature. We have demonstrated the simplicity and effectiveness of these schemes in terms of resource usage and latency, both of which are important design parameters for pervasive computing systems. Especially in data storage security, low-latency is of top priority. MonkeyDuplex scheme achieves low-latency at an incredibly high throughput of 4.9 Gbps within only 5.9 KGE area, with a 128-bit claimed security.

This is still a very new area of research. The security of these schemes have yet to be studied in deep. On the other hand, the performance results we have obtained shows the value of such research. As the next step, we will evaluate these ciphers with different technologies and parameters (using different variants of each sponge function). We will also provide simulated power figures, and compare our results with block cipher based authenticated encryption schemes, as well.

References

[1] Hansmann, U., Merk, L., Nicklous, M.S., Stober, T.: Pervasive Computing: The Mobile World. Springer (August 2003)

[2] Mangard, S., Oswald, E., Popp, T.: Power Analysis Attacks: Revealing the Secrets of Smart Cards (Advances in Information Security). Springer-Verlag New York, Inc. (2007)

[3] Allied Technique. Smart Cards (June 2012), `http://www.alliedtechnique.com/smartcards/`

[4] Soliman, M.I., Abozaid, G.Y.: FPGA Implementation and Performance Evaluation of a High Throughput Crypto Coprocessor. J. Parallel Distrib. Comput. 71(8), 1075–1084 (2011)

[5] Bertoni, G., Daemen, J., Peeters, M., Van Assche, G.: Duplexing the Sponge: Single-Pass Authenticated Encryption and Other Applications. In: Miri, A., Vaudenay, S. (eds.) SAC 2011. LNCS, vol. 7118, pp. 320–337. Springer, Heidelberg (2012)

[6] Saarinen, M.-J.O., Engels, D.W.: A Do-It-All-Cipher for RFID: Design Requirements (Extended Abstract). IACR Cryptology ePrint Archive, 2012:317 (2012)

[7] Aumasson, J.-P., Knellwolf, S., Meier, W.: Heavy Quark for secure AEAD. In: DIAC - Directions in Authenticated Ciphers, Sweden, July 5-6 (2012)

[8] Ege, B., Kavun, E.B., Yalçın, T.: Memory Encryption for Smart Cards. In: Prouff, E. (ed.) CARDIS 2011. LNCS, vol. 7079, pp. 199–216. Springer, Heidelberg (2011)

[9] Bertoni, G., Daemen, J., Peeters, M., Van Assche, G.: Keccak Specifications (2009)

[10] Guo, J., Peyrin, T., Poschmann, A.: The PHOTON Family of Lightweight Hash Functions. In: Rogaway, P. (ed.) CRYPTO 2011. LNCS, vol. 6841, pp. 222–239. Springer, Heidelberg (2011)

[11] Aumasson, J.-P., Henzen, L., Meier, W., Naya-Plasencia, M.: Quark: A Lightweight Hash. In: Mangard, S., Standaert, F.-X. (eds.) CHES 2010. LNCS, vol. 6225, pp. 1–15. Springer, Heidelberg (2010)

[12] Hell, M., Johansson, T., Meier, W.: Grain: A Stream Cipher for Constrained Environments. Int. J. Wire. Mob. Comput. 2(1), 86–93 (2007)

[13] De Cannière, C., Dunkelman, O., Knežević, M.: KATAN and KTANTAN — A Family of Small and Efficient Hardware-Oriented Block Ciphers. In: Clavier, C., Gaj, K. (eds.) CHES 2009. LNCS, vol. 5747, pp. 272–288. Springer, Heidelberg (2009)

[14] Bogdanov, A., Knežević, M., Leander, G., Toz, D., Varıcı, K., Verbauwhede, I.: spongent: A Lightweight Hash Function. In: Preneel, B., Takagi, T. (eds.) CHES 2011. LNCS, vol. 6917, pp. 312–325. Springer, Heidelberg (2011)

[15] Bogdanov, A., Knudsen, L.R., Leander, G., Paar, C., Poschmann, A., Robshaw, M.J.B., Seurin, Y., Vikkelsoe, C.: PRESENT: An Ultra-Lightweight Block Cipher. In: Paillier, P., Verbauwhede, I. (eds.) CHES 2007. LNCS, vol. 4727, pp. 450–466. Springer, Heidelberg (2007)

[16] Bertoni, G., Daemen, J., Peeters, M., Van Assche, G.: On the Indifferentiability of the Sponge Construction. In: Smart, N.P. (ed.) EUROCRYPT 2008. LNCS, vol. 4965, pp. 181–197. Springer, Heidelberg (2008)

[17] Daemen, J.: Permutation-based Encryption, Authentication and Authenticated Encryption. In: DIAC - Directions in Authenticated Ciphers, Sweden, July 5-6 (2012)

Compact Implementation and Performance Evaluation of Hash Functions in ATtiny Devices

Josep Balasch[1], Barış Ege[2], Thomas Eisenbarth[3], Benoit Gérard[4], Zheng Gong[5], Tim Güneysu[6], Stefan Heyse[6], Stéphanie Kerckhof[4], François Koeune[4], Thomas Plos[7], Thomas Pöppelmann[6], Francesco Regazzoni[8], François-Xavier Standaert[4], Gilles Van Assche[9], Ronny Van Keer[9], Loïc van Oldeneel tot Oldenzeel[4], and Ingo von Maurich[6]

[1] Department of Electrical Engineering ESAT/COSIC, KULeuven, Belgium
[2] Digital Security Group - ICIS, Radboud Universiteit Nijmegen, The Netherlands
[3] Dept. of Electrical & Computer Engineering, Worcester Polytechnic Institute, USA
[4] ICTEAM/ELEN/Crypto Group, Université catholique de Louvain, Belgium
[5] School of Computer Science, South China Normal University
[6] Horst Görtz Institute for IT-Security, Ruhr-Universität Bochum, Germany
[7] Institute for Applied Information Processing and Communications (IAIK), Graz University of Technology, Austria
[8] ALaRI Institute, University of Lugano, Switzerland
[9] STMicroelectronics

Abstract. The pervasive diffusion of electronic devices in security and privacy sensitive applications has boosted research in cryptography. In this context, the study of lightweight algorithms has been a very active direction over the last years. In general, symmetric cryptographic primitives are good candidates for low-cost implementations. For example, several previous works have investigated the performance of block ciphers on various platforms. Motivated by the recent SHA3 competition, this paper extends these studies to another family of cryptographic primitives, namely hash functions. We implemented different algorithms on an ATMEL AVR ATtiny45 8-bit microcontroller, and provide their performance evaluation. All the implementations were carried out with the goal of minimizing the code size and memory utilization, and are evaluated using a common interface. As part of our contribution, we make all the corresponding source codes available on a web page, under an open-source license. We hope that this paper provides a good basis for researchers and embedded system designers who need to include more and more functionalities in next generation smart devices.

1 Introduction

Whenever trying to compare different algorithms, such as in the currently running SHA3 competition for choosing a new standard hash function, compact implementations in small embedded devices are an important piece of the puzzle. In particular, they usually reveal a part of the algorithms complexity that does not directly appear in high-end devices, e.g., the need to share resources or

S. Mangard (Ed.): CARDIS 2012, LNCS 7771, pp. 158–172, 2013.

to minimize memory. Besides, implementations in small embedded devices such as smart cards, RFIDs and sensor nodes are also motivated by an increasing number of applications. As a result, studying the performance of cryptographic algorithms systematically in this challenging scenario is generally useful.

In a recent work, the implementation of 12 lightweight and standard block ciphers in an ATMEL AVR ATtiny45 has been investigated [14]. In order to increase the relevance of their work, the authors additionally provided open source codes for all their implementations on a public web page. In this paper, we extend this initiative towards hash functions. For this purpose, we considered three main types of algorithms. First, we targeted SHA256 and the SHA3 finalists. For the latter ones, we only focused on the candidates satisfying the SHA3 security requirements for the 256-bit output length [23], i.e., providing at least 2^{256} (second) preimage resistance and 2^{128} collision resistance. Second, we selected a number of recently published lightweight hash functions, providing both 2^{80} and 2^{128} "flat" security levels[1] [24]. Eventually, we also implemented several block cipher based constructions, e.g., relying on the AES Rijndael. For all these algorithms, we aimed for the same optimization criteria (namely small source code size and limited memory use) and used a uniform interface (see the details in Section 2). Resistance against physical (e.g., side-channel, fault) attacks was explicitly excluded from the requirements. As the project involves many different programmers, we naturally acknowledge possible biases in our performance evaluation results, due to slightly different implementation choices and interpretation of the guidelines. In order to mitigate these (usual) limitations, we provide all our source codes on a public web page [1]. As a result, we hope that this initiative can be used as a first step in better understanding the performance of hash functions in a specific but meaningful class of devices.

Selected Algorithms. We investigated hash functions in three main categories. First, we considered SHA256 [22] and SHA3 candidates BLAKE-256 [4], Grøstl-256 [18], JH-256 [34], Keccak[r=1088,c=512] [6,7] and Skein-512-256 [17]. Second, we evaluated the lightweight hash functions Quark (S and Q versions) [2], PHOTON (160/36/36 and 256/32/32 versions) [19], SPONGENT (160/160/80 and 256/256/128 versions) [9] and Keccak (i.e. low-cost alternatives to the standard version). Eventually, we also focused on block cipher based constructions such as Rogaway-Steingberger [27], Hirose [20], Davies-Meyer and Shrimpton-Stam [29], based on NOEKEON [10], AES-256, Rijndael 256 [11] and SEA-192 [30]. More details on these algorithms are given in the extended paper [5].

2 Methodology and Metrics

In order to be able to compare the performance of the different hash functions in terms of speed and memory space, the developers were asked to respect a list of common constraints, detailed hereunder. (1) The code has to be written in

[1] i.e. the same security is required for collision, preimage and 2nd preimage resistance.

assembly, if possible in a single file. It has to be commented and easily readable, for example, giving the functions the name they have in their original specifications. (2) The function has to be implemented in a low-cost way, minimizing the code size and the RAM use. (3) Data does not have to be preserved by the hashing process. This allows direct modification of the data zones in RAM, hence reducing the amount of memory needed. (4) The interface should be made up of 3 functions. (a) *init* takes no input and initializes the internal state, which is a dedicated memory zone seen as a black box, and returns no output; (b) *update* takes as input a full block of data, updates its internal state by processing that block and returns no output; (c) *final* takes as input the (possibly empty) last chunk of data together with its size and processes it before finalizing the hash computation. By convention, the data passed to *final* is necessarily an incomplete block. (5) Data exchanges are performed with pre-defined memory zones where data has to be put before calling functions, or can be found on their return. For example, the data block to hash has to be put at the pre-defined address *SRAM_DATA* before a call to *update*, and the final hash can be found at *SRAM_STATE* on return of *final*. Most input/output values are thus implicitly passed. The only explicitly passed value is the size of the data passed to *final*. (6) Only the internal state is preserved between calls to these functions. No assumption can be made that other RAM zones (e.g. *SRAM_DATA*) or registers will stay unchanged. (7) The target device is an 8-bit microcontroller from the ATMEL AVR device family, more precisely the ATtiny45. It has a reduced set of instructions and no hardware multiplier. A common interface file was provided to all designers (available on [1]). Note that for some functions (e.g., for block cipher based), the padding was not explicitly defined. In these cases, we appended n null bytes, followed by the length of the message coded as a 64-bit value, where n is chosen to make the global message length a multiple of the block size. The basic metrics considered for evaluation are code size, number of RAM words, and cycle count. Performances were measured on 4 different message lengths: 8, 50, 100 and 500 bytes, ranging from a very small (smaller than one block) to a large message. Finally note that some of the guidelines were not always followed, because of the cipher specifications making them less relevant (which will be specified when necessary).

3 Description of the ATtiny45 Microcontroller

The ATtiny45 is a 8-bit RISC microcontroller from ATMEL's AVR series. The microcontroller uses a Harvard architecture with separate instruction and data memory. Instructions are stored in a 4 kB Flash memory (2048 × 16 bits). Data memory involves the 256-byte static RAM, a register file with 32 8-bit general-purpose registers, and special I/O memory for peripherals like timers, analog-to-digital converters or serial interfaces. Different direct and indirect addressing methods are available to access data in RAM. Especially indirect addressing allows accessing data in RAM with very compact code size. Moreover, the ATtiny45 integrates a 256-bytes EEPROM for non-volatile data storage.

The instruction-set of the microcontroller contains 120 instructions which are typically 16-bits wide. Instructions can be divided into arithmetic logic unit (ALU) operations (arithmetic, logical and bit operations) and conditional and unconditional jump and call operations. The instructions are processed within a two-stage pipeline with a pre-fetch and an execute phase. Most instructions are executed within a single clock cycle, leading to a good instructions-per-cycle ratio. Compared to other microcontrollers from ATMEL's AVR series such as the ATmega devices, the ATtiny45 has a reduced instruction set (e.g. no multiply instruction), smaller memories (Flash, RAM, EEPROM), no in-system debug capability, and less peripherals. The ATtiny45 also has lower power consumption and is cheaper.

4 Implementation Details

4.1 SHA256 and SHA3 Candidates

SHA256. Like its predecessor SHA1, SHA256 is optimized for 32-bit software implementation. Hence, it can be expected to be similarly efficient on 8-bit AVR processors. When implementing the iteration step of its compression function, the main observation is that six out of eight working registers are just circularly copied. To reduce code and cycles for memory transfer operations, the addresses of the RAM-based working registers are reassigned using circular pointer arithmetic instead of addressing these registers by its names `A-H` explicitly.

Circular pointer arithmetic as part of the iteration step is also used to update the input word according to the message expansion. Besides 32-bit modular additions, SHA2 requires 32-bit right rotations by $r = \{2, 6, 7, 11, 13, 17, 18, 19, 22, 25\}$ bits and right shifts by $s = \{3, 10\}$. Rotations and shifts by parameters larger than 8 bits first swap 8-bit register accordingly; then single bit operations on the swapped 32-bit word are performed to correspond to $f = \{r, s\} \bmod 8$. SHA256 uses up to three 32-bit bit rotations processing the same input in a row so that reordering of rotation and shift operations by ascending f-values improves efficiency.

BLAKE-256. The RAM consumption is mainly due to storing 64 byte input data, 64 byte state, 32 byte chain value, 8 byte salt, and an 8 byte counter. The initialization vectors (32 byte) and constants (64 byte) are stored in the flash memory of the microcontroller. We refrained from transferring the constant table into the RAM in order to keep RAM consumption low. BLAKE's permutation table σ consists of 10×16 entries. However, each entry is only a four bit number so we merged two entries in one byte and later select the upper/lower 4-bits by masking. Thus, the permutation table requires just 80 instead of 160 bytes in ROM. In order to maintain a decent performance while keeping the code size down we incorporated the observation by Osvik [25] to efficiently load and store in-/outputs of the round function $G_i\,(a, b, c, d)$. Furthermore, we use loops where applicable and move recurring tasks such as loading and storing the counter into functions. An exception to this rule is the implementation of the round function.

Since it is called 80 times when hashing one message block its runtime heavily impacts the overall performance. Therefore, we decided to unroll critical parts of the round function.

Grøstl-256. Grøstl has a state of 64 bytes. During the update function, we need to keep the state, the input message and the previously computed hash in memory. Thus, we need 192 byte of RAM. The ShiftBytes is computed by offloading each row, one at a time, from the state into the register of the microcontroller and then writing it back in the new position. In order to increase the performance and reduce the number of accesses to the memory, the SubBytes is computed together with the ShiftBytes. The MixBytes is computed as proposed by Johannes Feichtner [16,28], and is carried out one column at a time. Finally, to easily compute the padding, 8 bytes of memory are used to keep track of the numbers of messages. This 8 bytes are copied directly in the appropriate position of the padding block.

JH-256. Specifications for a bitsliced implementation of JH are available, but require to store 42 256-bit round constants in memory, which is not compliant with our low-cost constraints. Hence, JH was implemented according to the reference specifications. The utilization percentage of the RAM is high as JH needs 128 bytes to store the state, 64 for the input block and 32 for the round constant. In order to improve the performance, the S-box and linear transformation were combined into two look-up tables, of 32 bytes each, as was done in the optimized 8-bit implementation provided by JH author [33]. For the same reason, the initial state was precomputed and stored in program memory. It allows us to save the initialization phase which is equivalent to the processing of one input block. Regarding the permutation, it is performed by reading the state bytes in a different order at the beginning of each round. Finally, the state bits are reorganized at the beginning and end of each function E8. This bitwise permutation is time consuming and requires additional memory. Those problems can be partially prevented by reorganizing the input bytes before XORing them with the state.

Keccak. In a first level, we implemented the sponge construction, which comes down to XORing r-bit message blocks into the state, with $r > 0$ the *rate*, and to calling the underlying permutation. In a second level, we implemented the permutations Keccak-$f[b]$ for $b \in \{200, 400, 800, 1600\}$. The sponge construction imposes that the *capacity* c is twice the security strength level and that $b = r + c$, and our implementation allows any combination of rate and capacity under these constraints. For clarity, the benchmark focuses on three specific instances: the SHA3 candidate Keccak$[r = 1088, c = 512]$, and the lightweight variants Keccak$[r = 144, c = 256]$ and Keccak$[r = 40, c = 160]$ for the 128-bit and 80-bit security strengths levels, respectively. Any pair of instances with $c = 256$ and $c = 160$ would have satisfied the requirements, but our choice aims at minimizing b for a given c and thereby the RAM usage, consistently with a lightweight context. Inside the implementation, some operations (i.e., the rotations in θ and ρ) are performed on a lane basis, mapping a lane to $b/200$

byte(s). Some other operations, such as χ or the parity computation in θ, are instead slice-oriented, taking advantage of the representation of 8 consecutive slices in 25 bytes [8]. Note that in the specific case of Keccak-f[200], the two approaches collide as the state contains exactly 8 slices or 25 lanes, mapped to 25 bytes. RAM usage is composed of $b/8$ bytes for the state and some working memory ($b/40$ bytes, or 0 for Keccak-f[200] as the AVR registers suffice). If the desired output length is greater than the rate (e.g., for lightweight instances), an additional output buffer is needed to perform the squeezing phase.

Skein-x-y. We implemented the SHA3 finalist Skein-512-256, with an output of 256 bits, limited to the hashing functionality. The internal state is therefore made of eight 64-bit words. To keep the program memory space small and the code readable, some basic 64-bits functions like loading, saving, adding, ..., have been employed. The registers are only used temporarily, except the round counter. The message, the state, the key, the key-schedule and the tweak are always in the data space, and modified directly. The three main Threefish functions (addkey, mix and permute) were implemented following the reference specifications. Besides, the modulo 3 and modulo 9 values used in the key schedule were saved in the program memory space. We have also developed Skein-256-256, slightly optimized for the speed and data memory space performance, by leaving most of the time three out of the four state words in the registers.

4.2 Lightweight Hash Functions

S-Quark and D-Quark. The critical point in the implementation of QUARK hash functions is the update of the state[2]. This update phase considers the state as two LFSRs that will be updated using three retro-action polynomials[3]. This design is thought for hardware, a context where it is very efficient, but is much more expensive in software. Nevertheless, our choice to implement this step using a bit-slice approach provides rather good performance. The platform is an 8-bit microprocessor and the retro-action polynomials are such that the last 8 bits of each LFSR are not considered. Hence, our implementation performs 8 updates at the same time reducing from 1024/704 to 128/88 polynomial computations. The state is stored in RAM, as it is too large to be kept in registers. Computations are ordered in such a way that the shift of the state is performed on the fly.

PHOTON-160/36/36 and PHOTON-256/32/32. First note that these implementation significantly differ, since PHOTON-160 has a state matrix with 4-bit cells and uses the PRESENT S-box while PHOTON-256 has 8-bit entries and uses the AES S-box. This results in different implementation strategies.

[2] During implementation, a minor inconsistency was discovered between the paper description [2] and the reference code [3], which use different bit ordering conventions. We chose to comply with the description provided in the original article. Compliance with the C code can be obtained by inverting the order of bits in the input message.

[3] An additional third will provide constants for the 1024/704 executions required to apply the permutation P.

The state of the implemented PHOTON-160/36/36 variant consists of 7-by-7 4-bit elements which are packed into 25 bytes in order to save memory. This allows an optimal usage of the RAM but naturally also results in additional code in order to extract the correct nibble out of the state. It is a trade-off between code size/speed and RAM usage. As the interface only allows messages that are a multiple of 8 bits while each iteration of a PHOTON-160/36/36 round function absorbs 36 bits, we just process an input block of length 72 bits and call the PHOTON round function internally twice for a full 72-bit block. The largest amount of computational time is spend in the permutation layer for ShiftRows and especially during the MixColumnsSerial step as finite field arithmetic has to be carried out on 4-bit values. The internal state of PHOTON-256/32/32 consists of 36 bytes, arranged as a 6-by-6 matrix, that goes over four different transformations to produce a 32 byte hash digest. Due to their sizes both state and digest have to be stored in SRAM. This generates an inherent implementation overhead, as state bytes need to be fetched from and stored to SRAM once for each transformation. We partially reduce this overhead by merging all row-based transformations, and also by incrementing code size. Due to its use of AES-like permutations, the implementation of the PHOTON-256/32/32 transformations can be carried out quite efficiently on 8-bit controllers. The SubCells transformation is implemented as a memory aligned lookup table resulting in important cycle savings. The MixColumnsSerial transformation, consisting of six consecutive calls to the AES MixColumns transformation, is similarly optimized by implementing the multiplication by ‘02’ as a memory aligned LUT [12].

SPONGENT-160/160/80 and -256/256/128. The SPONGENT-160 state is $160 + 80 = 240$ bits or 30 bytes large. Therefore, the state can be stored in the registers already available on the target device. However, SPONGENT uses a PRESENT-like bit permutation in π_b and therefore every output bit of an S-box is mapped to a distinct nibble after permutation. If we were to store the state in the available registers, we would only have two registers for additional computations and this would lead to a large code size when implementing the bit permutation. Therefore, the state is stored in SRAM and a three-step iterative approach is used for the bit permutation to achieve a smaller code size. For the permutation, each four consecutive nibbles are permuted and stored in SRAM at the same places. Then, the permuted nibbles are re-ordered to obtain permuted bytes and finally bytes are re-ordered to their appropriate places in the state. Although this approach is code-size efficient, note that it leads to an increase in running time of the overall hashing process. The remaining operations like round constant computation, padding and control logic are implemented in a straightforward manner. The state of SPONGENT-256 is $256 + 128 = 384$ bits or 48 bytes large. Since the state does not fit into the available registers, we optimized this variant with respect to code size and the state is kept in SRAM. For the permutation, iteratively four successive bytes are loaded into registers and the permuted byte is constructed from two bits at fixed offsets of each of these four bytes. Afterwards the processed bytes are stored back to SRAM. This method keeps the code very small but requires a copy of the 48 bytes state and therefore doubles the required memory. Besides the two states no additional memory is required. The S-boxes are stored in flash memory and must be aligned to a address dividable by 16 for easier pointer arithmetic. Again, the remaining operations are straightforwardly implemented.

4.3 Block Cipher-Based Constructions

Rogaway-Steinberger LP/lp362. For realizing the Rogaway and Steinberger construction principle, the matrix A suggested by Lee and Park [21] with $\alpha = 2$ has been used. For operations in $\mathbb{F}_{2^{128}}$ (addition and multiplication) we have selected the same irreducible polynomial $x^{128} + x^7 + x^2 + x + 1$ as stated in [27]. The implementation of the block cipher NOEKEON is based on the open source version published in [14], but the decryption functionality has been removed since it is not required for the generation of a permutations. Two variants of the Rogaway-Steinberger scheme have been implemented: LP362 and lp362. The two variants mainly differ in code size. The lp362 scheme uses a single fixed key for all permutations, leading to about 100 bytes less code than for the LP362 scheme which uses a different fixed key for each of the six permutations. Both variants have similar execution time, consume 92 bytes of RAM, and make use of 8 registers for computing the hash value of a message.

Hirose Double Block Length (DBL) Construction. For simplicity we chose an all-zero IV and the additive constant to be 1. One of the advantages of Hirose is that the two parallel AES executions use the same key. However, due to memory restrictions, the key should be computed on-the-fly. Hence, the two encryptions need to be processed in parallel. The AES design is similar to the one presented in [14], with a further optimized `Shift_Rows` operation. Decryption code is not needed and has been removed. The key scheduling is performed on-the-fly and and processes 32 bit at a time. The full 128-bit state of one encryption block is kept in the registers. Since both encryptions are performed in parallel, the two states have to be swapped in and out of SRAM regularly. Due to the large key size, the swap is performed as little as every 4 rounds, keeping the resulting overhead at a minimum. The implementation needs 82 bytes of RAM. We chose not to overwrite the input to the update function, which results in a need for 16 additional RAM bytes for the input. By overwriting the input these additional 16 bytes can be saved if RAM size is critical.

Davies-Meyer Construction. The implementation of the Davies-Meyer construction simply requires making a copy of the message to be XORed with the resulting encryption, resulting in an additional consumption of 32 bytes of RAM.

Shrimpton-Stam Construction. The implementation of Shrimpton-Stam construction only requires to take care of remembering inputs of the ciphers to be able to XOR them to the result of the encryptions. We chose simple keys to instantiate the functions f_i so that no extra memory is required to store them. More precisely, we respectively set all key bytes to `0x00, 0x11` and `0x22` for f_1, f_2 and f_3.

Rijndael-256/256. The operations to be performed during a Rijndael-256/256 encryption are simple and can be made efficient using the well-known techniques for implementing AES on lightweight processors, like the use of a lookup table for the S-box and the efficient multiplication by '02' for MixColumns [13]. The main issue when working on an ATtiny45 is the state size: whereas AES state can be kept in registers, this is not possible any more for 256-bit blocks. As RAM accesses are time-consuming on the ATtiny, the design of this implementation focuses on minimizing the number of these accesses. This has been done by reorganizing the round loop (without, of course, affecting the behaviour of the cipher) in such a way that the round ends with a ShiftRows operation. Additionally, we used an auxiliary state to perform ShiftRows efficiently. As a result, we can fetch a full column from RAM, immediately perform MixColumns, AddRoundKey and SubBytes, and write the result in the auxiliary RAM state, taking the effect of ShiftRow into account to determine the exact locations in RAM. The next round is then performed similarly, but writing data from the auxiliary state to the initial one, and so on.

SEA. The reference code was written following directly the cipher specifications, and is a natural extension of the 96-bit version designed in [14]. During its execution, plaintexts and keys are stored in RAM (accounting for a total of 48 bytes), limiting the register consumption to 12 registers for the running state, one register for the round counter and some additional temporary storage. The S-box was implemented using its bitslice representation. The block cipher was then inserted in a Davies-Meyer mode of operation, using a similar code as the version using Rijndael-256/256. Overall, the implementation maintains low code size and RAM use at the cost of a large cycle count, mainly due to the large number of rounds (177) in the 196-bit version of the cipher based on 8-bit words.

5 Performance Evaluation and Conclusions

We first refer to a number of other implementations of hash functions in ATMEL AVR devices [8,15,25,26,28,31,32]. In general, these previous works present benchmarking results in devices from the ATmega family rather than the ATtiny one, hence tolerating larger code sizes and RAM use. As they are hardly comparable with ours and because of space constraints, we do not detail them in this section. Overall, we believe they provide a complementary view to ours. In particular, the pretty complete comparisons of the XBX website certainly sheds another light on the different algorithms [32]. Note also that some of these previous works consider older versions of the SHA3 candidates. Our following results consider the exact SHA3 finalists, according to their last updated specifications. We recall that for the functions appearing several times in the tables (e.g. Keccak, Skein, Quark, PHOTON, SPONGENT), the different lines correspond to different specifications and not different implementations of the same algorithm.

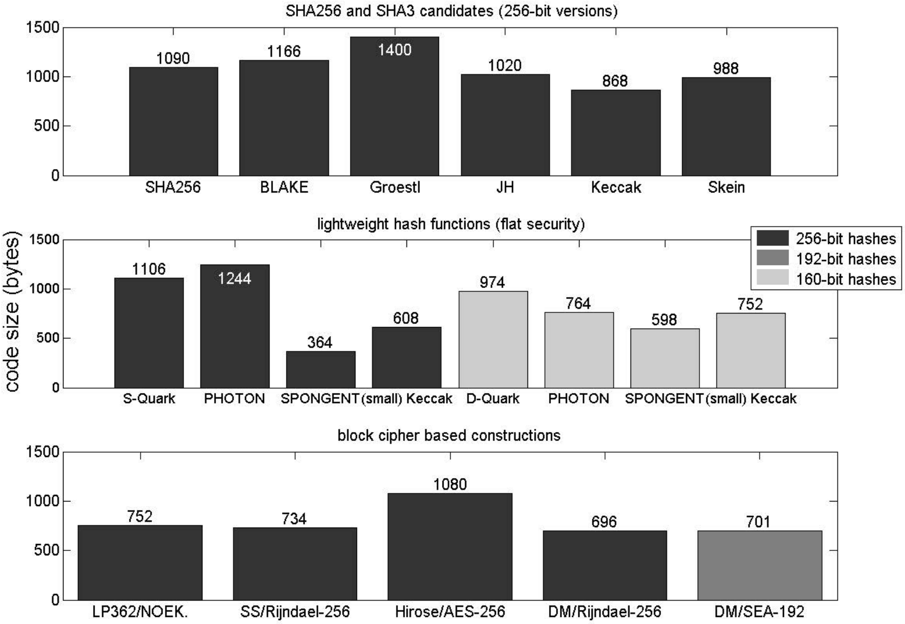


Fig. 1. Performance evaluation: code size (bytes)

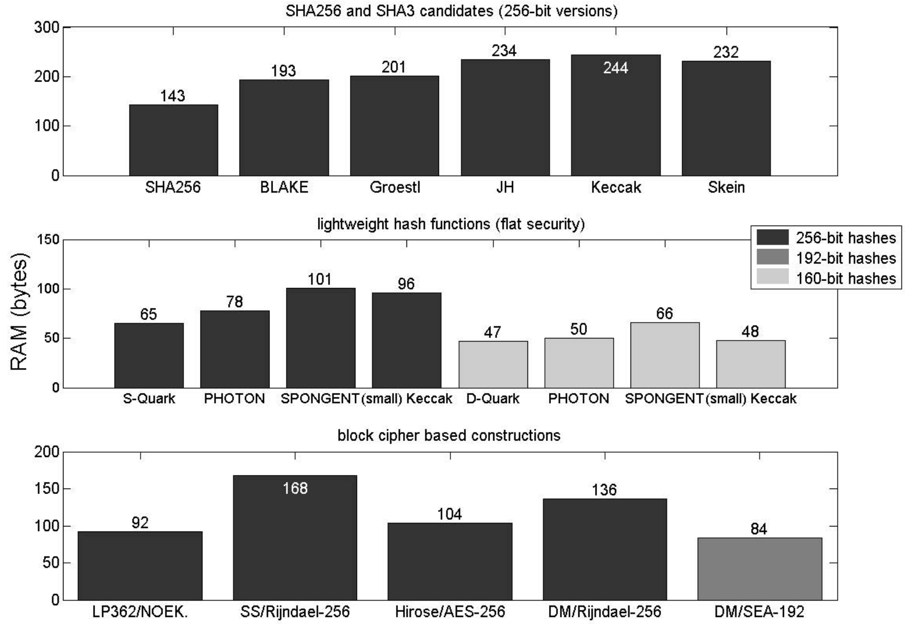


Fig. 2. Performance evaluation: RAM (bytes)

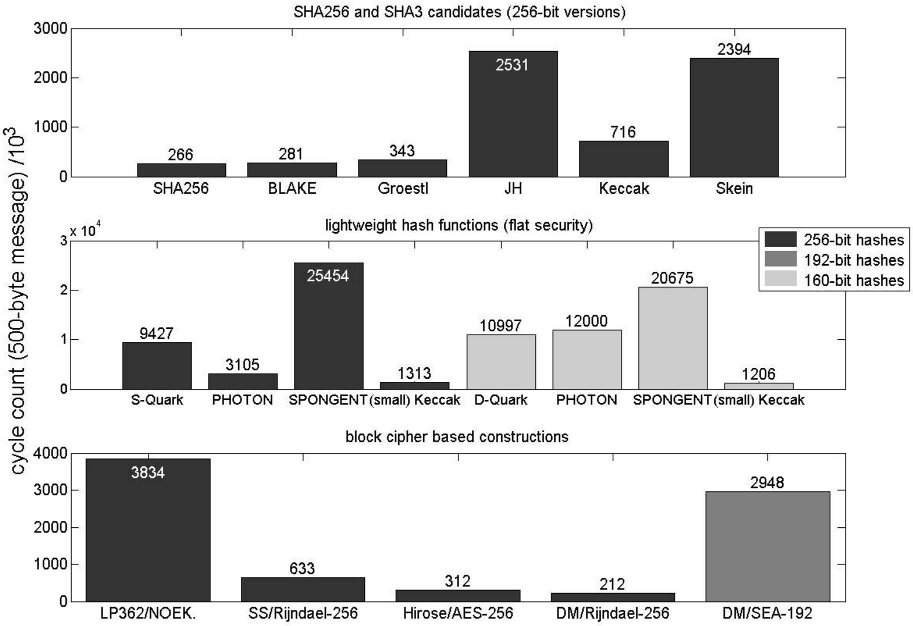


Fig. 3. Performance evaluation: cycle count (500-byte message)

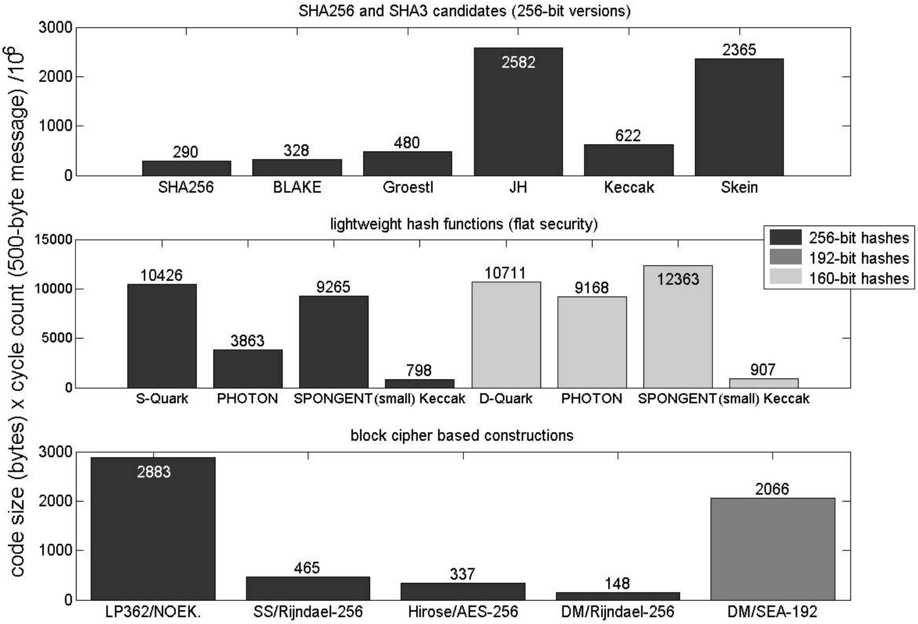


Fig. 4. Performance evaluation: code size (bytes) x cycle count (500-byte message)

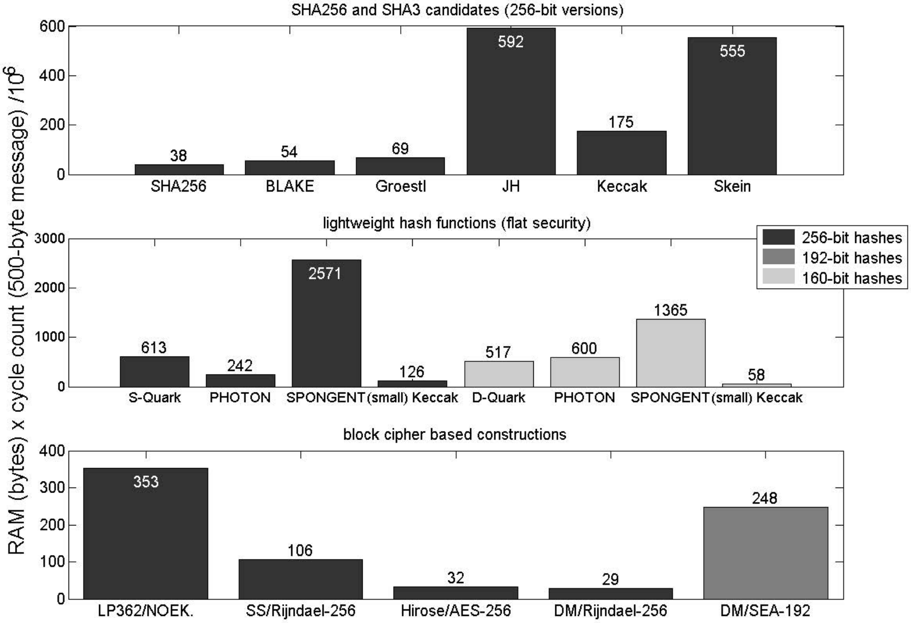


Fig. 5. Performance evaluation: RAM (bytes) x cycle count (500-byte message)

Following Section 2, we evaluated the performance of our different algorithms based on three main metrics, namely the code size (in bytes), RAM use (in bytes) and cycle counts for different message sizes[4]. They are represented in Figures 1, 2 and 3. Besides, we also produced so-called combined metrics that aim to summarize the efficiency of the hash functions in the ATtiny45. We used the product of the code size and cycle count and the product of the RAM use and cycle count for this purpose. Eventually, we note that all our results are given numerically in the performance tables of the extended paper [5]. As already mentioned, these results have to be interpreted with care, as they both represent the skills of the programmer and the algorithms efficiency. Yet, given this cautionary note, we believe a number of general observations can be extracted.

First, the code size and RAM usage illustrate that the implementation constraints were reached for all algorithms. Nevertheless, the cost of the SHA3 candidates is generally higher than the one of both lightweight hash functions and block cipher based constructions. For some of them, the RAM use is close to the limit of the ATtiny device (i.e. 256). This can be explained by the generally larger states of all SHA3 candidates. Second, we observe that lightweight algorithms have large cycle counts compared to other hash functions. This implies that their

[4] Note that for certain (e.g., sponge-based) functions, the data part of the RAM could be arbitrarily reduced by changing the interface. In this case, the RAM use evaluation in the figures excluded the data RAM (reported in gray in the tables).

overall efficiency (measured with the combined metrics) is generally low in our implementation context. By contrast, the flexible nature of sponge-based functions (including all lightweight proposals) allows reducing the RAM usage quite significantly, which is an interesting feature for hardware and embedded software implementations. Third, it is noticeable that the SHA3 candidates hardly compete with AES-256 in Hirose construction or Rijndael-256-256 in Davies-Meyer mode. This observation is quite consistently observed for all our metrics. Eventually, and as far as SHA3 finalists (in the 256-bit versions) are concerned, our investigations suggest that BLAKE offers the best performance figures, followed by Grøstl, Keccak, Skein and JH.

All these results were naturally obtained within a limited time frame. Hence, we encourage the reader to download codes and possibly improve them with further optimization. Looking at how the AES implementations have evolved following its selection as standard, it is likely that similar improvements can be expected for the hash functions in this work.

Acknowledgments. This work has been funded in part by the European Commission's ECRYPT-II NoE (ICT-2007-216676), by the Belgian State's IAP program P6/26 BCRYPT, by the ERC project 280141 (acronym CRASH), by the 7th framework European project TAMPRES, by the Walloon region's S@T Skywin and MIPSs projects. Stéphanie Kerckhof is a PhD student funded by a FRIA grant, Belgium. François-Xavier Standaert is a Research Associate of the Belgian Fund for Scientific Research (FNRS-F.R.S). Zheng Gong is supported by NSFC (No. 61100201).

References

1. http://perso.uclouvain.be/fstandae/source_codes/hash_atmel/
2. Aumasson, J.-P., Henzen, L., Meier, W., Naya-Plasencia, M.: Quark: A lightweight hash. In: Mangard, S., Standaert, F.-X. (eds.) CHES 2010. LNCS, vol. 6225, pp. 1–15. Springer, Heidelberg (2010)
3. Aumasson, J.-P., Henzen, L., Meier, W., Naya-Plasencia, M.: QUARK C implementation (2010), https://www.131002.net/quark/
4. Aumasson, J.-P., Henzen, L., Meier, W., Phan, R.C.-W.: SHA-3 proposal BLAKE. Submission to NIST, Round 3 (2010)
5. Balasch, J., Ege, B., Eisenbarth, T., Gérard, B., Gong, Z., Güneysu, T., Heyse, S., Kerckhof, S., Koeune, F., Plos, T., Pöppelmann, T., Regazzoni, F., Standaert, F.-X., Van Assche, G., Van Keer, R., van Oldeneel tot Oldenzeel, L., von Maurich, I.: Compact implementation and performance evaluation of hash functions in attiny devices. Cryptology ePrint Archive, Report 2012/507 (2012), http://eprint.iacr.org/
6. Bertoni, G., Daemen, J., Peeters, M., Van Assche, G.: Sponge functions. Ecrypt Hash Workshop 2007 (May 2007), also available as public comment to NIST from http://www.csrc.nist.gov/pki/HashWorkshop/Public_Comments/2007_May.html
7. Bertoni, G., Daemen, J., Peeters, M., Van Assche, G.: The Keccak reference (January 2011), http://keccak.noekeon.org/
8. Bertoni, G., Daemen, J., Peeters, M., Van Assche, G., Van Keer, R.: Keccak implementation overview (September 2011), http://keccak.noekeon.org/

9. Bogdanov, A., Knezevic, M., Leander, G., Toz, D., Varici, K., Verbauwhede, I.: Spongent: The design space of lightweight cryptographic hashing. IACR Cryptology ePrint Archive, 2011:697 (2011)
10. Daemen, J., Peeters, M., Van Assche, G., Rijmen, V.: Nessie proposal: NOEKEON (2000), http://gro.noekeon.org/Noekeon-spec.pdf
11. Daemen, J., Rijmen, V.: The block cipher rijndael. In: Quisquater, J.-J., Schneier, B. (eds.) CARDIS 2000. LNCS, vol. 1820, pp. 277–284. Springer, Heidelberg (2000)
12. Daemen, J., Rijmen, V.: The Design of Rijndael. Springer-Verlag New York, Inc., Secaucus (2002)
13. Daemen, J., Rijmen, V.: AES proposal: Rijndael. In: Proc. First AES Conference (August 1998), Available on-line from the official AES page: http://csrc.nist.gov/encryption/aes/aes_home.htm
14. Eisenbarth, T., Gong, Z., Güneysu, T., Heyse, S., Indesteege, S., Kerckhof, S., Koeune, F., Nad, T., Plos, T., Regazzoni, F., Standaert, F.-X., van Oldeneel tot Oldenzeel, L.: Compact implementation and performance evaluation of block ciphers in attiny devices. In: Mitrokotsa, A., Vaudenay, S. (eds.) AFRICACRYPT 2012. LNCS, vol. 7374, pp. 172–187. Springer, Heidelberg (2012)
15. Eisenbarth, T., Heyse, S., von Maurich, I., Poeppelmann, T., Rave, J., Reuber, C., Wild, A.: Evaluation of sha-3 candidates for 8-bit embedded processors. In: The Second SHA-3 Candidate Conference (2010)
16. Feichtner, J.: http://www.groestl.info/implementations.html
17. Ferguson, N., Lucks, S., Schneier, B., Whiting, D., Bellare, M., Kohno, T., Callas, J., Walker, J.: The skein hash function family, version 1.3 (2010), http://www.skein-hash.info/
18. Gauravaram, P., Knudsen, L.R., Matusiewicz, K., Mendel, F., Rechberger, C., Schläffer, M., Thomsen, S.S.: Sha-3 proposal grøstl, version 2.0.1 (2011), http://www.groestl.info/
19. Guo, J., Peyrin, T., Poschmann, A.: The PHOTON family of lightweight hash functions. In: Rogaway, P. (ed.) CRYPTO 2011. LNCS, vol. 6841, pp. 222–239. Springer, Heidelberg (2011)
20. Hirose, S.: Some plausible constructions of double-block-length hash functions. In: Robshaw, M.J.B. (ed.) FSE 2006. LNCS, vol. 4047, pp. 210–225. Springer, Heidelberg (2006)
21. Lee, J., Park, J.H.: Preimage resistance of lpmkr with r=m-1. Inf. Process. Lett. 110(14-15), 602–608 (2010)
22. National Institute of Standards and Technology. FIPS 180-3, Secure Hash Standard, Federal Information Processing Standard (FIPS), Publication 180-3. Technical report, U.S. Department of Commerce (October 2008)
23. NIST. Announcing request for candidate algorithm nominations for a new cryptographic hash algorithm (SHA-3) family. Federal Register Notices 72(212), 62212–62220 (November 2007), http://csrc.nist.gov/groups/ST/hash/index.html
24. NIST. NIST special publication 800-57, recommendation for key management (revised) (March 2007)
25. Osvik, D.A.: Fast embedded software hashing. Cryptology ePrint Archive, Report 2012/156 (2012), http://eprint.iacr.org/
26. Otte, D.: Avr-crypto-lib (2009), http://www.das-labor.org/wiki/Crypto-avr-lib/en
27. Rogaway, P., Steinberger, J.P.: Constructing cryptographic hash functions from fixed-key blockciphers. In: Wagner, D. (ed.) CRYPTO 2008. LNCS, vol. 5157, pp. 433–450. Springer, Heidelberg (2008)

28. Roland, G.: Efficient implementation of the grøstl-256 hash function on an atmega163 microcontroller (June 2009), http://groestl.info/groestl-0-8bit.pdf
29. Shrimpton, T., Stam, M.: Building a collision-resistant compression function from non-compressing primitives. In: Aceto, L., Damgård, I., Goldberg, L.A., Halldórsson, M.M., Ingólfsdóttir, A., Walukiewicz, I. (eds.) ICALP 2008, Part II. LNCS, vol. 5126, pp. 643–654. Springer, Heidelberg (2008)
30. Standaert, F.-X., Piret, G., Gershenfeld, N., Quisquater, J.-J.: SEA: A scalable encryption algorithm for small embedded applications. In: Domingo-Ferrer, J., Posegga, J., Schreckling, D. (eds.) CARDIS 2006. LNCS, vol. 3928, pp. 222–236. Springer, Heidelberg (2006)
31. Walter, J.: Fhreefish (skein implementation) website, http://www.syntax-k.de/projekte/fhreefish/
32. Wenzel-Benner, C., Gräf, J., Pham, J., Kaps, J.-P.: XBX benchmarking results January 2012. In: Third SHA-3 Candidate Conference (March 2012), http://xbx.das-labor.org/trac/wiki/r2012platforms_atmega1284p_16mhz
33. Wu, H.: JH Documentation Website, http://www3.ntu.edu.sg/home/wuhj/research/jh/
34. Wu, H.: The Hash Function JH (January 2011), http://www3.ntu.edu.sg/home/wuhj/research/jh/

Putting together What Fits together - GrÆStl

Markus Pelnar[1,2], Michael Muehlberghuber[1], and Michael Hutter[2]

[1] Integrated Systems Laboratory (IIS), ETH Zurich,
Gloriastrasse 35, 8092 Zurich, Switzerland
m.pelnar@student.tugraz.at, mbgh@iis.ee.ethz.ch

[2] Institute for Applied Information Processing and Communications (IAIK),
Graz University of Technology, Inffeldgasse 16a, 8010 Graz, Austria
michael.hutter@iaik.tugraz.at

Abstract. We present GrÆStl, a combined hardware architecture for the Advanced Encryption Standard (AES) and Grøstl, one of the final round candidates of the SHA-3 hash competition. GrÆStl has been designed for low-resource devices implementing AES-128 (encryption and decryption) as well as Grøstl-256 (tweaked version). We applied several resource-sharing optimizations and based our design on an 8/16-bit datapath. As a feature, we aim for high flexibility by targeting both ASIC and FPGA platforms and do not include technology or platform-dependent components such as RAM macros, DSPs, or Block RAMs. Our ASIC implementation (fabricated in a 0.18 μm CMOS process) needs only 16.5 kGEs and requires 742/1,025 clock cycles for encryption/decryption and 3,093 clock cycles for hashing one message block. On a Xilinx Spartan-3 FPGA, our design requires 956 logic slices and 302 logic slices on a Xilinx Virtex-6. Both stand-alone implementations of AES and Grøstl outperform existing FPGA solutions regarding low-area design by needing 79 % and 50 % less resources as compared to existing work. GrÆStl is the first combined AES and Grøstl implementation that has been fabricated as an ASIC.

Keywords: Hardware implementation, AES, Grøstl, ASIC, FPGA, embedded systems, low-resource design.

1 Introduction

Among the most commonly used cryptographic primitives in classical communication protocols are block ciphers and hash functions. The Advanced Encryption Standard (AES) [1] is by far the most widespread block cipher since its standardization in 2001. Grøstl [2] was one of the final round candidates of the SHA-3 cryptographic hash function competition from which KECCAK [3] emerged as winner. Both AES and Grøstl share several similarities such as a common S-box or similar diffusion layers which encourage the implementation of a combined hardware architecture.

In this paper, we describe a hardware architecture—called GrÆStl—that combines the functionality of AES-128 and Grøstl-256 in one piece of silicon.

S. Mangard (Ed.): CARDIS 2012, LNCS 7771, pp. 173–187, 2013.

Our design aims for high flexibility supporting both Application Specific Integrated Circuit (ASIC) and Field Programmable Gate Array (FPGA) platforms without using technology dependent components such as Random Access Memory (RAM) macros, Digital Signal Processors (DSPs), or Block RAMs. We exploit various optimization techniques to reduce the required area, for example, by sharing registers and a common datapath. The ASIC version of our design has been fabricated using the 0.18 μm Complementary Metal Oxide Semiconductor (CMOS) process technology from the United Microelectronics Corporation (UMC) and therefore represents the first tape-out version of a combined AES/Grøstl architecture in the literature. It requires only 16.5 kGEs in total and needs 742/1,025 clock cycles for AES encryption/decryption and 3,093 clock cycles for hashing. The small area requirements and also the low-power consumption of about 20 μW at 100 kHz make the design applicable to resource-constrained devices such as smart cards or contact-less powered devices. We also compared the results of our stand-alone implementations of AES and Grøstl. It shows that the implementations outperform existing FPGA solutions in terms of low-area. They require up to 79 % less resources on a Spartan-3 as compared to existing implementations.

The remainder of this paper is organized as follows. In Section 2, an overview about related work on area-constrained AES and Grøstl implementations is given. Section 3 presents the architecture of GrÆStl and describes the design starting from the top module down to the implemented (combined) datapath. In Section 4, we present our results and compare them with related work. Finally, we summarize our results and draw conclusions in Section 5.

2 Related Work

There exist many papers in the literature that present low-resource hardware implementations of AES or Grøstl. Since publication of the algorithms, several optimization techniques have been proposed that reduce the area requirements for both ASIC and FPGA platforms. One example is the optimized AES S-box implementation of Canright [4] that has been also used by Feldhofer et al. [5] to realize a very compact version of AES-128 in 2005. Their implementation requires 3.4 kGEs for both encryption and decryption. Similar results have been reported by Hämäläinen et al. [6], Kaps et al. [7], and Kim et al. [8] who reported about 4 kGEs in total. At EUROCRYPT 2011, Moradi et al. [9] presented an area-optimized implementation of (encryption-only) AES which needs about 2.4 kGEs that marks the lowest level of state-of-the-art AES implementations.

As opposed to AES, there exist only a few publications so far that describe low-area optimizations for Grøstl on ASIC devices. Tillich et al. [10] have been the first who presented an implementation requiring 14.6 kGEs. This number also corresponds well to the area estimations given in the Grøstl specification from Gauravaram et al. [2] which reported a size of less than 15 kGEs. Further implementations have been presented by Katashita et al. [11], Guo et al. [12], and Henzen et al. [13] which require between 34.8 and 72 kGEs.

In view of FPGA platforms, large effort has been made to reduce the complexity by reusing existing hardware components, e.g., Block RAMs and DSPs. Chodowiec and Gaj [14] presented a low-resource AES-128 implementation on FPGAs in 2003 and described different optimization techniques. Their implementation needs 222 slices and 3 Block RAMs on a Spartan-2 supporting both encryption and decryption. In particular, they made use of Look-Up Tables (LUTs) to efficiently implement shift registers such that intermediate values can be easily shifted without generating additional address logic. In the upcoming years, improvements have been reported by, e.g., Good et al. [15], Chi-Wu et al. [16], and Bulens et al. [17].

A very compact FPGA implementation of Grøstl has been presented by Jungk et al. [18,19,20] in 2010, 2011, and 2012. Their design needs 967 slices on a Spartan-3 FPGA without interface and 328 slices on a Virtex-6 FPGA requiring no Block RAMs. Sharif et al. [21] reported 1,627 slices (w/o Block RAMs) and 1,141 slices (using 18 Block RAMs) on a Virtex-5. In the same year, Kerckhof et al. [22] presented an implementation that needs only 343 slices on a Spartan-6 and 260 slices on a Virtex-6 FPGA (w/o using any Block RAMs or DSPs). At the final SHA-3 conference, Kaps et al. [23] published a lightweight Grøstl implementation requiring approx. 560 slices and one Block RAM on a Spartan-3 FPGA.

While there exist several papers that analyze the combination of different block ciphers and hash functions (mostly combining MD5 with SHA-1, e.g., [24,25,26,27]), there exists only one publication that focuses on the combination of AES and Grøstl on FPGA platforms. Järvinen [28] analyzed various resource-sharing techniques to reduce the area requirements for an Altera Cyclone III. Their smallest design needs 12,387 Logic Cells (LCs) whereas AES takes an overhead of about 2.5 %, i.e., 300 LCs.

3 Hardware Architecture of GrÆStl

GrÆStl has been designed with the aim for a very compact solution that supports both AES-128 and Grøstl-256 in one piece of silicon. We targeted a low-resource design (primarily area and power optimized) and applied different design techniques that are new or already known, e.g., from existing low-area AES implementations. Low-resource architectures are especially interesting for embedded systems such as contactless smart cards, sensor nodes, and RFID-based devices where area and power are stringent requirements for a practical deployment.

Next to the low-resource requirements, we aim for high flexibility so that the design can be applied on both ASIC and FPGA platforms. We therefore avoid the use of process-dependent technologies like RAM macros or the use of Block RAMs or DSPs on FPGA architectures. The disadvantage of these technologies are that they might not be available in all CMOS libraries and that they have to be recreated in case of a possible CMOS-process variation. Moreover, in case of FPGAs, resources such as Block RAMs or DSPs might be already used by

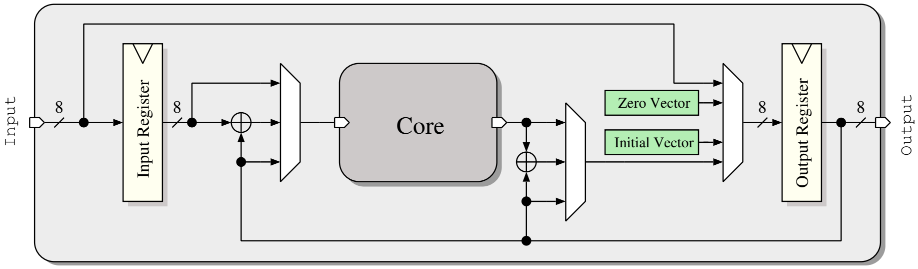


Fig. 1. Architecture of our 8-bit GrÆStl implementation

other system components such that the applicability of the design depends on their availability. In order to avoid those dependencies, we based our design only on standard cells and generic hardware components which makes it very flexible and portable to other platforms.

Since Grøstl needs per se more resources than AES, cf. [2], it is advisable to reuse existing hardware components to keep the overhead for AES as low as possible. We therefore decided to implement a combined datapath that effectively shares resources such as the S-box, temporary registers, or the *State* matrix. We based the top-module design and also the interface on an 8-bit architecture because it reduces the area complexity and power consumption of our design compared to larger datawidths. For the common datapath, we used 16 bits for Grøstl and only 8 bits for AES. This has two advantages. First, since two S-boxes are required for Grøstl, one S-box can be used in parallel to the SubBytes operation of AES to improve the performance of round-key generation. Second, parts of the computation can be switched off in order to reduce the overall power consumption.

An overview of the GrÆStl architecture is shown in Fig. 1. The main components are a common datapath (denoted as *Core* unit) that combines most of the round transformations for AES and Grøstl and two shared 512-bit shift registers. In order to keep the area requirements low, we decided to calculate the P and Q permutations of Grøstl sequentially instead of calculating them in parallel. This reduces the performance of hashing but allows to implement only one shared permutation instance in hardware. The need of two additional 512-bit shift registers is compensated by the fact that they are needed anyway in order to store the original message (needed for the second permutation Q), the result of the first permutation P and intermediate hash values in case of messages not fitting one block. These shift registers feature an 8-bit I/O through which large de-/multiplexers can be avoided. In the following, we present the common datapath and describe the implemented optimization techniques. Afterwards, we make use of the shared registers to improve the performance of the AES round-key generation.

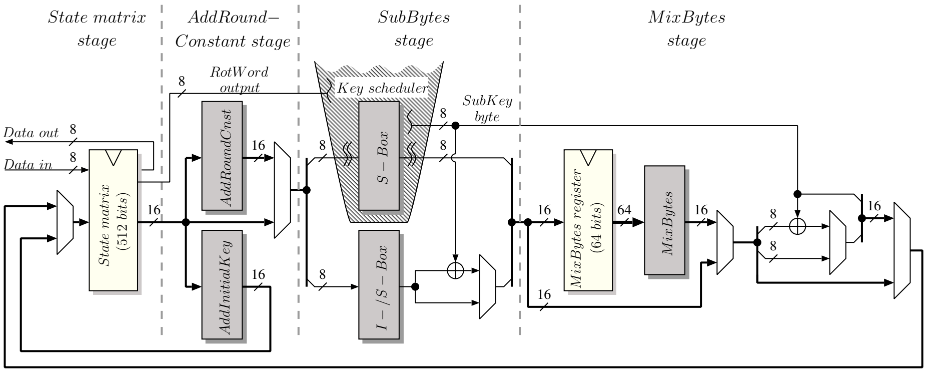


Fig. 2. Core datapath w/o round-key generation and un-/loading of the State matrix

3.1 The Common Datapath

Figure 2 shows the architecture of the common datapath. It has been separated into four main components according to the round transformations of AES and Grøstl: a shared State, an AddRoundConstant/AddInitialKey, SubBytes, and MixBytes unit.

Sharing the State. One of the most obvious ways to share resources between AES and Grøstl is to share the memory resources for the State. The size of the State for AES is 128 bits, i.e., a 4 × 4-byte matrix. Grøstl, in contrast, needs 512 bits (for variants returning a message digest of a size up to 256 bits), i.e., an 8 × 8-byte matrix. Thus, up to four AES States can fit into one Grøstl State which allows to integrate up to four AES encryption/decryption units in parallel to speed up the computation with minimal overhead. The work of Järvinen [28], for example, reported such an integration with an additional overhead of 13.5 % for four parallel AES encryptions and only 2.5 % for one AES encryption (neither of these two architectures contain the key generation). In contrast to our implementation, Järvinen targeted a high-throughput architecture with a datapath width of 512 bits.

For the design of GrÆStl, we decided to avoid any parallel computations to keep the area requirements as low as possible. Thus, we mapped one AES structure into the Grøstl State as illustrated in Fig. 3, i.e., the data (requiring the upper left 4 × 4 bytes) and the round key (requiring the lower left 4 × 4 bytes). In addition to these memory locations, we reused four bytes of the upper right 4 × 4 matrix as temporary registers for the round-key generation, further on denoted as *RotWord* shift-register. The round keys are then generated "on-the-fly" during AES computation.

The common State has been implemented using shift registers. This has several advantages. First, they reduce the area requirements on common FPGA platforms since the Look-Up Table (LUT) in certain logic blocks can be configured as

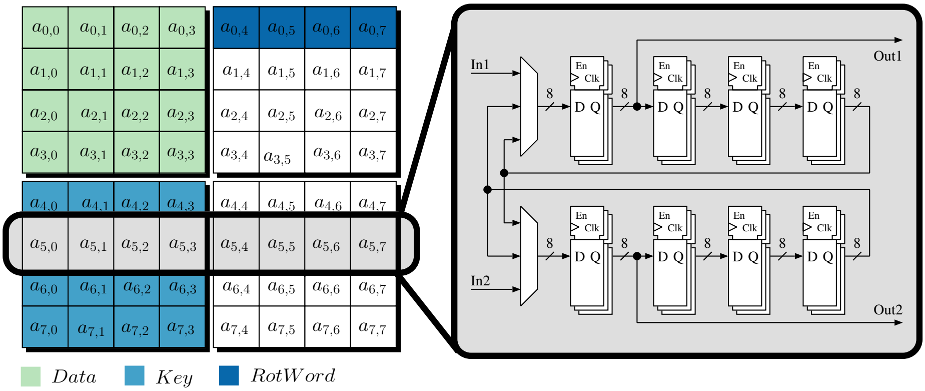

$a_{0,0}$ $a_{0,1}$ $a_{0,2}$ $a_{0,3}$ $a_{0,4}$ $a_{0,5}$ $a_{0,6}$ $a_{0,7}$
$a_{1,0}$ $a_{1,1}$ $a_{1,2}$ $a_{1,3}$ $a_{1,4}$ $a_{1,5}$ $a_{1,6}$ $a_{1,7}$
$a_{2,0}$ $a_{2,1}$ $a_{2,2}$ $a_{2,3}$ $a_{2,4}$ $a_{2,5}$ $a_{2,6}$ $a_{2,7}$
$a_{3,0}$ $a_{3,1}$ $a_{3,2}$ $a_{3,3}$ $a_{3,4}$ $a_{3,5}$ $a_{3,6}$ $a_{3,7}$
$a_{4,0}$ $a_{4,1}$ $a_{4,2}$ $a_{4,3}$ $a_{4,4}$ $a_{4,5}$ $a_{4,6}$ $a_{4,7}$
$a_{5,0}$ $a_{5,1}$ $a_{5,2}$ $a_{5,3}$ $a_{5,4}$ $a_{5,5}$ $a_{5,6}$ $a_{5,7}$
$a_{6,0}$ $a_{6,1}$ $a_{6,2}$ $a_{6,3}$ $a_{6,4}$ $a_{6,5}$ $a_{6,6}$ $a_{6,7}$
$a_{7,0}$ $a_{7,1}$ $a_{7,2}$ $a_{7,3}$ $a_{7,4}$ $a_{7,5}$ $a_{7,6}$ $a_{7,7}$
In1
In2
Out1
Out2
En
Clk
D Q
8
🟩 *Data*
🟦 *Key*
🟦 *RotWord*

Fig. 3. Mapping the AES State into the Grøstl State and the construction of a single State matrix row

a shift register without using the flip-flops available in each slice as also noticed by Chodowiec et al. [14]. Second, they are very flexible and can be used for both ASIC and FPGA designs as opposed to other memory architectures such as RAM macros or Block RAMs. Third, due to automatic shifts of intermediate values, additional address logic and multiplexer stages can be avoided. Thus, no ShiftRows or ShiftBytes units are needed because they are implicitly performed by the applied shift registers.

Each row in the State has been implemented as an 8-byte shift register that can be split into two 4-byte shift registers with two independent inputs. Figure 3 shows one internal row composed of two 4-byte shift registers. When AES is performed, only 4-byte shift registers are used, 8-byte shift registers are used during Grøstl computations. In order to reduce the power consumption during AES computation, we applied an operand-isolation technique that switches off unused parts of the matrices, e.g., the lower right 4×4 bytes of the State matrix. Furthermore, we applied clock gating cells to minimize toggling activity.

Tweaked AddRoundConstant Stage. For Grøstl, the State is modified through the AddRoundConstant function, which varies with the type of the required permutation. In case of the P permutation, the first row gets modified through fixed constants whereas the rest stays untouched. This changes for the Q permutation, where instead of the first row, the last row gets modified. Additionally, the rest of the State bits get flipped, cf. [2]. Note that in contrast to the Grøstl-0 specification (from the 31th of October 2008), where only a single byte gets modified (for both permutations), in the tweaked Grøstl version (from the 2nd of March 2011, version 2.0.1) multiple bytes get modified. Compared to existing work, which mostly presents solutions for Grøstl-0, e.g., in [10], we present an implementation that considers the tweaked variant including the modified round constants and initial vectors.

For AES, the State gets modified through the AddRoundKey function, which simply adds (XOR-operation) the round key located in the lower left 4 × 4 bytes of the State matrix to the data in the upper left 4 × 4 bytes.

Reusing SubBytes for AES Round-Key Generation. The SubBytes transformation in Grøstl can be efficiently combined with the S-box operation of AES, because both algorithms make use of the same S-box transformation. Minor effort has to be made in order to provide the inverse S-box transformation required for AES decryption.

There exist several implementation optimizations for the AES S-box, e.g., given in [4], [29], or [30]. Most of the related work transformed the finite-field operations over $GF(2^8)$ into a composite of smaller fields, e.g., $GF((2^4)^2)$. We implemented the method proposed by Wolkerstorfer et al. [30], where an S-box is composed of two transformations, namely the calculation of a multiplicative inverse in the finite field $GF(2^8)$ and an affine transformation. For AES decryption, the affine transformation is exchanged with its counterpart and executed before computing the multiplicative inverse. In order to save some additional gates, one may replace the S-box implementation by Wolkerstorfer et al. with the one by Canright [4]. As our Grøstl version is based on a 16-bit wide datapath, we implemented two S-boxes, one of them providing both transformation directions. AES is based on an 8-bit datapath, thus we exploited the presence of an additional S-box in order to improve the performance of the AES round-key generation as described in the following.

Basically, there exist two possibilities to generate the round keys for AES encryption and decryption. First, the round keys are pre-computed and stored in non-volatile memory. Second, the round keys are computed "on-the-fly". While the first option provides fast access to existing round keys, the second option is cheaper in terms of area requirements since no memory is needed to store the keys. We therefore decided to implement the second option.

While one S-box is used to perform the SubBytes operation of AES, the second S-box can be reused to calculate the round keys in parallel. For the round-key generation, one S-box, XOR operations, and a small LUT is required that holds the round constants. Figure 5 and Fig. 6 in Appendix A illustrate the general forward and backward round-key generation.

The forward round-key generation is done as follows. First, during the initialization of the common State, the last four bytes of the master key are loaded into the *RotWord* shift-register (located in the upper right 4 × 4 matrix as shown in Fig. 3). The output of the *RotWord* shift-register gets substituted by the shared S-box and modified with the round-dependent constant *Rcon* before it gets added to the output of the first row of the key matrix (located in the lower left 4 × 4 matrix as shown in Fig. 3). Afterwards, the result is loaded back into the *RotWord* shift-register and the first row of the key matrix before both get shifted. This is done for the first byte of each row of the key matrix in order to obtain the highest four bytes of the next round key. The following three columns of the next round key get calculated by applying the same procedure while bypassing the S-box and *Rcon* modifications. Due to the similarity of the forward and backward round-key generation, we do not explain the latter in detail.

Combined MixColumns and MixBytes. MixColumns and MixBytes have been combined to a common MixBytes function. It has been implemented such that it takes the 64-bit output of the *MixBytesReg* as input, representing either a column of AES or a column of Grøstl. As for AES one column only has a size of 32 bits, the remaining 32 bits are not used. For both AES and Grøstl the matrices, which have to be multiplied with the State, are circulant and constant. This fact helps in reducing the complexity by implementing from each of the three matrices only one row. The circulant behavior is gained through the *MixBytesReg*, realized as an 8-byte shift register. In the next four clock cycles after this register has been loaded, always 8 bits for AES respectively 16 bits for Grøstl are processed and stored in the State matrix. Note that the MixColumns operation for AES encryption comes for free as it is implicitly computed during a Grøstl MixBytes operation.

Furthermore, the MixColumns operation for AES decryption takes larger coefficients than those required for Grøstl. Therefore, additional effort has been made to provide also the inverse operation of the MixColumns function.

3.2 I/O-Register Sharing

Since two 512-bit registers are required for our Grøstl implementation in order to hold intermediate values and the original input message during the sequential execution of permutation P and Q, we can reuse them to improve the performance of AES decryption as follows. As soon as an AES encryption or decryption finishes, we store its master key together with the corresponding last round key in the output register. If afterwards another decryption takes place, we compare its master key with the previously stored one. In case the keys match, we reuse the previously stored last round key instead of the new master key and can therefore save the time (i.e., 330 clock cycles) for calculating the last round key required for decryption.

4 Results

We implemented GrÆStl-256 in VHDL and synthesized it for both ASIC and FPGA platforms. For the ASIC design, we used *Mentor Graphics ModelSim 6.5c* and *Synopsys DesignCompiler 2010.03* for functional simulations (RTL and post-layout verification) and synthesis. We mapped our architecture onto a standard-cell library based on the $0.18\,\mu m$ CMOS process by UMC. One single Gate Equivalent (GE) therefore corresponds to the area of a 2-input NAND gate, i.e., $9.3744\,\mu m^2$. For FPGA synthesis, we used *Xilinx ISE Design Suite 12.1* and applied the parameter set “Area Reduction with Physical Synthesis”. Furthermore, we removed the reset feature from all shift registers as sub-optimal reset strategies can prevent the use of device-library components, such as shift register look-up tables (SRLs) [31].

Table 1. ASIC area results after synthesis

Component	**AES** [GE]	**Grøstl** [GE]	**GrÆStl** [GE]
Top Level Glue Logic	50	460	800
Input/Output Reg.	-	8,250	8,250
Core	2,550	6,340	7,500
State Matrix	1,100	4,200	4,350
AddRndCnst.	-	100	100
SubBytes	330	540	600
MixBytes	490	900	1,100
Core Glue Logic	630	600	1,350
Round-Key Gen.	2,150	-	-
Overall	4,750	15,050	16,550

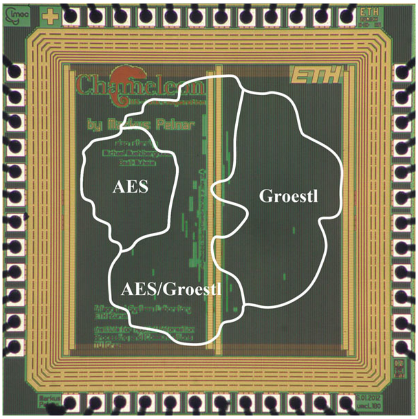


Fig. 4. Chip layout

In order to evaluate the efficiency of GrÆStl compared to separate implementations, we provide results for stand-alone variants of AES and Grøstl. However, we have to note that a fair comparison of our stand-alone variants with related work is largely infeasible since we targeted a combined version and integrated optimization techniques that do not affect the single-implementation variants. Table 1 lists the area occupation for AES, Grøstl, and GrÆStl for a target frequency of 100 MHz. It shows that GrÆStl needs only 16,550 GEs of area not including an interface. This is about 16 % less than the sum of the two separate implementations. Furthermore, it shows that the overhead for AES is only 1.5 kGEs which is smaller than the smallest available stand-alone AES implementation given in the literature, cf. [5,6].

In view of execution time, our stand-alone version of AES and GrÆStl need 945 clock cycles for encryption. This includes an 8-bit four-phase-handshaking that requires 203 clock cycles to load/unload the input/output register. Decryption needs 1,558 clock cycles. Our GrÆStl implementation even reduces the number of clock cycles by 330 because the round-key generation can be avoided if the last round key is already maintained in the I/O registers. Hashing of a 512-bit block takes 3,465 clock cycles (including 404 clock cycles for the interface) without applying message padding. The number of clock cycles results from the State matrix design, the datapath width of 8 bits and the required *MixBytes-Reg* through which the datapath is split into halves. In short, this register gets filled in each cycle with one valid byte related to a certain column of AES or Grøstl. After finishing loading, the State matrix gets updated through the MixBytes function. Afterwards the State matrix is shifted by one column and the procedure starts over again.

We taped out our design as an ASIC. For this, we targeted a maximum clock frequency of 125 MHz. This in combination with clock gating and eight scanchains required an additional amount of 750 GEs of chip area. A photo of the fabricated chip, highlighting the floorplan of the three different modules, is

Table 2. ASIC comparison of AES-128 (incl. decryption) and Grøstl-256

Source	**Width**	**Techn.**	**f$_{\mathbf{max}}$**	**Cycles/Block**			**Power**	**Area**	
	[bits]	[μm]	[MHz]	Enc.	Dec.	Hash	[$\mu W/MHz$]	[kGE]	
Hämäläinen [6]	8	0.13	153	160	n/a	-	37 @ 1.2 V	3.9[1]	AES
Feldhofer [5]	8	0.35	80	1,032	1,165	-	45 @ 1.5 V	3.4	
Ours - AES	8	0.18	100	742	1,025	-	130 @ 1.8 V	4.75	
Tillich [10]	64	0.35	56	-	-	196	2,210 @ 3.3 V	14.6	Grøstl
Ours - Grøstl	16	0.18	100	-	-	3,093	200 @ 1.8 V	15.05	
Ours - GrÆStl	8/16	0.18	100	742	1,025	3,093	200 @ 1.8 V	16.55	GrÆStl

[1] The area for the design, including the decryption has been estimated with additional 25 % of the original (encryption only) AES design.

illustrated in Fig. 4. To the best of our knowledge, there is no ASIC design of Grøstl (or a combination of AES and Grøstl) available so far, which targets a low-resource implementation and has finally been taped out.

Table 2 gives a comparison with related work. While our stand-alone implementation of AES is about 1 kGE larger than existing work [5,6], only a small overhead is required for Grøstl to support also AES, i.e., 10 %. Moreover, our GrÆStl implementation is only slightly larger than the work by Tillich et al. [10] which can be attributed to the design decisions to support both Grøstl (tweaked version instead of Grøstl-0) and AES which requires additional area to optimally merge both algorithms (to keep the overall area as low as possible). Regarding power consumption, it shows that our design meets most requirements even for a contactless operation, e.g., for contactless smart cards. However, due to different fabrication technologies and supply voltages, we cannot fairly compare the given numbers.

4.1 FPGA Results

We used Xilinx Spartan-3 and Virtex-6 FPGAs to evaluate the performance of our design on reconfigurable hardware. Table 3 shows the results after place-and-route and without an interface included. On the Spartan-3, our design needs only 956 slices. On the Virtex-6, 302 slices are needed. The sum of slices required by the independent Spartan-3 AES/Grøstl implementations is smaller than those required for the GrÆStl design. This is due to the structure of the State matrix of the inner 8/16-bit datapath. For the Grøstl design, this State matrix is built upon eight 8-byte shift registers which perfectly fit the SRL-16 mode (provided by Xilinx) as such a 8-byte shift register can be achieved within only a single CLB. For GrÆStl, each row of the State matrix is replaced by two independent 4-byte shift registers since AES operates on a smaller State compared to Grøstl. Because of the 4-byte shift registers and the control logic required to form a 8-byte shift register, it is not possible for the compiler to fit the two 4-byte

Table 3. Post place-and-route results for various Xilinx FPGAs

	Spartan-3			Virtex-6		
	AES	**Grøstl**	**GrÆStl**	**AES**	**Grøstl**	**GrÆStl**
Number of slice LUTs	838	896	1,805	455	531	1,046
Number used as logic.	788	743	1,554	419	443	927
Number used as shift registers	48	128	192	24	64	96
Number used as a route-thru	2	25	59	12	24	23
Number of slice registers	48	293	407	169	279	360
Frequency(MHz)	35	40	30	110	115	80
Number of occupied slices	442	488	956	142	202	302

Table 4. FPGA comparison of AES-128 (incl. decryption) and Grøstl-256

Source	**Width**	**Output**	**Device**	$\mathbf{f_{max}}$	**Cyc./Block**			**Area**	
	[bits]	[bits]		[MHz]	Enc.	Dec.	Hash	[Slices]	
Bulens [17]	128	128	Spartan-3	150	12	12	-	2,150	AES
Ours - AES	8	128	Spartan-3	35	742	1,025[1]	-	442	
Bulens [17]	128	128	Virtex-5	350	11	11	-	550	
Ours - AES	8	128	Virtex-6	110	742	1,025[1]	-	142	
Jungk [19]	64	224/256	Spartan-3	182	-	-	160	967	Grøstl
Ours - Grøstl	8/16	256	Spartan-3	40	-	-	3,093	488	
Kerckh. [22]	64	256	Virtex-6	280	-	-	450	260	
Ours - Grøstl	8/16	256	Virtex-6	115	-	-	3,093	202	
Ours - GrÆStl	8/16	256	Spartan-3	30	742	1,025[1]	3,093	956	GrÆStl
Ours - GrÆStl	8/16	256	Virtex-6	80	742	1,025[1]	3,093	302	

[1] Decryption with stored last round key.

shift registers into one CLB. Particular changes to the design, which target a certain FPGA device (i.e., a more *target-oriented mapping* of the shift registers) may lead to better results for all three designs. However, this would lead to a platform-dependent architecture what would contradict with one of our main goals, i.e., to stay platform independent.

Table 4 shows a comparison with related work. Note that we listed only AES implementations that require no Block RAMs, DSPs, etc. to allow a fair comparison. It shows that our stand-alone AES implementation is the smallest on the Spartan-3 needing only 442 slices. For the Virtex-6, our design needs only 142 slices. Our Grøstl implementation needs 488 slices on the Spartan-3 and is therefore about 2 times smaller than the work of Jungk et al. [19] which requires 967 slices. Compared to the work of Kerckhof et al. [22], we need only 202 slices on the Virtex-6 instead of 260, i.e., a factor of about 1.3.

5 Conclusion

In this paper, we presented GrÆStl—a combined hardware implementation of AES-128 and Grøstl-256. GrÆStl has been designed for embedded systems and therefore shares resources as much as possible to lower the area requirements. We integrated the following optimization techniques: (1) we mapped the AES State into the GrÆStl State matrix to avoid the need of additional memory, (2) we made use of shift registers to provide high flexibility (for ASICs as well as FPGAs) and avoid the implementation of ShiftBytes and ShiftRows, (3) we implemented the tweaked AddRoundConstant function instead of Grøstl-0 as given in related work, (4) we reused the S-box for AES and Grøstl and reused it also to increase the performance of AES round-key generation, (5) we combined MixBytes and MixColumns, and finally (6) we proposed to share the I/O registers which avoids forward round-key generation during decryption which helps to reduce 330 clock cycles in addition.

As result, GrÆStl is the first combined hardware implementation fabricated as ASIC and occupies 16.55 kGEs in total whereas AES needs an overhead of only 10 %. In particular, on a Spartan-3 FPGA, our stand-alone AES and Grøstl implementations outperform existing solutions by a factor of 4.8 and 2 in terms of area. The small area requirements and the low-power consumption of about $20\,\mu W$ at 100 kHz make the design right suitable for low-resource devices such as contact-less smart cards and Radio Frequency (RF) communication based devices. Besides that, GrÆStl might be also considered to implement a low-resource authenticated-encryption scheme, since it provides the required cryptographic primitives in a single architecture.

Acknowledgment. The authors would like to thank the team from the Microelectronics Design Center of ETH Zurich, namely Beat Muheim and Frank Gürkaynak, for their constant support during the backend design of the ASIC. We also thank Martin Schläffer for fruitful discussions. This work has been supported by the European Commission through the ICT program under contract ICTSEC- 2009-5-258754 (Tamper Resistant Sensor Node - TAMPRES) and ICT-2007-216646 (European Network of Excellence in Cryptology - ECRYPT II).

References

1. NIST: Advanced Encryption Standard (AES) (FIPS PUB 197). National Institute of Standards and Technology (November 2001)
2. Gauravaram, P., Knudsen, L.R., Matusiewicz, K., Mendel, F., Rechberger, C., Schläffer, M., Thomsen, S.S.: Grøstl – a SHA-3 candidate. Submission to NIST, Round 3 (2011)
3. Bertoni, G., Daemen, J., Peeters, M., Van Assche, G.: The Keccak SHA-3 submission. Submission to NIST, Round 3 (2011)
4. Canright, D.: A Very Compact S-Box for AES. In: Rao, J.R., Sunar, B. (eds.) CHES 2005. LNCS, vol. 3659, pp. 441–455. Springer, Heidelberg (2005)

5. Feldhofer, M., Wolkerstorfer, J., Rijmen, V.: AES implementation on a grain of sand. IEE Proceedings - Information Security 152(1), 13–20 (2005)
6. Hamalainen, P., Alho, T., Hannikainen, M., Hamalainen, T.D.: Design and Implementation of Low-Area and Low-Power AES Encryption Hardware Core. In: Proceedings of the 9th EUROMICRO Conference on Digital System Design, DSD 2006, pp. 577–583. IEEE Computer Society, Washington, DC (2006)
7. Kaps, J.-P., Sunar, B.: Energy Comparison of AES and SHA-1 for Ubiquitous Computing. In: Zhou, X., Sokolsky, O., Yan, L., Jung, E.-S., Shao, Z., Mu, Y., Lee, D.C., Kim, D.Y., Jeong, Y.-S., Xu, C.-Z. (eds.) EUC Workshops 2006. LNCS, vol. 4097, pp. 372–381. Springer, Heidelberg (2006)
8. Kim, M., Ryou, J., Choi, Y., Jun, S.: Low Power AES Hardware Architecture for Radio Frequency Identification. In: Yoshiura, H., Sakurai, K., Rannenberg, K., Murayama, Y., Kawamura, S. (eds.) IWSEC 2006. LNCS, vol. 4266, pp. 353–363. Springer, Heidelberg (2006)
9. Moradi, A., Poschmann, A., Ling, S., Paar, C., Wang, H.: Pushing the Limits: A Very Compact and a Threshold Implementation of AES. In: Paterson, K.G. (ed.) EUROCRYPT 2011. LNCS, vol. 6632, pp. 69–88. Springer, Heidelberg (2011)
10. Tillich, S., Feldhofer, M., Issovits, W., Kern, T., Kureck, H., Mühlberghuber, M., Neubauer, G., Reiter, A., Köfler, A., Mayrhofer, M.: Compact Hardware Implemenations of the SHA-3 Candidates ARIRANG, BLAKE, Grøstl and Skein. In: Auer, M., Pribyl, W., Söser, P. (eds.) Proceedings of Austrochip 2009, Graz, Austria, October 7, pp. 69–74 (2009)
11. Katashita, T.: Grøstl Compact (August 2012), http://www.morita-tech.co.jp/SASEBO/en/sha3/implement.html
12. Guo, X., Huang, S., Nazhandali, L., Schaumont, P.: Fair and Comprehensive Performance Evaluation of 14 Second Round SHA-3 ASIC Implementations. In: Second SHA-3 Candidate Conference (2010)
13. Henzen, L., Gendotti, P., Guillet, P., Pargaetzi, E., Zoller, M., Gürkaynak, F.K.: Developing a Hardware Evaluation Method for SHA-3 Candidates. In: Mangard, S., Standaert, F.-X. (eds.) CHES 2010. LNCS, vol. 6225, pp. 248–263. Springer, Heidelberg (2010)
14. Chodowiec, P., Gaj, K.: Very Compact FPGA Implementation of the AES Algorithm. In: Walter, C.D., Koç, Ç.K., Paar, C. (eds.) CHES 2003. LNCS, vol. 2779, pp. 319–333. Springer, Heidelberg (2003)
15. Good, T., Benaissa, M.: AES on FPGA from the Fastest to the Smallest. In: Rao, J.R., Sunar, B. (eds.) CHES 2005. LNCS, vol. 3659, pp. 427–440. Springer, Heidelberg (2005)
16. Huang, C.W., Chang, C.J., Lin, M.Y., Tai, H.Y.: Compact FPGA Implementation of 32-bits AES Algorithm Using Block RAM. In: TENCON 2007 - 2007 IEEE Region 10 Conference, October 30-November 2, pp. 1–4 (2007)
17. Bulens, P., Standaert, F.-X., Quisquater, J.-J., Pellegrin, P., Rouvroy, G.: Implementation of the AES-128 on Virtex-5 FPGAs. In: Vaudenay, S. (ed.) AFRICACRYPT 2008. LNCS, vol. 5023, pp. 16–26. Springer, Heidelberg (2008)
18. Jungk, B., Apfelbeck, J.: Area-Efficient FPGA Implementations of the SHA-3 Finalists. In: Athanas, P.M., Becker, J., Cumplido, R. (eds.) ReConFig, pp. 235–241. IEEE Computer Society (2011)
19. Jungk, B., Reith, S.: On FPGA-Based Implementations of the SHA-3 Candidate Grøstl. In: 2010 International Conference on Reconfigurable Computing and FPGAs (ReConFig), pp. 316–321 (December 2010)

20. Jungk, B.: Evaluation of Compact FPGA Implementations for All SHA-3 Finalists. In: SHA-3 Conference (March 2012)
21. Sharif, M.U., Shahid, R., Rogawski, M., Gaj, K.: Use of Embedded FPGA Resources in Implementations of Five Round Three SHA-3 Candidates. In: CRYPT II Hash Workshop 2011 (2011)
22. Kerckhof, S., Durvaux, F., Veyrat-Charvillon, N., Regazzoni, F., de Dormale, G.M., Standaert, F.-X.: Compact FPGA Implementations of the Five SHA-3 Finalists. In: Prouff, E. (ed.) CARDIS 2011. LNCS, vol. 7079, pp. 217–233. Springer, Heidelberg (2011)
23. Kaps, J.P., Yalla, P., Surapathi, K.K., Habib, B., Vadlamudi, S., Gurung, S.: Lightweight Implementations of SHA-3 Finalists on FPGAs. In: SHA-3 Conference (March 2012)
24. Cao, D., Han, J., Yang Zeng, X.: A Reconfigurable and Ultra Low-Cost VLSI Implementation of SHA-1 and MD5 Functions. In: 7th International Conference on ASIC Proceeding – ICASIC 2007, Guilin, China, October 25-29, pp. 862–865. IEEE (2007)
25. Ganesh, T.S., Sudarshan, T.S.B.: ASIC Implementation of a Unified Hardware Architecture for Non-Key Based Cryptographic Hash Primitives. In: Proceedings of the International Conference on Information Technology: Coding and Computing (ITCC 2005), Las Vegas, Nevada, USA, April 4-6, vol. 1, pp. 580–585. IEEE Computer Society (2005)
26. Järvinen, K.U., Tommiska, M., Skyttä, J.: A Compact MD5 and SHA-1 Co-Implementation Utilizing Algorithm Similarities. In: International Conference on Engineering of Reconfigurable Systems and Algorithms – ERSA 2005, Las Vegas, Nevada, USA, June 27-30, pp. 48–54. CSREA Press (2005)
27. Wang, M.Y., Su, C.P., Huang, C.T., Wu, C.W.: An HMAC Processor with Integrated SHA-1 and MD5 Algorithms. In: Imai, M. (ed.) Proceedings of the Conference on Asia South Pacific Design Automation: Electronic Design and Solution Fair 2004 (ASP-DAC), Yokohama, Japan, January 27-30, pp. 456–458. IEEE (2004)
28. Järvinen, K.: Sharing Resources Between AES and the SHA-3 Second Round Candidates Fugue and Grøstl. In: Second SHA-3 Candidate Conference (August 2010)
29. Nikova, S., Rijmen, V., Schläffer, M.: Using Normal Bases for Compact Hardware Implementations of the AES S-box. In: 6th International Conference Security in Communication Networks (SCN)
30. Wolkerstorfer, J., Oswald, E., Lamberger, M.: An ASIC Implementation of the AES SBoxes. In: Preneel, B. (ed.) CT-RSA 2002. LNCS, vol. 2271, pp. 67–78. Springer, Heidelberg (2002)
31. Xilinx: HDL Coding Practices to Accelerate Design Performance (May 2012), http://www.xilinx.com/support/documentation/white_papers/wp231.pdf

A AES-128 Key Generation

Figure 5 and Fig. 6 illustrate the general forward and backward round-key generation of AES-128. Each of the round key words is stored in a single column and updating the keys is done column by column.

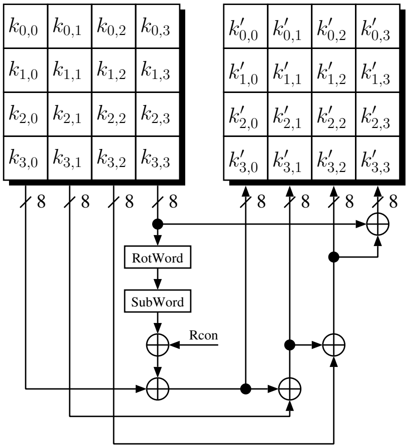

$k_{0,0}$ $k_{0,1}$ $k_{0,2}$ $k_{0,3}$
$k_{1,0}$ $k_{1,1}$ $k_{1,2}$ $k_{1,3}$
$k_{2,0}$ $k_{2,1}$ $k_{2,2}$ $k_{2,3}$
$k_{3,0}$ $k_{3,1}$ $k_{3,2}$ $k_{3,3}$
$k'_{0,0}$ $k'_{0,1}$ $k'_{0,2}$ $k'_{0,3}$
$k'_{1,0}$ $k'_{1,1}$ $k'_{1,2}$ $k'_{1,3}$
$k'_{2,0}$ $k'_{2,1}$ $k'_{2,2}$ $k'_{2,3}$
$k'_{3,0}$ $k'_{3,1}$ $k'_{3,2}$ $k'_{3,3}$
8
RotWord
SubWord
Rcon

Fig. 5. Forward round-key generation

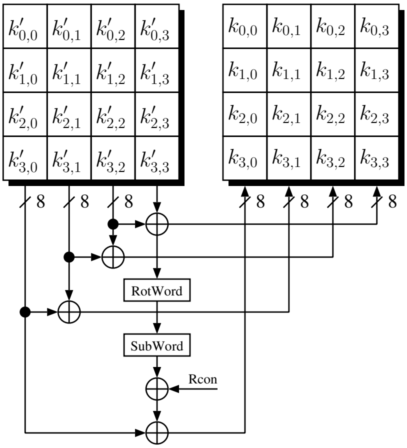

$k'_{0,0}$ $k'_{0,1}$ $k'_{0,2}$ $k'_{0,3}$
$k'_{1,0}$ $k'_{1,1}$ $k'_{1,2}$ $k'_{1,3}$
$k'_{2,0}$ $k'_{2,1}$ $k'_{2,2}$ $k'_{2,3}$
$k'_{3,0}$ $k'_{3,1}$ $k'_{3,2}$ $k'_{3,3}$
$k_{0,0}$ $k_{0,1}$ $k_{0,2}$ $k_{0,3}$
$k_{1,0}$ $k_{1,1}$ $k_{1,2}$ $k_{1,3}$
$k_{2,0}$ $k_{2,1}$ $k_{2,2}$ $k_{2,3}$
$k_{3,0}$ $k_{3,1}$ $k_{3,2}$ $k_{3,3}$
8
RotWord
SubWord
Rcon

Fig. 6. Backward round-key generation

Memory Access Pattern Protection for Resource-Constrained Devices

Yuto Nakano[1], Carlos Cid[2], Shinsaku Kiyomoto[1], and Yutaka Miyake[1]

[1] KDDI R&D Laboratories Inc.
2-1-15 Ohara, Fujimino, Saitama 356-8502, Japan
{yuto,kiyomoto,miyake}@kddilabs.jp

[2] Information Security Group, Royal Holloway, University of London
Egham, TW20 0EX, UK
Carlos.Cid@rhul.ac.uk

Abstract. We propose a practice-oriented scheme for protecting RAM access pattern. We first consider an instance which relies on the use of a secure (trusted) hardware buffer; it achieves both security and performance levels acceptable in practice by adapting ideas from oblivious RAM mechanisms, yet without the expensive (re-)shuffling of buffers. Another instance requires no special hardware, but as a result leads to a higher, yet practical overhead. One of the main features of the proposal is to maintain the *history* of memory access to help hiding the access pattern. We claim that under reasonable assumptions, the first scheme with trusted memory is secure with overhead of only $6\times$, as is the second scheme with overhead of $(2m+2\ell_h+2)\times$ where m and ℓ_h are respectively the size of the buffer and history. We note that although the proposal is particularly focused on the software execution protection environment, its security may well be appropriate for most uses in the remote storage environment, to prevent access pattern leakage of cloud storage with much lower performance overhead than existing solutions.

Keywords: Access Pattern Protection, Oblivious RAM, Shuffle Buffer.

1 Introduction

The problem of preventing leakage of information arising from both running software on untrusted systems, as well as storing data in remote untrusted servers has attracted much attention. While encryption can be used to protect data confidentiality, the problem of protecting access pattern with manageable and practical overhead is harder to address.

In the software execution environment, one can identify two main motivations for protecting memory access pattern privacy: the traditional application is to protect Intellectual Property, and prevent software piracy. More recently, it has been shown how access pattern leakage can be used to attack certain implementations of cryptographic algorithms (in the so-called *cache attacks* [15]). In the remote storage environment, one would like to protect access pattern from a

S. Mangard (Ed.): CARDIS 2012, LNCS 7771, pp. 188–202, 2013.

curious but not malicious server, which may benefit from gaining information about the client's pattern of access of stored data.

The traditional solution for memory access pattern protection is known as *Oblivious RAM.* It was first proposed by Goldreich [5], and later extended by Goldreich and Ostrovsky [6]. The main construction is based on the *hierarchical solution*, in which the data structure is organised in levels consisting of hash-tables (using secret hash functions known by the client only), and requires periodic expensive *oblivious* re-shuffling of data.

In the past few years, many improvements have been proposed, for example [1,16,4,9,17,12]. Several schemes consider improvements for the application to cloud computing [3,19,7,8,10,11,13,20]. Improvements typically arise from the use of different data structures and hash function schemes, more efficient sorting algorithms (for the oblivious shuffling step), and the use of secure local (client) memory. Besides the hierarchical solution, Goldreich and Ostrovsky [6] also proposed the *square root* construction, which uses secret random permutations when storing data. Boneh *et al.* [2] have recently presented a hybrid algorithm between the square-root and hierarchical algorithms, and proposed a new notion of *oblivious storage.*

Despite much recent progress, where both the asymptotic efficiency as well as the constant terms of oblivious RAM solutions have been improved (making it particularly attractive for remote storage access pattern protection), current solutions remain inefficient for *software execution protection*, i.e. to prevent leakage of relatively limited-in-size memory access pattern. In these cases, the constant terms involved in the computational complexity make the overhead unacceptably high. Yet an efficient and secure mechanism for access pattern protection is a particularly desirable feature in this environment. For instance, an emerging issue is the rapid increase of malicious software targeting smartphones. Most existing protection schemes, originally designed for PCs, are not suitable for smartphones due to several limitations, such as the computational power and available storage size. In these environments, solutions range from use of obfuscation to hardware-based access pattern protection mechanisms. For instance, Zhuang *et al.* [21] proposed a practical, hardware-assisted scheme for embedded processors, with low computational overhead. Their *control flow obfuscation* scheme for embedded processors employs a small secure hardware obfuscator to hide program recurrence. Their proposal however trades security for low overhead (and the cost of the trusted hardware buffer), and in some situations an adversary with access to the device can retrieve information about memory access.

In this paper, we propose a practice-oriented scheme for protecting RAM access pattern. We first consider an instance which, similar to the proposal by Zhuang *et al.*, also relies on the use of a secure (trusted) hardware buffer. However it achieves higher security by adapting ideas from Goldreich and Ostrovsky's square root solution, yet without the expensive (re-)shuffling of buffers. By applying this scheme, we can construct a secure platform that is suitable for executing software that deals with user private information. A potential application is to secure program execution in smartphones: these devices typically contain sensitive user information and are increasingly under severe threat from malware. Most smartphones have a SIM card, which is generally considered as a secure area, and deployment and security of our scheme could rely on the SIM card. Another instance requires no special hardware, but as a result leads to a higher, yet practical overhead. This scheme can offer the same level of security without any special hardware. Many applications have their security relying on IC cards or other trusted hardware. One of the roles of trusted hardware is offering a secure computation, which can be realised with our proposal. Another role is a secure storage for a secret information, which is yet to be realised.

The main feature of our proposal is to maintain the *history* of memory access, which together with the access of dummy data, helps one to hide data access pattern. The security of the schemes depends on the size of the buffer (as cache or in RAM) and how the history is used. We claim that under reasonable assumptions and by selecting appropriate parameters, the schemes achieve both security and performance levels acceptable in practice. We note that although the proposal is particularly focused on the software execution protection environment (i.e. to prevent RAM access pattern leakage), its security may well be appropriate for most uses in the remote storage environment, to prevent access pattern leakage of cloud storage with much lower performance overhead than existing solutions.

2 Access Pattern Protection Problem

We can model the problem of access pattern protection as follows. We consider a *client*, with potentially small secure memory, and a *server* providing large insecure storage. This storage consists of several data blocks (for simplicity, all of the same size). In the software execution environment, we can think of the CPU as the client, and RAM as the server, where a malicious entity (e.g. malware) has access to RAM and the memory bus used in the communication between the CPU and memory.

The client accesses data by making requests to the server to either retrieve the contents of a particular data block at location i or by writing x into a data block at location j. We denote these operations by $\texttt{read}(i)$ and $\texttt{write}(j,\ x)$, respectively. We consider the security goals as: to protect the confidentiality[1] of data, as well as hide the client's access pattern to the stored data blocks from a computationally bounded adversary, which can access the data storage and the communication channel between the client and the server.

We can address the first goal by using a semantically secure probabilistic encryption scheme, such that two encryptions of the same data block will look indistinguishable to a computationally bounded adversary. The use of encryption adds a constant overhead to the system. For the remaining of this work, we will

[1] In environments where there is a requirement, we will also want to protect data integrity; this can be done by adding a MAC or by using an authenticated encryption algorithm.

assume that the data is stored encrypted; it is decrypted whenever it is read from memory, and re-encrypted whenever it is written into memory.

For the second goal, we would like that a particular sequence of operations in the stored data does not substantially leak any information, other than how many data blocks were accessed. To formalise it, we use the definition in [16] for a *secure oblivious RAM* system.

Definition 1 ([16]). *The input y of the client is a sequence of data blocks, denoted by $((v_1, x_1), \ldots, (v_n, x_n))$ and a corresponding sequence of operations, denoted by $(op_1, \ldots, op_m)$, where each operation is either a read operation, denoted* `read(v)`*, which retrieves the data of the block indexed by v, or a write operation, denoted* `write(v, x)`*, which sets the value of block v to be equal to x.*

The access pattern $A(y)$ is the sequence of accesses to the remote storage system. It contains both the indices accessed in the system and the data blocks read or written. An oblivious RAM system is considered secure if for any two inputs y and y' of the client, of equal length, the access patterns $A(y)$ and $A(y')$ are computationally indistinguishable for anyone but the client.

The typical, straightforward method for preventing an adversary from distinguishing between the `read` and `write` operations is to always perform both operations in every access. As a result of using this method, an access pattern $A(y)$ (for the purpose of indistinguishability) can be thought as simply as a sequence of indices i (corresponding to the data blocks accessed). The trivial solution to the access pattern protection problem consists of accessing all data blocks on the memory for each query. Another trivial solution is to use a secure client hardware. Both schemes are however too costly in practice.

In addition, data blocks are typically organised in memory based on a secret permutation or hash function in an oblivious way. This is the most expensive component of the schemes, and is the main responsible for the (amortised) computational overhead. Our proposal does not employ oblivious re-shuffling of memory; while this will affect the security provided by the scheme, we claim that under reasonable assumptions, the proposals achieve both security and performance levels acceptable in practice.

3 Related Work

We briefly introduce below some of the main research results related to our work.

3.1 Oblivious RAM

Oblivious RAM is the traditional solution for memory access pattern protection, having been first proposed by Goldreich [5], and later extended by Goldreich and Ostrovsky [6]. The main construction is based on the *hierarchical solution*, in which the data structure is organised in levels consisting of hash-tables (using secret hash functions known by the client only), and requires frequent *oblivious* re-shuffling of data. Data is scanned by visiting each level, after which the item found is written in the top level. Levels eventually overflow with data, leading to their move downwards. This process requires the re-shuffling of data, which is done *obliviously.* This is the most complex component of the construction, and is the main factor in its (amortised) complexity overhead. The original scheme requires $O(n \log n)$ in storage, and has computation overhead of $O((\log n)^3)$ per query (using a particularly impractical sorting algorithm, or $O((\log n)^4)$ with more reasonable constant in practice).

Since the original proposal of hierarchical solution, several improved schemes have been proposed. Pinkas and Reinman [16] improved the performance of oblivious RAM using Cuckoo hashing and a new oblivious sorting algorithm. However, Kushilevitz *et al.* [12] pointed out a security flaw of the scheme presented in [16]. They also proposed a new scheme with $O((\log n)^3)$ worst-case overhead. Goodrich and Mitzenmacher [8] achieved $O((\log n)^2)$ amortised cost with $O(1)$ client-side storage. Their scheme could achieve higher performance of $O(\log n)$ by using $O(n^\alpha)$ client storage where $0 \leq \alpha \leq 1$. Stefanov *et al.* [18] proposed a scheme with an amortised overhead of (20–35)× by partitioning a bigger oblivious RAM into smaller oblivious RAMs. An efficient worst-case overhead scheme was proposed by Goodrich *et al.* [9]. This scheme also requires $O(n^\tau)$ client storage, where τ is a small constant, and achieves $O(\log n)$ access overhead and $O(n)$ storage overhead.

Goldreich [5,6] also proposed the *square root* construction, which uses secret random permutations when storing data. The solution requires $O(n + \sqrt{n})$ in storage, and has computation overhead of $O(\sqrt{n} \log n)$ per query. See appendix of the extended version of [16] for an overview, or refer to the original paper for a detailed description. We note that our proposal borrows ideas from the square-root solution.

3.2 Hardware-Assisted Control Flow Obfuscation

Oblivious RAM constructions remain too expensive to be implemented on embedded processors. In [21], Zhuang *et al.* proposed a practical, hardware-assisted scheme for embedded processors, with low computational overhead. Their *control flow obfuscation* scheme for embedded processors employs a small secure hardware obfuscator (called *shuffle buffer*) to hide program recurrence. The shuffle buffer is within the CPU trusted boundary, and an adversary is not able to observe access pattern in the shuffle buffer. The first m blocks in memory are moved to the shuffle buffer, which can hold m data blocks. When making a request for a data block, if the block is found in the shuffle buffer, access the block. Otherwise fetch the block from memory and a random block in the shuffle buffer is written back to memory. For more details, refer to [21].

This scheme however leaks information about access of data blocks. The scheme accesses memory only when it is necessary (i.e. when the block is not in the shuffle buffer), hence, one knows the exact block being accessed. Furthermore, an access in memory to a data block which was previously swapped out from the buffer indicates with high probability the existence of repeated access to a particular data.

Despite the limitations of the proposal, it adds a very low overhead to the program execution (besides the read/write and encryption/decryption overhead, only an extra read/write operation due to cache misses). We will surmount this limitation in our proposal.

4 Our Scheme for Memory Access Pattern Protection

While oblivious RAM constructions can completely hide memory access pattern, they are too expensive to be implemented on resource-constrained devices. In this section, we describe our proposal for a practice-oriented memory access pattern protection scheme. One of its main features is the inclusion of an extra buffer used to maintain the *history* of memory access, to help hiding the access pattern. We present below two instances: the first one uses a secure (trusted) hardware buffer, as in [21]; however it achieves higher security by adapting ideas from oblivious RAM mechanisms, yet without the expensive (re-)shuffling of buffers. The second instance requires no special hardware, but as a result leads to a higher, yet practical overhead.

4.1 Assumptions

We consider the problem of hiding the access pattern of memory with n data blocks. Our scheme will assume that the *client* has access to an efficient pseudorandom number generator (to make random choices of addresses), and a semantically secure probabilistic encryption scheme. Data is always stored encrypted: it is decrypted whenever it is read from memory, and re-encrypted whenever it is written into memory. Furthermore, either a `read` or `write` operation will always perform the two operations in every access, the difference is the value being written in the `write` step. As a result of using this method, an access pattern (for the purpose of indistinguishability) can be thought as simply as a sequence of indices i (corresponding to the data blocks accessed).

Our scheme will also require a way to generate a pseudo-random permutation, to map memory addresses. This can be achieved by using a deterministic encryption algorithm $\mathcal{E}$. This mapping will be described either by the function $\mathcal{E}$, with the output computed in each call, or explicitly described as a table lookup (with input-output pairs). We note that the latter requires $O(n)$ memory within the client's trusted boundary (as in [21]), which in some scenarios may be impractical.

Perhaps the main challenge of access pattern protection schemes is to hide the repeated access of data blocks. In general, when not deploying expensive ORAM solutions, access pattern can typically be distinguishable by observing a long series of accesses. Therefore, we define a relaxed, still practical, security definition of access pattern protection, which we call *δ-length security.* Assume that during a certain sequence of data access, a particular block 'a' is accessed twice by a program at t-th access and $(t+\delta)$-th access; we call this a *δ-distance access* of 'a'. Informally, an access pattern protection scheme is δ-length secure if the probability that an adversary identifies repeated accesses in $A(y)$ at distance at most δ is small.

Definition 2. *We say that an access pattern protection scheme is δ-length ϵ-secure if the probability that an adversary identifies any d-distance access in $A(y)$ is at most ϵ for every $d \leq \delta$.*

4.2 Set-Up

On loading n data blocks to memory, our scheme will use $\mathcal{E}$ to permute the corresponding addresses. Dummy data will be typically added to the original data, so that at initialisation we have kn data blocks being loaded to memory, with $k \geq 1$ a small constant. The constant k will be selected based on the typical *epoch* length of the program and the availability of memory within the client's trusted boundary to describe the random permutation $\mathcal{E}$. We discuss the use of dummy data in more detail in Section 5.3.

Our scheme will partition memory into two regions: a (secure) buffer $\mathcal{M}$ and an unsecured memory $\mathcal{L}$, of sizes m and ℓ, respectively, with $m \ll \ell$. It follows that $m+\ell = kn$. Furthermore, we will require a secure table $\mathcal{H}$, called the *history* table. The table $\mathcal{H}$ stores addresses of data blocks that have been moved from the secure buffer $\mathcal{M}$ to the unsecured memory $\mathcal{L}$, and has size ℓ_h. We denote the address of a data block 'a' as i_a. Typically, we may have $\mathcal{H}$ implemented as part of $\mathcal{M}$. In the case where we are able to store the permutation mapping table explicitly, $\mathcal{H}$ can be implemented by adding a *set* bit into the table. Despite these choices, in our discussions below we consider $\mathcal{H}$ as a separate table.

4.3 Instance 1: Construction with Secure Memory

The first instance of our scheme considers the buffer $\mathcal{M}$ being implemented within the client's trusted boundary (as in [21]). The table $\mathcal{H}$ is also stored within this boundary. After loading data into memory, m data blocks are copied into $\mathcal{M}$; the table $\mathcal{H}$ starts empty[2]. On the first access to a data block 'a' (either a `read`(a) or `write`(a, x)), we search for 'a' in $\mathcal{M}$ and access $\mathcal{L}$ twice: if 'a' is in $\mathcal{M}$, we replace two random elements (not 'a') from $\mathcal{M}$ with two random *dummy* elements from $\mathcal{L}$ and we access 'a'; if 'a' is not in $\mathcal{M}$, two random elements from $\mathcal{M}$ are replaced by 'a' and one random *dummy* element from $\mathcal{L}$ (and 'a' is accessed). In both cases, the corresponding addresses of blocks being kicked out from $\mathcal{M}$ are written in $\mathcal{H}$. In subsequent calls, we proceed as follows:

1. if 'a' is in $\mathcal{M}$, we replace two random elements (not 'a') from $\mathcal{M}$ by a random element from $\mathcal{L}$ and a random element from $\mathcal{L}$ which had already been accessed before (as recorded in the *history* buffer), and we access 'a'.

[2] Although, before the program starts to run, the scheme can operate by accessing a number of dummy data blocks to populate the history buffer.

Algorithm 1. Pseudocode of access pattern protection scheme

```
scan $\mathcal{M}$ for '$a$'
if '$a$' $\in \mathcal{M}$ then
    replace two random elements (not '$a$') in $\mathcal{M}$ with two random blocks in $\mathcal{L}$, one
    of them is chosen from the history $\mathcal{H}$ and the other is randomly chosen from $\mathcal{L}$
else
    scan $\mathcal{H}$ for '$i_a$'
    if '$i_a$' $\in \mathcal{H}$ then
        replace two random blocks in $\mathcal{M}$ with a random block in $\mathcal{L}$ and '$a$'
    else
        replace two random blocks in $\mathcal{M}$ with '$a$' and a random block whose address
        is registered in $\mathcal{H}$
    end if
end if
choose $\ell_h$ elements from $\ell_h + 2$ to update history table $\mathcal{H}$
access '$a$'
```

2. if 'a' is not in $\mathcal{M}$, and its address is in the history table, we replace two random elements from $\mathcal{M}$ by a random element from $\mathcal{L}$ and 'a' (as recorded in the *history* buffer). Note that $\mathcal{H}$ holds only addresses and data itself is stored in $\mathcal{L}$.
3. if 'a' is not in $\mathcal{M}$, and its address is not in the history table either, we replace two random elements from $\mathcal{M}$ by 'a' and a random element from $\mathcal{L}$ which had already been accessed before (as recorded in the *history* buffer).

Every time the data blocks are kicked out from $\mathcal{M}$ to $\mathcal{L}$, data blocks are written in $\mathcal{L}$ taking their original position (as described by $\mathcal{E}$), and addresses of those blocks are registered in the history table $\mathcal{H}$. As the program continues to access data blocks, the table may eventually get full. When this is the case, at each access we select at random ℓ_h elements among the $\ell_h + 2$ elements (the current history elements and the two new ones). We have a pseudocode of our scheme in Algorithm 1 and show an example in the appendix.

The Objective of the History Buffer. The goal of our scheme is to efficiently protect the privacy of data access pattern. It is clear that if during a run of the program, data blocks are only accessed once, then the use of a random permutation alone will suffice to hide the access pattern. The case of relevance is thus when a data block is accessed more than once. When it is accessed for the first time, it is copied into $\mathcal{M}$. If accessed again, and it is still in $\mathcal{M}$, then access is oblivious from an adversary; still we would like to hide the fact that the data accessed was found in $\mathcal{M}$. If it is no longer in $\mathcal{M}$, then we would like to hide the fact we are accessing it again from an adversary. Thus, if we consider the case that a program keeps reading 'a' at different intervals, we have the following three types of access to consider:

1. 'a'$\in \mathcal{M}$;
2. 'a'$\notin \mathcal{M}$ and 'i_a' $\in \mathcal{H}$;
3. 'a'$\notin \mathcal{M}$ and 'i_a' $\notin \mathcal{H}$.

Note that case 3 is likely to occur when 'a' is accessed for the first time. As discussed, the cases that require most attention are 2 and 3 when 'a' is accessed again: since we do not perform an oblivious re-shuffling, the adversary would notice that 'a' is being accessed again.

To address this, in our scheme, we first search 'a' in $\mathcal{M}$ and then, depending on the cases described above, the access of memory is done as follows:

1'. access (r, p),
2'. access (r, a),
3'. access (a, p),

where r is a random location in $\mathcal{L}$ and p is a location recorded in the table $\mathcal{H}$.

Note that we need to keep the contents in the history table secret: although the adversary can record all blocks previously accessed, in practice we may not be able to keep the addresses of all accessed blocks in the history table (since ℓ_h may be small). If we had the addresses in cleartext, one may note that in case 3' above, although 'a' had being accessed before, its address was no longer in the history buffer (for lack of space).

4.4 Instance 2: Construction without Secure Memory

We consider a second instance of our proposal, which does not require a secure buffer. The buffer $\mathcal{M}$ and history table $\mathcal{H}$ are kept in the unsecured memory area, as with $\mathcal{L}$. In this case, access to $\mathcal{M}$ (and $\mathcal{H}$) is made by reading and writing every data block in the buffers (which requires decryption and encryption of data). Thus, to find out whether 'a' is in $\mathcal{M}$, we read/write all values; when replacing data blocks in $\mathcal{M}$, we again read/write all values. Except for this, the access is made as described in Section 4.3. The security provided is the same, but the computation overhead is obviously increased, and it is dependent of the sizes of buffers $\mathcal{M}$ and $\mathcal{H}$.

5 Security Analysis

We discuss the security of our scheme.

5.1 Access Pattern Hiding

Regarding recurrence, recall that in our scheme, we first search 'a' in $\mathcal{M}$ and then, depending on the cases described in Section 4, the access of memory is as follows:

1. 'a'$\in \mathcal{M}$, and 'a' is accessed: access (r, p) in $\mathcal{L}$;
2. 'a'$\notin \mathcal{M}$, 'a'$\in \mathcal{H}$, and 'a' is accessed: access (r, a) in $\mathcal{L}$;
3. 'a'$\notin \mathcal{M}$, 'a'$\notin \mathcal{H}$, and 'a' is accessed: access (a, p) in $\mathcal{L}$.

Assume that the program accesses 'a' at time t; we denote it by $X_t = a$. We have the following lemma.

Lemma 1. *Assume that $X_t = a$, and let m and ℓ_h denote the sizes of the $\mathcal{M}$ and $\mathcal{H}$, respectively. Then after δ steps we have $p_M = \Pr[a \in \mathcal{M}] \geq \left(\frac{m-2}{m}\right)^{\delta}$ and $p_H = \Pr[a \in \mathcal{H}] \geq \left(\frac{\ell_h}{\ell_h+2}\right)^{\delta}$.*

Proof. To compute $\Pr[a \in \mathcal{M}]$, note that the right-hand side of the expression corresponds to the probability that an element remains in a set of size m after δ replacements of 2 elements at time, which is how the scheme manages the buffer $\mathcal{M}$. The inequality comes from the fact that even if removed after $d < \delta$ steps, 'a' may be re-inserted during the normal operation of the scheme. Showing the second probability is similar (noting however that in the history buffer, the scheme draws 2 elements among $\ell_h + 2$ elements). □

For the sake of simplicity, we will in the remaining of this paper assume equality in the two expressions above. Furthermore, we will also assume that the two events are independent (obviously the probability of 'a' being in $\mathcal{H}$ after δ steps will be influenced by whether/when 'a' leaves the buffer $\mathcal{M}$; however we believe this assumption is reasonable for typical values of m, ℓ_h and δ – and substantially simplifies our computations). The simple lemma below then follows.

Lemma 2. *Consider the three cases for memory access discussed above, and assume that $X_t = X_{t+\delta} = a$. Then the probability that we have case 1 is p_M, the probability that we have case 2 is $p_H(1 - p_M)$, and the probability that we have case 3 is $(1 - p_H)(1 - p_M)$.*

Now assume that an adversary observes the scheme at time $t + \delta$ in case 1, i.e. we have $a \in \mathcal{M}$ and access to (r, p) from $\mathcal{L}$. Then following a conservative estimate, we have $\Pr[X_{t+\delta} = a \mid \text{ case 1 }] \leq \frac{1}{m-2}$. For case 2, we have a similar upper bound: $\Pr[X_{t+\delta} = a \mid \text{ case 2 }] \leq \frac{1}{m-2}$. Case 3 is perhaps the one in which an adversary can extract more information (since it is very likely that, unlike the other two cases, the pair of elements (a, p) drawn from $\mathcal{L}$ have already been observed by the adversary). We will again adopt a conservative approach, and have the upper-bound $\Pr[X_{t+\delta} = a \mid \text{ case 3 }] \leq 1/2$.

Theorem 1. *The proposed scheme is δ-length ϵ-secure access pattern protection scheme, where*

$$\epsilon \leq \frac{p_M}{(m-2)^2} + \frac{(1-p_M)p_H}{(m-2)^2} + \frac{(1-p_M)(1-p_H)}{2(m-2)}.$$

Proof. Let $\mathcal{A}$ be adversary who is able to observe an access sequence $X_i = a_i$, for $i = 1, \ldots, N$. Let us assume that $X_t = X_{t+\delta} = a$. We wish to compute the probability $\Pr[X_t = a, X_{t+\delta} = a]$. We assume that the first access to a is at time t (the proof and figures can be slightly modified when this is not the case), and that the accesses at time t and $t + \delta$ are independent events (i.e. we assume no knowledge of statistics of the original program being protected). Then we have $\Pr[X_t = a] \leq 1/(m-2)$, and by lemmas and discussion above.

$$\begin{aligned}\Pr[X_t = a, X_{t+\delta} = a] &= \Pr[X_t = a] \cdot \Pr[X_{t+\delta} = a] \\ &\leq \frac{1}{m-2}(\Pr[X_{t+\delta} = a \mid \text{case 1 }] \cdot \Pr[\text{ case 1 }] \\ &\quad + \Pr[X_{t+\delta} = a \mid \text{case 2 }] \cdot \Pr[\text{ case 2 }] \\ &\quad + \Pr[X_{t+\delta} = a \mid \text{case 3 }] \cdot \Pr[\text{ case 3 }]) \\ &\leq \frac{p_M}{(m-2)^2} + \frac{(1-p_M)p_H}{(m-2)^2} + \frac{(1-p_M)(1-p_H)}{2(m-2)}. \qquad \square\end{aligned}$$

5.2 Parameters: Size of Secure Memory and History Table

The choice for sizes of the secure memory $\mathcal{M}$ and history table $\mathcal{H}$ have an obvious influence in the security and efficiency/cost of the scheme: it follows from Theorem 1 that large values for m and ℓ_h significantly decreases the chances that an adversary can identify a repeated access to a particular memory block. However large $\mathcal{M}$ and $\mathcal{H}$ negatively affect the performance of the scheme (as well as increase its costs in the hardware-assisted version).

For instance, if we take the size of the shuffle buffer as 128 16-byte blocks (as in [21]), and the same size for $\mathcal{H}$ (that is, 512 32-bit addresses), then for $\delta = 20$, we have $p_M \approx 0.73$ and $p_H \approx 0.92$, and as a result $\epsilon \leq 1.4 \times 10^{-4}$. The value increases to 4.4×10^{-4} if $\delta = 50$, and to 1.06×10^{-3} if $\delta = 100$ In general, SIM cards have a capacity of 64KB, which means we can allocate much more space, say 1024 blocks and 4×1024 history addresses. In this case, we have $p_M \approx 0.82$ and $p_H \approx 0.95$, and as a result $\epsilon \leq 0.5 \times 10^{-5}$ for $\delta = 100$.

As discussed before, we note however that our scheme does not achieve strong indistinguishability: As we use a static permutation $\mathcal{E}$, addresses are fixed (after the permutation), and this implies some leakage of information (an adversary will, for instance, know when the sets of potentially accessed blocks are disjoint, implying no recurrence). Overall, we believe that, a choice of parameters can be made to achieve both security and performance levels acceptable in practice.

5.3 How Much Dummy Data Should We Use?

Oblivious RAMs provide security by making sure that a data block is only accessed once while it remains at the same address. When it is accessed again, it will be allocated to a completely different address and the adversary will have no information about the contents and whether/when it was accessed before. Our scheme, on the other hand, tries to ensure the security in the circumstance that the data block is accessed multiple times while in the same address.

If there are several dummy data blocks, that can be used as random elements that the protection scheme can access. They will also prevent the adversary from determining which blocks are actually accessed by the program. We can also make dummy accesses to the actual data blocks when it is not accessed by the program. When the program accesses a data block, the access pattern protection scheme will access 2 blocks as the scheme always swaps 2 blocks between $\mathcal{M}$ and $\mathcal{L}$. If we add n dummy data blocks, the number of accesses to the actual data and dummy data will be roughly the same. Guessing the access pattern correctly becomes harder as n increases, and less dummy blocks will be required. While we still need to confirm it experimentally, we expect that the required number of dummy blocks will be at most $2n$ and the constant k will be $1 \leq k \leq 3$.

6 Comparison

Our scheme requires secure memory for storing m data blocks. It also requires storage for the history table of size ℓ_h. When we read a data block, we first access the secure memory to check whether the data block sought is in $\mathcal{M}$. We then move two data blocks into $\mathcal{M}$ and read the data block. We also need to access and update the history table. Thus the total cost is 4 operations, plus the cost associated with reading and writing the history table (which can be only two extra accesses if $\mathcal{H}$ is within the secured boundary). Thus we have the overhead of 7 operations. We discuss how to improve the overhead later this section. The history can be potentially stored (encrypted) in the memory area $\mathcal{L}$. When $\mathcal{H}$ is not within the secured boundary, we have to read and update all blocks in the table. Thus we have the overhead of $4 + 2\ell_h$. Finally, if using *dummy* data, we also have data storage overhead in $\mathcal{L}$ depending on the constant k.

Our scheme can also be implemented without any special hardware. The difference in this case is that we need to scan the entire buffer $\mathcal{M}$ twice, plus the access to the history table and two swaps. We have therefore the cost of $2m + 2\ell_h + 2$ operations. As we assume m is much smaller than n, our scheme without a hardware is still more efficient than most of oblivious RAMs. According to [18], practical overhead of oblivious RAMs are, in general, on the range of thousands to hundred of thousands. When we choose $m = \ell_h = 128$, the overhead is 514, which is less than half of that of oblivious RAMs. The scheme proposed of [18] achieved the overhead of $(20 - 35)\times$, which is faster than our second scheme, still our scheme has the advantage in terms of storage size, that is, ours requires less than $3n$ storage for n original data while the scheme in [18] requires $4n + o(n)$.

Overall, the parameters m, ℓ_h and k can be set to suitable values, to offer the appropriate trade-off between security and implementation/computational costs. While oblivious RAMs are too expensive especially for resource constrained devices and Zhuang *et al.*'s scheme is not suitable scheme for this purpose, our scheme can offer reasonable security for the access pattern protection problem with constant computational overhead and practical storage.

Improvement of Performance. When we update the history table, addresses of two data blocks are registered into the history table independently in the original scheme. Assume that two addresses 'i_a' and 'i_b' need to be registered, then two random locations are selected and those locations are updated with

Table 1. Comparison with Other Schemes

	Pattern Leakage	Computational Overhead	Storage
Square root [6]	No	$O(\sqrt{n}\log n)$	$O(n+\sqrt{n})$
Pinkas-Reinman [16]	No	$O((\log n)^2)$	$8n$
Stefanov *et al.* [18]	No	$O(\log n)$	$4n+o(n)$
Goodrich-Mitzenmacher [8]	No	$O(\log n)$	$8n$
Lu-Ostrovsky [14]	No	$O(\log n)$	$O(n)$
Kushilevitz *et al.* [12]	No	$O((\log n)^2/\log\log n)$	$O(n)$
Zhuang *et al.* [21]	Yes	2	$2n$
Ours w/ Sec. Mem.	see Theorem 1	6	$\leq 3n$
Ours w/o Sec. Mem.	see Theorem 1	$2(m+\ell_h+1)$	$\leq 3n$

the new blocks. Therefore, the update requires two operations in each cycle. The update can be done with only one operation by modifying the entry as a bigger one $i_a||i_b$, where $||$ is concatenation. When choosing one data block from the history table, one can first choose the concatenated block and then can choose higher half or lower half of the block. As a result, we can reduce the cost for updating the history table from 2 to 1 and the overall overhead is improved to 6. Table 1 gives the comparison of the performance with other schemes.

7 Conclusion

In this paper, we proposed two new schemes for protecting memory access patterns. The distinctive character of our scheme is that we do not re-shuffle the order of the data blocks in memory. To protect the access pattern without reshuffling, we used a *history* of the accesses. We first considered an instance which is similar to the proposal in [21], and also relies on the use of a secure (trusted) hardware buffer; however it achieves higher security by adapting ideas from Oblivious RAM mechanisms, without the expensive (re-)shuffling of buffers. Another instance requires no special hardware, but as a result leads to a higher, yet practical overhead. We defined a new security notion called δ-length ϵ-security and proved that the proposed two schemes are secure in this notion. We also discussed the size of parameters, which are the size of secure memory, history table and dummy data and compared the performance with existing schemes. We claim that under reasonable assumptions, the schemes can achieve both security and performance levels acceptable in practice.

References

1. Ajtai, M.: Oblivious RAMs without cryptographic assumptions. In: Schulman, L.J. (ed.) STOC, pp. 181–190. ACM (2010)
2. Boneh, D., Mazieres, D., Popa, R.A.: Remote Oblivious Storage: Making Oblivious RAM Practical. Technical Report MIT-CSAIL-TR-2011-018, Massachusetts Institute of Technology (2011)

3. Chor, B., Goldreich, O., Kushilevitz, E., Sudan, M.: Private Information Retrieval. In: FOCS, pp. 41–50. IEEE Computer Society (1995)
4. Damgård, I., Meldgaard, S., Nielsen, J.B.: Perfectly Secure Oblivious RAM without Random Oracles. In: Ishai, Y. (ed.) TCC 2011. LNCS, vol. 6597, pp. 144–163. Springer, Heidelberg (2011)
5. Goldreich, O.: Towards a Theory of Software Protection and Simulation by Oblivious RAMs. In: Aho, A.V. (ed.) STOC, pp. 182–194. ACM (1987)
6. Goldreich, O., Ostrovsky, R.: Software Protection and Simulation on Oblivious RAMs. J. ACM 43(3), 431–473 (1996)
7. Goodrich, M.T.: Data-Oblivious External-Memory Algorithms for the Compaction, Selection, and Sorting of Outsourced Data. In: Rajaraman, R., Meyer auf der Heide, F. (eds.) SPAA, pp. 379–388. ACM (2011)
8. Goodrich, M.T., Mitzenmacher, M.: Privacy-Preserving Access of Outsourced Data via Oblivious RAM Simulation. In: Aceto, L., Henzinger, M., Sgall, J. (eds.) ICALP 2011, Part II. LNCS, vol. 6756, pp. 576–587. Springer, Heidelberg (2011)
9. Goodrich, M.T., Mitzenmacher, M., Ohrimenko, O., Tamassia, R.: Oblivious RAM simulation with Efficient Worst-Case Access Overhead. In: Cachin, C., Ristenpart, T. (eds.) CCSW, pp. 95–100. ACM (2011)
10. Goodrich, M.T., Mitzenmacher, M., Ohrimenko, O., Tamassia, R.: Practical Oblivious Storage. In: Bertino, E., Sandhu, R.S. (eds.) CODASPY, pp. 13–24. ACM (2012)
11. Goodrich, M.T., Mitzenmacher, M., Ohrimenko, O., Tamassia, R.: Privacy-Preserving Group Data Access via Stateless Oblivious RAM Simulation. In: Rabani, Y. (ed.) SODA, pp. 157–167. SIAM (2012)
12. Kushilevitz, E., Lu, S., Ostrovsky, R.: On the (In)security of Hash-based Oblivious RAM and a New Balancing Scheme. In: Randall, D. (ed.) SODA, pp. 143–156. SIAM (2012)
13. Lorch, J.R., Mickens, J.W., Parno, B., Raykova, M., Schiffman, J.: Toward Practical Private Access to Data Centers via Parallel ORAM. IACR Cryptology ePrint Archive, 2012:133 (2012)
14. Lu, S., Ostrovsky, R.: Distributed Oblivious RAM for Secure Two-Party Computation. IACR Cryptology ePrint Archive, 2011:384 (2011)
15. Osvik, D.A., Shamir, A., Tromer, E.: Cache Attacks and Countermeasures: The Case of AES. In: Pointcheval, D. (ed.) CT-RSA 2006. LNCS, vol. 3860, pp. 1–20. Springer, Heidelberg (2006)
16. Pinkas, B., Reinman, T.: Oblivious RAM Revisited. In: Rabin, T. (ed.) CRYPTO 2010. LNCS, vol. 6223, pp. 502–519. Springer, Heidelberg (2010)
17. Shi, E., Chan, T.-H.H., Stefanov, E., Li, M.: Oblivious RAM with $O((\log N)^3)$ Worst-Case Cost. In: Lee, D.H., Wang, X. (eds.) ASIACRYPT 2011. LNCS, vol. 7073, pp. 197–214. Springer, Heidelberg (2011)
18. Stefanov, E., Shi, E., Song, D.: Towards Practical Oblivious RAM. CoRR, abs/1106.3652 (2011)
19. Williams, P., Sion, R.: Usable PIR. In: NDSS. The Internet Society (2008)
20. Williams, P., Sion, R.: SR-ORAM: Single Round-trip Oblivious RAM. In: ACNS, Industrial Track, pp. 19–33 (2012)
21. Zhuang, X., Zhang, T., Lee, H.-H.S., Pande, S.: Hardware Assisted Control Flow Obfuscation for Embedded Processors. In: Irwin, M.J., Zhao, W., Lavagno, L., Mahlke, S.A. (eds.) CASES, pp. 292–302. ACM (2004)

Appendix

We present an example of the scheme in Figure 1. In this example, we have $m = 4$ and $\ell_h = 6$, then the program reads data in the order $5 \rightarrow D \rightarrow 8 \rightarrow 5$. For simplicity, we will not be using dummy data (i.e. $k = 1$) and we have a fixed permutation.

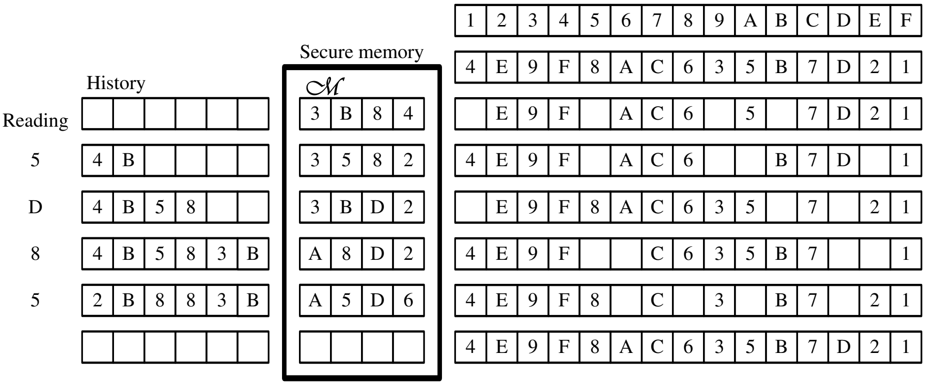


Fig. 1. Our Scheme

1. All data in the memory $\mathcal{L}$ is randomly permuted and $\mathcal{M}$ is filled with the 4 random blocks.
2. When the program tries to access 5, since it is not in $\mathcal{M}$, blocks 5 and 2 are brought into $\mathcal{M}$. Since we do not have any history yet, block 5 and a random block is chosen. Two blocks (4 and B) in $\mathcal{M}$ are written back to $\mathcal{L}$, and their addresses are entered in the history.
3. When the program accesses D, the block D is brought into $\mathcal{M}$. The block B is chosen from the history and also brought into $\mathcal{M}$. Blocks 5 and 8 are instead written back to $\mathcal{L}$, and their addresses are entered in the history.
4. When the program accesses 8, since 8 is in the history, block A is randomly chosen. Blocks 3 and B are written back to $\mathcal{L}$, and their addresses are entered in the history.
5. When the program accesses 5, since the address of 5 is in the history, the block 6 is randomly chosen to be brought into the $\mathcal{M}$. Blocks 2 and 8 are written back to $\mathcal{L}$. Now the history table is full, two addresses of random blocks (in this example, 4 and 5) are replaced with those of 2 and 8.
6. All blocks in $\mathcal{M}$ are written back into $\mathcal{L}$.

Multipurpose Cryptographic Primitive ARMADILLO3

Petr Sušil[*] and Serge Vaudenay

EPFL, Lausanne, Switzerland
{petr.susil,serge.vaudenay}@epfl.ch

Abstract. This paper describes a new design of the multipurpose cryptographic primitive ARMADILLO3 and analyses its security. The ARMADILLO3 family is oriented on small hardware such as smart cards and RFID chips. The original design ARMADILLO and its variants were analyzed by Sepehrdad et al. at CARDIS'11, the recommended variant ARMADILLO2 was analyzed by Plasencia et al. at FSE'12 and by Abdelraheem et al. at ASIACRYPT'11. The ARMADILLO3 design takes the original approach of combining a substitution and a permutation layer. The new family ARMADILLO3 introduces a reduced-size substitution layer with 3 × 3 and 4 × 4 S-boxes, which covers the substitution layer from 25% to 100% of state bits, depending on the security requirements. We propose an instance ARMADILLO3-A1/4 with a pair of permutations and S-boxes applied on 25% of state bits at each stage.

1 Introduction

Tiny computing devices such as smart cards, sensor networks and RFID tags are becoming more and more widespread. The implementation of standardized cryptographic algorithms such as the block cipher AES [21] or the hash functions SHA [10] are very expensive in terms of the number of gates and power consumption. Moreover, the security requirements of these tiny devices are often weaker than which of algorithms such as AES or SHA. The widespread usage of the constrained devices triggered a spontaneous competition for the tiniest and the most secure designs. There have been several designs of such primitives [5,8,9,14,15,17,25].

Since these devices communicate over an insecure channel, usually a wireless channel, there is a threat of an attacker trying to listen to the communication or trying to impersonate a server or another device. Therefore, there is a need for an authentication protocol to provide authenticity of the device, and an encryption to provide the confidentiality. However, as we want to reduce the implementation cost as much as possible, it is important to find a universal design, which can be used in many different applications. This allows to further reduce the

[*] Supported by a grant of the Swiss National Science Foundation, 200021_134860/1. This work has been supported in part by the European Commission through the ICT program under contract ICT-2007-216646 ECRYPT II.

S. Mangard (Ed.): CARDIS 2012, LNCS 7771, pp. 203–218, 2013.

implementation cost, as it is not necessary to implement multiple algorithms on the small device. Some recent designs deal with this issue by reusing some parts of the implementation, for instance the hash function QUARK [2] and the message authentication code SQUASH-128 [23] use some components of the stream cipher GRAIN [14]. This approach is the first step towards a multipurpose cryptographic primitive, that can be used in all applications.

We introduce a new primitive ARMADILLO3 which is designed to be used as a message authentication code (MAC), a hash function and a pseudo-random number generator. The ARMADILLO3 is the third generation of the multipurpose cryptographic function ARMADILLO [3] introduced at CHES'10. The new version ARMADILLO3 prevents all known attacks against the ARMADILLO [22] design and the attack against ARMADILLO2 based on parallel matching [1], and Hamming weight preservation in PRNG mode [19]. We provide a security analysis against known types of attacks and discuss some dedicated attacks and counter-measures. We support our security claims using the security analysis based on properties of the underlying expander graph of ARMADILLO3.

The ARMADILLO is a family of cryptographic functions based on data dependent permutations. That is, we use an internal function P defined by $P(p\|b, Z) = P(p, S(Z_{\sigma_b}))$ iteratively, where b is the tailing bit of the first operand $p\|b$, S is a substitution layer, σ_b is a permutation (σ_0 or σ_1) and Z_{σ_b} denotes the transposition of Z based on permutation σ_b. The extension ARMADILLO3 adopts a preprocessing to prevent the known attacks against ARMADILLO1 reported in [22], and it introduces a reduced-size S-box layer to improve the confusion of ARMADILLO2 which lead to a practical low complexity attack reported in [19].

The ARMADILLO3 internal function generalizes the SPN structure by introducing a second permutation. In every round, we choose one of the two permutations based on a pseudorandom value.The internal function is then followed by an XOR with the input and the control register value similar to the Davies-Meyer construction.

The ARMADILLO3 reduces the number of S-boxes due to both the higher number of rounds and the pseudorandom selection of the permutation. This means that only some bits of the internal state go through the S-boxes in a single round. The pair of permutations for ARMADILLO3 has to be selected in such a way that even when the attacker controls the selection of the permutation at every round, which is the case for hash functions, she should be unable to prevent the diffusion of the input. Therefore, the selection of the two permutations is a non-trivial task. We introduce a notion of Hierarchical Permutations which ensure that every bit goes through an S-box in a minimum number of steps for all possible sequences of data-dependent permutations, while making no significant restrictions on other properties of these permutations. The selection of the final pair is based on the diffusion properties of both permutations and the expansion properties of the expander graph corresponding to the pair of the permutations.

2 The ARMADILLO3 Function

The ARMADILLO3 is based on a recursive function P which takes two parameters $P(Y, X)$. The register Y is used as a control register for selecting the permutation in each step of the function P, and the ith step of P consists of applying permutation σ_0 or σ_1, depending on the value $Y[i]$, and the S-boxes on specified bits. Since the value Y has to be pseudo-random which is difficult to control for an attacker, we set $Y = P(X, X)$ for an input X. Therefore, the ARMADILLO3 consists of two steps: preprocessing step for computing the value Y and the computation of $P(Y, X)$ followed by an XOR with the input X and the control register Y. We give recursive definition of ARMADILLO3 followed by the pseudo-code. The parameters for the algorithm are: the type of S-boxes, the number and placement of S-boxes, and the permutation pair.

The ARMADILLO3 algorithm on input $\mathsf{W} = \mathsf{H}\|\mathsf{X}$ is defined as follows.

$$\begin{aligned} \mathsf{ARMADILLO3}(\mathsf{W}) &= P(\mathsf{Y}, \mathsf{W}) \oplus \mathsf{W} \oplus \mathsf{Y}, \text{ for} \\ \mathsf{Y} &= P(\mathsf{W}, \mathsf{W}) \\ P(p\|b, Z) &= P(p, S(Z_{\sigma_b})) \\ P(\lambda, Z) &= Z \end{aligned}$$

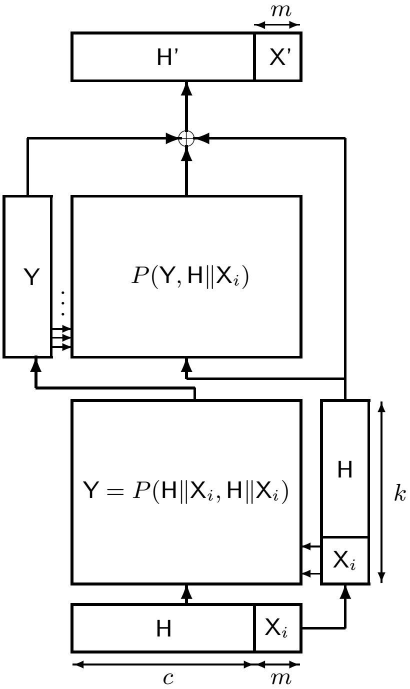


Fig. 1. Scheme of ARMADILLO3

where p denotes a bit string, b denotes a bit, S denotes the substitution layer which is defined separately, and λ denotes the empty string. The substitution layer of ARMADILLO3 consists of r identical $t \times t$ S-boxes. In our example, i.e., ARMADILLO3-A1/4, we build the permutation pair σ_0, σ_1 for the placement of 3×3 S-boxes at positions 0-32 (so we have 11 3×3 S-boxes), which gives the "coverage rate" $= \frac{rt}{k}$, where k is the size of register Y.

Algorithm 1. ARMADILLO3 pseudo-code

```
input X, H
W = H||X
$R$ = H||X
for i=0 to |W| do
  $b$ = W[$i$]
  $R \leftarrow S(R_{\sigma_b})$
end for
for i=0 to |W| do
  $b = P[i]$
  W ← $S(\mathsf{W}_{\sigma_b})$
end for
return W
```

The substitution layer S, i.e., the S-boxes and the bits covered by S-boxes, together with permutations σ_0, σ_1 are defined later for ARMADILLO3-A1/4.

The function ARMADILLO3 differs from the original design ARMADILLO1 [3] in several ways. Like ARMADILLO2 [3], it has an internal register of size k instead of $2k$, which makes the design more compact, and as ARMADILLO2 it also includes a preprocessing step, i.e., $\mathsf{Y} = P(\mathsf{H}||\mathsf{X}, \mathsf{H}||\mathsf{X})$. The preprocessing prevents the attacker from controlling the permutation $P(\mathsf{Y}, \mathsf{H}||\mathsf{X})$, since it is difficult for an attacker to predict $\mathsf{Y} = P(\mathsf{H}||\mathsf{X}, \mathsf{H}||\mathsf{X})$ without a knowledge of H. In the case of the hash function, when the attacker knows the value H or is allowed to choose this value to find a pseudocollision, the attacker can only control the register in the preprocessing phase. The ARMADILLO3 differs from ARMADILLO2 [3] by removing the XOR with a constant and adding a reduced-size substitution layer.

2.1 Modes of Operations in ARMADILLO3

FIL-MAC. The fixed input-length message authentication code is required in RFID applications. The output X' is used for authentication of the tag. In applications such as Pathchecker [20], the secret key of the RFID tag is renewed with the value H'.

Hashing. For a variable-length input message we use the strengthened Merkle-Damgård construction [18,7]. The ARMADILLO3 is used as a compression function. The value H is taken as the IV and the compression function ARMADILLO3 produces H' which is the new IV. The value X is a message block to be processed. Such construction is similar to a sponge construction proposed in [4]. The inner function of ARMADILLO3 could also be used in a sponge construction as an alternative to our construction.

PRNG, PRF. In this mode we use the j most significant bits of the output value $(\mathsf{H'}\|\mathsf{X'}) = \mathsf{ARMADILLO3}(\mathsf{H},\mathsf{X})$, where j is a parameter. The input value X is chosen sequentially, and can be sent in clear for the resynchronization purposes for a self-synchronizing stream cipher.

2.2 The Permutation Pair for ARMADILLO3

We introduce a concept of Hierarchical Permutation which ensures the avalanche effect in small number of rounds even if the selection of permutations is under full control of the attacker. Given the set of indices X and a set of indices S, covered by S-boxes, we define a height of $i \in X$ for the permutation π as $h(i) = \min_j\{j : \pi^j(i) \in S\}$. We now explain how to build the Hierarchical permutation for $t \times t$ S-boxes, and give a concrete example for the case $t = 3$. We suppose that the layer of S-boxes covers bits $[0, tr - 1]$. We define sets of indices A_i, B_i, and C_i so that $\sum_{i=0}^{h}|A_i| + \sum_{i=0}^{h-1}|B_i| + \sum_{i=0}^{h-2}|C_i| = k$, i.e., A_i, B_i, C_i are partitions of $[0, k-1]$. In the case $t = 3$ let a, b and c be integers such that $a + b + c = 3r$, and similarly in the case $t = 4$ let a, b, c and d be integers such that $a + b + c + d = 4r$. Ideally, we would have $a = b = c = r$, but this is not always possible. So, we target $a \approx b \approx c \approx r$. We define several sets A_i, B_i and C_i to partition $\{0, 1, 2, \dots, k-1\}$: $A_h = \{a+b+c, \dots, 2a+b+c-1\}$, $A_{h-1} = \{2a+b+c, \dots, 3a+b+c-1\}$, $B_{h-1} = \{3a+b+c, \dots, 3a+2b+c-1\}$, $A_{h-2} = \{3a+2b+c, \dots, 4a+2b+c-1\}$, $B_{h-2} = \{4a+2b+c, \dots, 4a+3b+c-1\}$, $C_{h-2} = \{4a+3b+c, \dots, 4a+3b+2c-1\}$, $A_{h-3} = \{4a+3b+2c, \dots, 5a+3b+2c-1\}$, $\dots$, $A_0 = \{0, 1, 2, \dots, a-1\}$, $B_0 = \{a, \dots, a+b-1\}$, $C_0 = \{a+b, \dots, a+b+c-1\}$. In what follows, AB_i denotes the union of A_i and B_i. ABC_i denotes the union of A_i, B_i, and C_i. We further define pairwise disjoint sets A_{h+1}, B_h, and C_{h-1} so that $A_{h+1} \cup B_h \cup C_{h-1} = S = ABC_0$ and that output bits from an S-box fall into different sets A_{h+1}, B_h and C_{h-1} (with very few exceptions). In the case when $|A_{h+1}| = |B_h| = |C_{h-1}|$ we set $A_{h+1} = \{3i : i \in [0, r-1]\}$, $B_h = \{3i + 1 : i \in [0, r-1]\}$ and $C_{h-1} = \{3i + 2 : i \in [0, r-1]\}$ or in case of 4×4 S-boxes $A_{h+1} = \{4i : i \in [0, r-1]\}$, $B_h = \{4i + 1 : i \in [0, r-1]\}$, $C_{h-1} = \{4i + 2 : i \in [0, r-1]\}$ and $D_{h-2} = \{4i + 3 : i \in [0, r-1]\}$. In the case when the sets A_{h+1}, B_h and C_{h-1} are not balanced, we select the excess elements to be far from each other. We construct σ such that A_{h+1} is mapped to A_h. B_h is mapped to B_{h-1}. C_{h-1} is mapped to C_{h-2}. A_h is mapped to A_{h-1}. AB_{h-1} is mapped to AB_{h-2}. ABC_{h-2} is mapped to ABC_{h-3}. ABC_{h-3} is mapped to ABC_{h-4}. Etc. These constraints are depicted in Fig. 2. Note that

$$\mathcal{T} = \{A_{h+1}, B_h, C_{h-1}, A_h, AB_{h-1}, ABC_{h-2}, ABC_{h-3}, \dots, ABC_1\}$$

is a partition of $\{1,\ldots,k\}$, since $A_{h+1} \cup B_h \cup C_{h-1} = ABC_0$. From the construction of Hierarchical Permutation, we have that every i in A_i $(i \le h+1)$ and B_i $(i < h)$, C_i $(i < h-1)$ has height i. The unbalanced-height structure makes it such that the output bits of the S-box will meet the S-box layer every $h+1$, h, or $h-1$ iterations. That is, two bits of the same height are likely to have different heights after going through their respective S-box. When the structure is balanced with $a = b = c$, we can take A_{h+1} to the list of the first output bits of S-boxes, B_h to the list of the second output bits of the S-boxes, and C_{h-1} to the list of the third output bits of the S-boxes. This way, two bits going out from the same S-box cannot meet in the same S-box the next time since they have different height. When the structure is unbalanced, it should be close to the same situation. Exceptions to this rule are called "collisions". In the case of ARMADILLO3-A1/4 we have $a = 9$, $b = 11$, and $c = 13$ for $r = 11$, $h = 4$. This gives coverage $\frac{33}{128} \approx \frac{1}{4}$, and A_5, B_4, C_3 as follows

- $A_5 = \{0,3,6,9,12,18,21,24,27\}$
- $B_4 = \{1,4,7,10,13,16,19,22,25,28,31\}$
- $C_3 = \{2,5,8,11,14,17,20,23,26,29,32,15,30\}$

Additionally, we check that for $\sigma'(x) = \sigma^{h(x)}(x)$ we have

$$\left\lfloor \frac{\sigma'(15)}{3} \right\rfloor \neq \left\lfloor \frac{\sigma'(17)}{3} \right\rfloor \quad \text{and} \quad \left\lfloor \frac{\sigma'(30)}{3} \right\rfloor \neq \left\lfloor \frac{\sigma'(32)}{3} \right\rfloor .$$

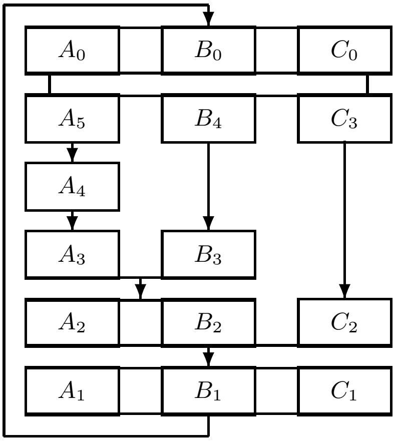


Fig. 2. Hierarchy for Permutations with $h = 4$ and $t = 3$

We provide additional criteria to achieve the highest possible diffusion for a pair of Hierarchical permutations.

Definition 1 (Distance). *Let σ_0, σ_1 be a pair of the permutations on a set $\{1,\ldots,k\}$. We say that they have a distance d_ℓ at level ℓ where*

$$d_\ell(\sigma_0,\sigma_1)=\min_{U,V\in\{0,1\}^\ell,U\neq V}|\{j:\sigma_U(j)\neq\sigma_V(j)\}|$$

For the selection of σ_0 and σ_1 we maximize $d_2(\sigma_0,\sigma_1)$. The high distance d_1 is associated with small number of fixed points of permutation $\sigma_0\sigma_1^{-1}$. Similarly, the high distance d_2 is associated with small number of fixed points of the permutation $\sigma_U\sigma_V^{-1}$, for all $U,V\in\{0,1\}^2$. Otherwise, for some values U and V of the control register, the bit from position i is mapped to $\sigma_U(i)=\sigma_V(i)=j$ for some j. This allows the attacker to predict the behavior of the unknown permutation. In ARMADILLO3-A1/4, which is defined in Section 4, we require $d_2(\sigma_0,\sigma_1)=127$, which means there is at most one index i which is mapped to the same index j by the permutations σ_U and σ_V, where $U,V\in\{0,1\}^2$.

Definition 2 (Graph $\Omega_{\sigma_0\sigma_1,U}$). *Let U be a bitstring and r, t be integers (representing the number and the types of S-boxes respectively). We define a multigraph $\Omega_{\sigma_0\sigma_1,U}=(V,E)$ for permutations σ_0, σ_1 and parameters r and t as follows:*

$$V=\bigcup_{i=0}^{|U|}V^i \qquad V^i=\{v_{i,j}:j\in\{1,\ldots,k\}\} \qquad E=\bigcup_{i=1}^{|U|}\left(E^i\cup S^i\right)$$

$$E^i=\left(\underbrace{\left(v_{i-1,j}\ ,\ v_{i,\sigma_{U_i}(j)}\right),\ldots,\left(v_{i-1,j}\ ,\ v_{i,\sigma_{U_i}(j)}\right)}_{t};\ rt<j\leq k\right)$$

$$S^i=\left(\left(v_{i-1,j*t+a+1}\ ,\ v_{i,\sigma_{U_i}(j*t+b+1)}\right);\ j<r\ \textit{and}\ a,b\in\{0,\ldots,t-1\}\right)$$

where U_i denotes the ith bit of U.

The set E^i is a multiset of edges between level $i-1$ and level i where every edge is taken t times, and the set S^i is a set of edges representing the S-boxes, i.e., for every S-box we have a complete bipartite graph $t\times t$. Therefore, the definition 2 gives a t-regular multigraph (since some edges are repeated t times), i.e., $\Omega_{\sigma_0\sigma_1,U}$ is an expander graph. Combinatorically, the expander graphs are highly connected sparse graphs, probabilistically expander graphs behave like random graphs. Let λ_0 denote the second largest eigenvalue of adjacency matrix of graph G. We now introduce a new criterion which measures the randomness of the graph $\Omega_{\sigma_0\sigma_1,U}$. This criterion is based on the expander graph theory, the reader is referred to [24] for details. We recall that an expander graph is a τ-regular graph G with expansion factor $D(G)>c$ for some constant $c>0$ and some $\tau\in\mathbb{N}$, where the expansion factor $D(G)$ is given by the following formula. Let $\delta(S)$ denote a set of edges neighboring of S, then

$$D(G)=\min_{0<|S|\leq\frac{|V|}{2}}\frac{|\delta(S)|}{|S|}$$

Let σ_0, and σ_1 be permutations. We say that the graph $\Omega_{\sigma_0\sigma_1,U}$ diffuses if for all $v \in V^0$ and $w \in V^{|U|}$ there exists an oriented path from v to w in graph $\Omega_{\sigma_0\sigma_1,U}$. We say that the pair (σ_0, σ_1) has diffusion level $\mathrm{dif}_{\sigma_0,\sigma_1}$ where

$$\mathrm{dif}_{\sigma_0,\sigma_1} = \min\{h : \forall U \in \{0,1\}^h \text{ graph } \Omega_{\sigma_0\sigma_1,U} \text{ diffuses }\}.$$

For the selection of σ_0, σ_1 we minimize $\mathrm{dif}_{\sigma_0,\sigma_0}$, $\mathrm{dif}_{\sigma_1,\sigma_1}$, and $\mathrm{dif}_{\sigma_0,\sigma_1}$.

Additionally, we verify the randomness of selected pair permutations. Let $G = (V, E)$ be a $4t$-regular multigraph where $V = V(\Omega_{\sigma_0\sigma_0,0})$ and $E = E(\Omega_{\sigma_0\sigma_0,0}) \cup E(\Omega_{\sigma_1\sigma_1,0}) \cup E(\Omega_{\sigma_0^{-1}\sigma_0^{-1},0}) \cup E(\Omega_{\sigma_1^{-1}\sigma_1^{-1},0})$. We require the second largest eigenvalue λ_0 of adjacency matrix of multigraph G to be small. According to the Expander mixing lemma, we have

$$\left| E(S,T) - \frac{d|S||T|}{n} \right| \leq \lambda_0 \sqrt{|S||T|}.$$

This criterion helps us to select pair of permutations which minimizes $\mathrm{dif}_{\sigma_0,\sigma_1}$, where $E(S, T)$ denotes the number of edges between S and T, and d is the degree of each vertex (in our case $4t$), and n is the total number of vertices (in our case $2k$). It allows to quantify the diffusion coming from the the data dependent permutation layer, as the high number of edges means the higher diffusion. The Expander mixing lemma gives an estimate, on how far we are from an optimum ($\Omega_{\sigma_0\sigma_1,U}$ behaving like a random d-regular graph). We refer the reader to [3] for further analysis of ARMADILLO family based on expander graphs.

3 The Security Analysis of ARMADILLO3 Function

3.1 Differential and Linear Cryptanalysis

The differential and linear cryptanalysis is complicated by the fact, that the attacker does not know the sequence Y, i.e., the sequence in which permutations σ_0 and σ_1 are selected. In the differential cryptanalysis, the attacker looks for differentials which propagate with a high probability through the cipher. Since the permutation Y is not fixed while it varies according to the input X and the input H, the input Y is hard to predict, i.e., it is hard for an attacker to find a good differential path and mount differential cryptanalysis. Similarly, the linear relations between input and output of ARMADILLO3-A1/4 depend on the value Y which is unpredictable, and therefore obtaining a good linear characteristic is hard. Moreover, the S-boxes are selected to provide good security guarantees against both differential and linear cryptanalysis, therefore even if the value Y is known to the attacker, the differential/linear cryptanalysis should be impossible. From LAT, resp. DDT we can see that any linear characteristic resp. differential have probability at most $\frac{1}{4}$. Therefore, any differential/linear characteristic over $(h + 1)$ rounds will have a probability at most $\frac{1}{4}$ from the construction of the S-box. Consequently, any $(h + 1) \cdot g$ round differential characteristic will have probability 2^{-2g} and any $(h+1)\cdot g$ round linear characteristic will have a correlation of at most 2^{-2g}. In the case of ARMADILLO3-A1/4, we have $k=128$, and $h=4$. Therefore, the best differential/linear characteristic has probability probability at most $2^{-2\frac{128}{5}} \approx 2^{-51}$. The security margin is obtained from the fact that the attacker have to know the values in the control register to be able to use such differential/linear characteristics.

3.2 High Order Differentials and Algebraic Attacks

The number of S-boxes and the structure of the selected permutations ensure that the degree of underlying ANF equations is close to maximum and the data-dependency of the design ensures that these equations do not have any simple structural properties.

3.3 Statistical Saturation Attacks

The statistical saturation attacks were introduced in [6] against PRESENT [5]. Such attack is based on low diffusion trails in the linear layer of PRESENT. However, as the low diffusion trails are not constant and the permutations are selected in such a way that the distance of σ_0, σ_1 and their compositions $\sigma_0\sigma_0$, $\sigma_0\sigma_1$, $\sigma_1\sigma_0$, and $\sigma_1\sigma_1$ are maximal, the low diffusion trace changes substantially for different sequences Y. As the value Y depends both on the secret key C and the challenge U, and since $\mathsf{Y} = P(\mathsf{X}, \mathsf{H}||\mathsf{X})$, we expect that it would be difficult for an attacker to control the low diffusion paths and to utilize them at the same time.

3.4 The Internal Collision Attack

The attacker can try to force an internal collision. The internal collision can appear during the computation, since it is possible to find a pair (Y, Y') such that $Y_{\sigma_0} = Y'_{\sigma_1}$. Let consider a single step of ARMADILLO3, i.e., $P(p||b, Z) = P(p, S(Z_{\sigma_b}))$ and let consider how can we obtain an internal collision $S(Z_{\sigma_b}) = S(Z'_{\sigma_{b'}})$. We need to have $Z' = Z_{\sigma_b^{-1}\sigma_{b'}}$ as the substitution layer is bijective. This means that either $Z = Z'$ for $b = b'$ or we have a prescribed relation between Z and Z'. This allows the attacker to force the value Y (from the preprocessing phase) to be the same for different inputs U and U'. However, the attacker has then no control over the propagation of the difference in the computation ARMADILLO3(H||X) and ARMADILLO3(H||X′).

3.4.1 Invariant States. The relation $Z' = Z_{\sigma_b^{-1}\sigma_{b'}}$ also allows the attacker to find invariant internal state, i.e, a state W such that $W_{\sigma_0} = W_{\sigma_1}$. Therefore, the invariant state has to follow an equation $W = W_{\sigma_b^{-1}\sigma_{b'}}$ which means that bits of W are constant for each cycle of permutation ${\sigma_0}^{-1}\sigma_1$. Therefore, selecting σ_0 and σ_1 so that ${\sigma_0}^{-1}\sigma_1$ has long cycles is a good protection against these types of attacks. We note that a cycle of permutation ${\sigma_0}^{-1}\sigma_1$ is a subset of the set A, B or C. Therefore, we cannot obtain a pair of permutations, such that ${\sigma_0}^{-1}\sigma_1$ has less than $h+1$ cycles. Moreover, as the S-boxes takes input/output bits from different cycles and changes the parity and therefore if $Z'^i = Z^i_{\sigma_b^{-1}\sigma_{b'}} \implies Z'^{i+1} \neq Z^{i+1}_{\sigma_b^{-1}\sigma_{b'}}$ which means there is no invariant state in ARMADILLO3.

4 ARMADILLO3-A1/4

This section gives a concrete proposal of ARMADILLO3-A1/4 with $k = 128$ $t = 3$ and $r = 11$ and "coverage rate" $\approx \frac{1}{4}$. This instance has only 11 3×3 S-boxes, which makes it an interesting design for study by cryptographic community. We give a description of S-boxes and the pair of permutations. We argue about the security of ARMADILLO3 based on the properties of Hierarchical Permutations and the low second largest eigenvalue of the selected pair of permutations.

4.1 The S-box Layer of ARMADILLO3

The function S is defined as follows:

$$S(z_1\|z_2\|z_3\|\ldots\|z_{31}\|z_{32}\|z_{33}\|\ldots\|z_{128}) = s(z_1, z_2, z_3)\|\ldots\|s(z_{31}, z_{32}, z_{33})\|z_{34}\|\ldots\|z_{128}$$

The reader should notice that the indices covered by S-boxes correspond to the sets A_0, B_0, and C_0 of the Hierarchical Permutation.

Table 1. ARMADILLO3 variants with "coverage" $\frac{1}{4}$

version	k	c	m	r	t
ARMADILLO3-A1/4	128	80	48	11	3
ARMADILLO3-B1/4	192	128	64	16	3
ARMADILLO3-C1/4	240	160	80	20	3
ARMADILLO3-D1/4	288	192	96	24	3
ARMADILLO3-E1/4	384	256	128	32	3

Table 2. Implementation results with throughput at 1MHz, using 0.35μm

Algorithm	Block	Key	Area	Time	Cell power
	(bits)	(bits)	(GE)	(cycles/block)	(mW)
ARMADILLO3-A1/4	48	80	4048	176	60
ARMADILLO3-B1/4	64	128	6065	256	89
ARMADILLO3-C1/4	80	160	7576	320	110
ARMADILLO3-D1/4	96	192	9133	384	134
ARMADILLO3-E1/4	128	256	12239	512	177

Table 3. Implementation comparison for hash functions with throughput at 100 kHz

Algorithm	Digest	Block	Area	Time	Throughput	Logic	FOM
	(bits)	(bits)	(GE)	(cycles/block)	(kb/s)	(μm)	(nanobit/cycle.GE^2)
ARMADILLO2-A	80	48	4 030	44	109	0.18	67.17
PHOTON [13]	80	20	1168	132	775	0.18	59.27
ARMADILLO3-A1/4	80	48	4 302	44	109	0.35	58.95
ARMADILLO2-A	80	48	2 923	176	27	0.18	31.92
ARMADILLO3-A1/4	80	48	2 991	176	27	0.35	30.49
PHOTON [13]	80	20	865	708	144	0.18	20.12
KECCAK-f[400][16]	128	128	5 090	32	200	0.18	110.41
H-PRESENT-128[5]	128	128	4 256	32	200	0.18	110.41
ARMADILLO2-B	128	64	6 025	64	1000	0.18	27.55
ARMADILLO3-B1/4	128	64	6 409	64	1000	0.35	24.34
PHOTON [13]	128	16	1 708	156	422	0.18	15.06
ARMADILLO2-B	128	64	4 353	256	250	0.18	13.19
ARMADILLO3-B1/4	128	64	4 449	256	250	0.35	12.62
MD5 [12]	128	512	8 400	612	83.66	0.13	11.86
U_QUARK[2]	136	8	2 392	68	476	0.18	8.51
PHOTON [13]	128	16	1 122	996	66	0.18	5.48
U_QUARK[2]	136	8	1 379	544	87	0.18	3.20
ARMADILLO2-C	160	80	7 492	80	100	0.18	17.81
PHOTON [13]	160	36	2 117	180	731	0.18	17.01
ARMADILLO3-C1/4	160	80	7 972	80	100	0.35	15.72
ARMADILLO2-C	160	80	5 406	320	250	0.18	8.55
ARMADILLO3-C1/4	160	80	5 526	320	250	0.35	8.18
D_QUARK[2]	176	16	2 819	88	616	0.18	8.08
PHOTON [13]	160	36	1 396	1332	98	0.18	5.28
SHA-1 [12]	160	512	8 120	1 274	40.18	0.35	6.10
D_QUARK[2]	176	16	1 702	704	76	0.18	2.77
ARMADILLO2-D	192	96	8 999	96	100	0.18	12.35
ARMADILLO3-D1/4	192	96	9 575	96	100	0.35	10.90
C-PRESENT-192[5]	192	192	8 048	108	59.26	0.18	9.15
ARMADILLO2-D	192	96	6 554	384	25	0.18	5.82
ARMADILLO3-D1/4	192	96	6 698	384	25	0.35	5.37
MAME [26]	256	256	8 100	96	266.67	0.18	40.64
ARMADILLO2-E	256	128	11 914	128	100	0.18	7.05
ARMADILLO3-E1/4	256	128	12 682	128	100	0.35	6.22
SHA-256 [12]	256	512	10 868	1 128	45.39	0.35	3.84
ARMADILLO2-E	256	128	8 653	512	25	0.18	3.34
ARMADILLO3-E1/4	256	128	8 845	512	25	0.35	3.19
PHOTON [13]	256	32	4 362	156	650	0.18	2.94
PHOTON [13]	256	32	2 177	996	1034	0.18	1.85

Table 4. Implementation comparison for encryption with throughput at 100 kHz

Algorithm	Key	Block	Area	Time	Throughput	Logic	FOM
	(bits)	(bits)	(GE)	(cycles/block)	(kb/s)	(μm)	(nanobit/cycle.GE^2)
PRESENT-80 [5]	80	64	1 570	32	200	0.18	811.39
Grain [14]	80	1	1 294	1	100	0.13	597.22
KTANTAN64 [8]	80	64	927	128	50	0.13	581.85
KATAN64 [8]	80	64	1 269	85	75	0.13	467.56
ARMADILLO2-A	80	128	4 030	44	291	0.18	179.12
ARMADILLO3-A1/4	80	128	4 302	44	291	0.35	157.19
Trivium [9]	80	1	2 599	1	100	0.13	148.04
PRESENT-80 [5]	80	64	1 075	563	11	0.18	98.37
ARMADILLO2-A	80	128	2 923	176	73	0.18	85.12
ARMADILLO3-A1/4	80	128	2 991	176	73	0.35	81.30
PRESENT-128 [5]	128	64	1 886	32	200	0.18	562.27
HIGHT [15]	128	64	3 048	34	189	0.25	202.61
TEA [25]	128	64	2 355	64	100	0.18	180.31
ARMADILLO2-B	128	192	6 025	64	300	0.18	82.64
ARMADILLO3-B1/4	128	192	6 409	64	300	0.35	73.03
ARMADILLO2-B	128	192	4 353	256	75	0.18	39.58
ARMADILLO3-B1/4	128	192	4 449	256	75	0.35	37.89
AES-128 [11]	128	128	3 400	1 032	12	0.35	10.73

Table 5. Permutation σ_0

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
↓	↓	↓	↓	↓	↓	↓	↓	↓	↓	↓	↓	↓	↓	↓	↓
33	61	92	34	52	86	36	54	89	41	59	93	39	53	84	94
16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
↓	↓	↓	↓	↓	↓	↓	↓	↓	↓	↓	↓	↓	↓	↓	↓
55	88	35	57	90	37	58	85	38	56	82	40	51	91	83	60
32	33	34	35	36	37	38	39	40	41	42	43	44	45	46	47
↓	↓	↓	↓	↓	↓	↓	↓	↓	↓	↓	↓	↓	↓	↓	↓
87	50	45	43	49	42	47	44	48	46	78	69	70	73	79	63
48	49	50	51	52	53	54	55	56	57	58	59	60	61	62	63
↓	↓	↓	↓	↓	↓	↓	↓	↓	↓	↓	↓	↓	↓	↓	↓
72	75	67	81	71	64	76	66	77	62	65	80	68	74	118	119
64	65	66	67	68	69	70	71	72	73	74	75	76	77	78	79
↓	↓	↓	↓	↓	↓	↓	↓	↓	↓	↓	↓	↓	↓	↓	↓
100	122	127	107	108	117	109	121	111	105	110	98	97	96	120	103
80	81	82	83	84	85	86	87	88	89	90	91	92	93	94	95
↓	↓	↓	↓	↓	↓	↓	↓	↓	↓	↓	↓	↓	↓	↓	↓
99	115	116	123	126	124	114	113	125	95	106	104	101	102	112	0
96	97	98	99	100	101	102	103	104	105	106	107	108	109	110	111
↓	↓	↓	↓	↓	↓	↓	↓	↓	↓	↓	↓	↓	↓	↓	↓
24	29	2	13	6	25	16	10	32	21	15	18	1	27	7	11
112	113	114	115	116	117	118	119	120	121	122	123	124	125	126	127
↓	↓	↓	↓	↓	↓	↓	↓	↓	↓	↓	↓	↓	↓	↓	↓
17	22	19	31	9	30	4	8	12	28	5	20	26	3	23	14

Table 6. Permutation σ_1

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
↓	↓	↓	↓	↓	↓	↓	↓	↓	↓	↓	↓	↓	↓	↓	↓
34	53	88	37	61	82	35	51	86	36	58	85	41	55	94	90
16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
↓	↓	↓	↓	↓	↓	↓	↓	↓	↓	↓	↓	↓	↓	↓	↓
57	87	40	52	89	38	59	83	33	60	84	39	56	92	93	54
32	33	34	35	36	37	38	39	40	41	42	43	44	45	46	47
↓	↓	↓	↓	↓	↓	↓	↓	↓	↓	↓	↓	↓	↓	↓	↓
91	46	49	42	47	48	44	43	50	45	64	65	67	80	75	76
48	49	50	51	52	53	54	55	56	57	58	59	60	61	62	63
↓	↓	↓	↓	↓	↓	↓	↓	↓	↓	↓	↓	↓	↓	↓	↓
66	71	68	63	73	70	72	74	79	77	62	78	69	81	104	116
64	65	66	67	68	69	70	71	72	73	74	75	76	77	78	79
↓	↓	↓	↓	↓	↓	↓	↓	↓	↓	↓	↓	↓	↓	↓	↓
113	106	126	105	95	119	127	124	100	122	117	114	112	123	96	102
80	81	82	83	84	85	86	87	88	89	90	91	92	93	94	95
↓	↓	↓	↓	↓	↓	↓	↓	↓	↓	↓	↓	↓	↓	↓	↓
125	120	103	110	98	99	97	111	121	115	109	118	108	101	107	25
96	97	98	99	100	101	102	103	104	105	106	107	108	109	110	111
↓	↓	↓	↓	↓	↓	↓	↓	↓	↓	↓	↓	↓	↓	↓	↓
5	18	22	21	12	16	23	4	26	32	11	0	7	30	17	29
112	113	114	115	116	117	118	119	120	121	122	123	124	125	126	127
↓	↓	↓	↓	↓	↓	↓	↓	↓	↓	↓	↓	↓	↓	↓	↓
13	15	8	24	6	20	9	14	19	31	1	3	10	27	28	2

The proposed instance of ARMADILLO3 has 3 × 3 S-boxes. The S-box satisfies the following equations. For an input (x_0, x_1, x_2) and output $(y_0, y_1, y_2) = S(x_0, x_1, x_2)$. We have

$$\begin{cases} y_0 = x_0 + x_1 + x_2 + x_0 * x_1 + 1 \\ y_1 = x_0 + x_1 + x_0 * x_2 + 1 \\ y_2 = x_0 + x_1 * x_2 + 1 \end{cases}$$

The permutation defined by the S-box expressed in decimal is a non-linear cycle: (0 7 5 3 2 4 6 1). Thus, with 3 AND, 6 XOR and 2 NOT we can implement a single ARMADILLO S-box in hardware.

4.2 The Permutation Pair

For a selection of a good pair of permutations, we create a pool of Hierarchical Permutations with low diffusion dif_σ and a small number of long cycles. Afterwards we select a pair which achieves a full diffusion in 25 rounds and the second largest eigenvalue of graph $\Omega_{\sigma_0\sigma_1,U}$ (Def. 2) is $\lambda_0 = 9.36$. The permutation σ_0 is given in Table 5 and the permutation σ_1 is given in Table 6.

4.3 Test Vectors and Implementation

We give a test vector for the ARMADILLO3-A1/4 with the S-boxes above and the permutation pair σ_0 in Table 5 and σ_1 in Table 6.

$$\text{ARMADILLO3-A1/4}(0^k) = \text{0xF89FCBAB 0x47D36AF6 0xDC51602D 0x31C3EEA1}$$

$$\text{ARMADILLO3-A1/4}(1^k) = \text{0x7C7A0E1F 0xBA9214DF 0x5FC3CD65 0x374EB994}$$

The synthesis results at 1MHz with typical 0.35μm library and 2.2V voltage supply can be found in Table 2. We give the details for several instances with "coverage" $\frac{1}{4}$. We give the figures for several variants of ARMADILLO3 depending on the size of internal state, see Table 1 for details on the variants with coverage "$\frac{1}{4}$". Concrete proposals for variants ARMADILLO3-B1/4, ARMADILLO3-C1/4, ARMADILLO3-D1/4, and ARMADILLO3-E1/4 are omitted due to the lack of space.

5 Conclusion

We introduced a new hardware oriented class of cryptographic primitives ARMADILLO3. Our design of ARMADILLO3 is based on data-dependent permutations and a reduced size substitution layer. To meet the criteria for good confusion and diffusion layers, we introduce the concept of Hierarchical Permutations. Such permutations give guarantees, that the diffusion is fast despite the reduced substitution layer. The applications for ARMADILLO3 include MACs, hashing and PRNG. We propose an instance ARMADILLO3-A1/4 to encourage the study of ARMADILLO3. The ARMADILLO3-A1/4 consists of a pair of carefully selected Hierarchical Permutations and 11 3 × 3 S-boxes.

References

1. Abdelraheem, M.A., Blondeau, C., Naya-Plasencia, M., Videau, M., Zenner, E.: Cryptanalysis of ARMADILLO2. In: Lee, D.H., Wang, X. (eds.) ASIACRYPT 2011. LNCS, vol. 7073, pp. 308–326. Springer, Heidelberg (2011)
2. Aumasson, J.-P., Henzen, L., Meier, W., Naya-Plasencia, M.: QUARK: A lightweight hash. In: Mangard, S., Standaert, F.-X. (eds.) CHES 2010. LNCS, vol. 6225, pp. 1–15. Springer, Heidelberg (2010)
3. Badel, S., Dağtekin, N., Nakahara Jr., J., Ouafi, K., Reffé, N., Sepehrdad, P., Sušil, P., Vaudenay, S.: ARMADILLO: A Multi-purpose Cryptographic Primitive Dedicated to Hardware. In: Mangard, S., Standaert, F.-X. (eds.) CHES 2010. LNCS, vol. 6225, pp. 398–412. Springer, Heidelberg (2010)
4. Bertoni, G., Daemen, J., Peeters, M., Van Assche, G.: On the indifferentiability of the sponge construction. In: Smart, N.P. (ed.) EUROCRYPT 2008. LNCS, vol. 4965, pp. 181–197. Springer, Heidelberg (2008)
5. Bogdanov, A., Leander, G., Paar, C., Poschmann, A., Robshaw, M.J.B., Seurin, Y.: Hash Functions and RFID Tags: Mind the Gap. In: Oswald, E., Rohatgi, P. (eds.) CHES 2008. LNCS, vol. 5154, pp. 283–299. Springer, Heidelberg (2008)

6. Collard, B., Standaert, F.X.: A Statistical Saturation Attack against the Block Cipher PRESENT. In: Fischlin, M. (ed.) CT-RSA 2009. LNCS, vol. 5473, pp. 195–210. Springer, Heidelberg (2009)
7. Damgård, I.B.: A Design Principle for Hash Functions. In: Brassard, G. (ed.) CRYPTO 1989. LNCS, vol. 435, pp. 416–427. Springer, Heidelberg (1990)
8. De Cannière, C., Dunkelman, O., Knežević, M.: KATAN and KTANTAN — A Family of Small and Efficient Hardware-Oriented Block Ciphers. In: Clavier, C., Gaj, K. (eds.) CHES 2009. LNCS, vol. 5747, pp. 272–288. Springer, Heidelberg (2009)
9. De Cannière, C., Preneel, B.: Trivium - A Stream Cipher Construction Inspired by Block Cipher Design Principles. eSTREAM, ECRYPT Stream Cipher (2005)
10. Eastlake, D.E., Jones, P.E.: US Secure Hash Algorithm 1 (SHA1), http://www.ietf.org/rfc/rfc3174.txt?number=3174
11. Feldhofer, M., Dominikus, S., Wolkerstorfer, J.: Strong authentication for RFID systems using the AES algorithm. In: Joye, M., Quisquater, J.-J. (eds.) CHES 2004. LNCS, vol. 3156, pp. 357–370. Springer, Heidelberg (2004)
12. Feldhofer, M., Rechberger, C.: A Case Against Currently Used Hash Functions in RFID Protocols. In: Meersman, R., Tari, Z., Herrero, P. (eds.) OTM Workshops 2006. LNCS, vol. 4277, pp. 372–381. Springer, Heidelberg (2006)
13. Guo, J., Peyrin, T., Poschmann, A.: The PHOTON Family of Lightweight Hash Functions. In: Rogaway, P. (ed.) CRYPTO 2011. LNCS, vol. 6841, pp. 222–239. Springer, Heidelberg (2011)
14. Hell, M., Johansson, T., Maximov, A., Meier, W.: The Grain family of stream ciphers. In: Robshaw, M., Billet, O. (eds.) New Stream Cipher Designs. LNCS, vol. 4986, pp. 179–190. Springer, Heidelberg (2008)
15. Hong, D., Sung, J., Hong, S., Lim, J., Lee, S., Koo, B.-S., Lee, C., Chang, D., Lee, J., Jeong, K., Kim, H., Kim, J., Chee, S.: HIGHT: A New Block Cipher Suitable for Low-Resource Device. In: Goubin, L., Matsui, M. (eds.) CHES 2006. LNCS, vol. 4249, pp. 46–59. Springer, Heidelberg (2006)
16. Kavun, E.B., Yalcin, T.: A Lightweight Implementation of Keccak Hash Function for Radio-Frequency Identification Applications. In: Ors Yalcin, S.B. (ed.) RFIDSec 2010. LNCS, vol. 6370, pp. 258–269. Springer, Heidelberg (2010)
17. Knudsen, L., Leander, G., Poschmann, A., Robshaw, M.J.B.: PRINTcipher: A Block Cipher for IC-Printing. In: Mangard, S., Standaert, F.-X. (eds.) CHES 2010. LNCS, vol. 6225, pp. 16–32. Springer, Heidelberg (2010)
18. Merkle, R.C.: A Fast Software One-Way Hash Function. J. Cryptology 3(1), 43–58 (1990)
19. Naya-Plasencia, M., Peyrin, T.: Practical cryptanalysis of ARMADILLO2. In: Canteaut, A. (ed.) FSE 2012. LNCS, vol. 7549, pp. 146–162. Springer, Heidelberg (2012)
20. Ouafi, K., Vaudenay, S.: Pathchecker: An RFID application for tracing products in Supply-chains. In: Batina, L. (ed.) Proceedings of RFIDSec 2009 (2009)
21. Federal Information Processing Standards Publications. Advanced Encryption Standard. Technical Report FIPS PUB 197, National Institute of Standards and Technology (November 2001)
22. Sepehrdad, P., Sušil, P., Vaudenay, S.: Fast Key Recovery Attack on ARMADILLO1 and Variants. In: Prouff, E. (ed.) CARDIS 2011. LNCS, vol. 7079, pp. 133–150. Springer, Heidelberg (2011)
23. Shamir, A.: SQUASH – A new MAC with provable security properties for highly constrained devices such as RFID tags. In: Nyberg, K. (ed.) FSE 2008. LNCS, vol. 5086, pp. 144–157. Springer, Heidelberg (2008)

24. Hoory, S., Linial, N., Wigderson, A.: Expander graphs and their applications. Bulletin of the AMS 43(4), 439–561 (2006)
25. Wheeler, D., Needham, R.: TEA, a Tiny Encryption Algorithm (1995)
26. Yoshida, H., Watanabe, D., Okeya, K., Kitahara, J., Wu, H., Küçük, Ö., Preneel, B.: MAME: A Compression Function with Reduced Hardware Requirements. In: Paillier, P., Verbauwhede, I. (eds.) CHES 2007. LNCS, vol. 4727, pp. 148–165. Springer, Heidelberg (2007)

Improving Side-Channel Analysis with Optimal Linear Transforms

David Oswald and Christof Paar

Horst Görtz Institute for IT Security
Ruhr-University Bochum, Germany
{david.oswald,christof.paar}@rub.de

Abstract. Pre-processing techniques are widely used to increase the success rate of side-channel analysis when attacking (protected) implementations of cryptographic algorithms. However, as of today, the according steps are usually chosen heuristically. In this paper, we present an analytical expression for the correlation coefficient after applying a linear transform to the side-channel traces. Doing so, we are able to precisely quantify the influence of a linear filter on the result of a correlation power analysis. On this basis, we demonstrate the use of optimisation algorithms to efficiently and methodically derive "optimal" filter coefficients in the sense that they maximise a given definition for the distinguishability of the correct key candidate. We verify the effectiveness of our methods by analysing both simulated and real-world traces for a hardware implementation of the AES.

Keywords: side-channel analysis, linear filtering, countermeasures, pre-processing, attacks on protected implementations, DPA contest v2.

1 Introduction

The use of Digital Signal Processing (DSP) to facilitate attacks based on Side-Channel Analysis (SCA) or to reduce the number of needed measurements (*traces*) has been demonstrated to be effective in numerous publications (cf. Sect. 1.1). Methods such as Differential Power Analysis (DPA) [12] or Correlation Power Analysis (CPA) [5] often benefit from prior signal processing, e.g., Finite Impulse Response (FIR) filtering. Yet, the precise effect of the pre-processing steps on the success rate of SCA has, to our knowledge, not been precisely quantified. In this paper, we utilise analytical properties of the correlation coefficient in order to derive a more systematic approach for optimising the – so far mostly heuristically selected – pre-processing parameters. From a designer's point of view, our method helps to objectively estimate the amount of leakage an adversary might extract by means of filtering.

1.1 Related Work

One of the first examples of DSP being applied to SCA can be found in [14]: the authors mention the use of a matched filter to increase the Signal-to-Noise

S. Mangard (Ed.): CARDIS 2012, LNCS 7771, pp. 219–233, 2013.

Ratio (SNR) and thus the height of a DPA peak. In [6], Clavier et al. propose to perform comb filtering, i.e., average measurement samples from multiple clock cycles in order to increase the success rate of a DPA in the presence of random process interrupts.

Especially for practical attacks on cryptographic devices, DSP pre-processing is often mandatory, for instance because of uncorrelated noise due to non-cryptographic parts of an Integrated Circuit (IC). In [2], digital filtering helped to isolate the frequency components containing the side-channel leakage of a cryptographic co-processor. Similarly, the attacks on the bitstream encryption mechanism of Xilinx Field Programmable Gate Arrays (FPGAs) required the removal of an interfering signal [15].

For SCA utilising the electro-magnetic (EM) emanation of a cryptographic device, digital filters have been applied to isolate the frequencies containing the side-channel leakage [1]. In the context of cryptographic Radio Frequency Identification (RFID) devices, DSP pre-processing steps have been shown to be necessary [18]. Accordingly, the real-world attacks on the Mifare DESFire MF3ICD40 RFID smartcard described in [11,17] involve several filter operations.

Yet, there is almost no systematic research how DSP operations such as filtering affect the outcome of SCA. In [3], the authors propose an approach to automatically determine appropriate bandpass filters, using CPA as a block algorithm that is repeatedly executed for different choices for the filter coefficients. In general, however, filtering is seen as a completely separate pre-processing step, and the parameters are usually chosen manually.

1.2 Contribution of This Paper

In this paper, we intend to improve on the current approach for devising suitable DSP operations for SCA. More precisely, we examine the effect of linear transforms (which cover amongst others linear filters) on the result of a CPA. The remainder of this paper is organised as follows: in Sect. 2, we briefly review CPA and linear filtering. Then, we introduce a matrix notation that provides a closed form for the correlation coefficient after a linear transform in Sect. 2.1. On this basis, we propose the use of numerical optimisation to determine "good" filter coefficients in Sect. 3. In Sect. 4 and Sect. 5, we compare our approach to normal CPA and to a CPA in the frequency domain, using simulated and real-world measurements (provided in the second DPA contest [7]), respectively. Finally, we conclude in Sect. 6.

2 CPA and Linear Transforms

In the following, we assume the usual setting of SCA: the adversary sends freely chosen input data to a Device Under Test (DUT) (that performs a cryptographic operation on this data) and obtains the corresponding output. The computation done by the DUT involves some secret information (in the following referred to as a key) $k^{\mathrm{dut}} \in K$ (with K the set of all possible keys) that the adversary aims to obtain by means of SCA.

Notation. The process of performing a CPA can be divided into two steps: in the measurement phase, the adversary has physical access to the DUT and records some side-channel signal (e.g., the power consumption or the electromagnetical emanation during the cryptographic computation) that is related to the processed data. This step is repeated N times with varying input data M_i, yielding N time-discrete waveforms $x_i\left(t\right)$ with T points each. In the evaluation phase, the key is recovered by fixing a (small) subset $\mathcal{K}_{cand} \subseteq \mathcal{K}$ and considering all key candidates $k \in \mathcal{K}_{cand}$: for each $k \in \mathcal{K}_{cand}$ and for each $i \in \{0, \ldots, N-1\}$, a hypothesis $V_{k,\,i}$ on the value of some intermediate is computed. Using a power model f, this value is then mapped to $h_{k,\,i} = f\left(V_{k,\,i}\right)$ to describe the physical process that causes the side-channel leakage. In practice, for DUTs such as FPGAs or Microcontrollers (μCs), the power model is often either the Hamming Weight (HW) or Hamming Distance (HD) model [13].

$h_{k,\,i}$ and $x_i\left(t\right)$ are treated as observations of discrete random variables. In order to detect the dependency between $h_{k,\,i}$ and $x_i\left(t\right)$, the *correlation coefficient* $\rho_k\left(t\right)$ (for each point in time $t \in \{0, \ldots, T-1\}$ and each key candidate $k \in \mathcal{K}_{cand}$) is given as $\rho_k\left(t\right) = {}^{cov(x(t),\,h_k)}\!/\!\sqrt{var(x(t))var(h_k)}$ with $var\left(\cdot\right)$ indicating the sample variance and $cov\left(\cdot,\,\cdot\right)$ the sample covariance according to the standard definitions [24]. The key candidate $\hat{k}$ with the maximum correlation $\hat{k} = \arg\max_{k,\,t} \rho_k\left(t\right)$ is assumed to be the secret key k^{dut} used by the DUT.

Linear Filters. As mentioned in Sect. 1.1, a CPA in the time domain is often preceded by a linear FIR filter: for example, a bandpass or bandstop filter may be used to isolate or remove certain frequencies present in a side-channel signal [2]. Integrating over multiple clock cycles can be interpreted as a comb filter [6].

An $S-1$-th order FIR filter is defined by S coefficients $a_i \in \mathbb{R}$, $i = 0 \ldots S-1$. The response $y\left(t\right)$ of the filter to the input signal $x\left(t\right)$ is computed as a (sliding) weighted sum of the points of the input signal, i.e., $y\left(t\right) = \sum_{i=0}^{S-1} a_i x\left(t-i\right)$.

In the following Sect. 2.1, we show how to compute the correlation coefficient between a prediction h_k and an arbitrary weighted sum like an FIR filter, given the "raw" correlation for each point in time and the covariance matrix of the input signal. To this end, we apply a matrix notation according to [10].

2.1 Matrix Notation

To improve the readability, we drop the index k for the key candidate in this section. All involved quantities are represented as vectors or matrices. A trace $x_i\left(t\right)$ is hence denoted as $T \times 1$ vector $\boldsymbol{x_i}$. $\Sigma_{\boldsymbol{xx}}$ is the $T \times T$ sample covariance matrix over all traces according to the standard definition.

For the purposes of this paper, the prediction is a scalar h_i, i.e., a 1×1 vector. Note that this restriction is not mandatory: the prediction could also be extended to a $P \times 1$ vector $\boldsymbol{h_i}$, for example to handle multiple bits of one prediction separately. Again, $\Sigma_{\boldsymbol{hh}}$ is the $P \times P$ covariance matrix. For the scalar case $P = 1$, this reduces to the usual variance.

Finally, $\Sigma_{\boldsymbol{xh}}$ is the $T \times P$ covariance matrix between $\boldsymbol{x}$ and $\boldsymbol{h}$. For $P = 1$, this corresponds to the covariance term in the denominator of the traditional formula for the correlation coefficient. Then, given a $T \times 1$ weight vector $\boldsymbol{a}$ and a $P \times 1$ weight vector $\boldsymbol{b}$, a *closed form* for the correlation coefficient between the dot products $\boldsymbol{a} \cdot \boldsymbol{x}_i$ and $\boldsymbol{b} \cdot \boldsymbol{h}_i$ is given by Equation 1 [10].

$$\rho_{\boldsymbol{xh}}\left(\boldsymbol{a},\, \boldsymbol{b}\right) = \frac{\boldsymbol{a}^T \cdot \Sigma_{\boldsymbol{xh}} \cdot \boldsymbol{b}}{\sqrt{\boldsymbol{a}^T \cdot \Sigma_{\boldsymbol{xx}} \cdot \boldsymbol{a}}\sqrt{\boldsymbol{b}^T \cdot \Sigma_{\boldsymbol{hh}} \cdot \boldsymbol{b}}} \tag{1}$$

This representation is fully equivalent to performing the dot products first (as a pre-processing step) and then computing the correlation coefficient on the pre-processed data. In particular, $\boldsymbol{a}$ can be seen as the coefficients of an FIR filter that is applied to each trace separately. Similarly, $\boldsymbol{b}$ corresponds to an arbitrary weighted sum of e.g. several bits of a predicted intermediate. As stated above, for the purposes of this work, we assume a scalar prediction h_i and hence set $P = 1$ and $b = 1$ for the remainder of this paper. As a side note, in order to obtain the unfiltered correlation coefficient at time index $t = 0, 1, \ldots$, only the t'th entry of $\boldsymbol{a}$ has to be set to a non-zero value, i.e., $\boldsymbol{a} = (1\,0\,0\,\ldots\,0)\,,\,(0\,1\,0\,\ldots,0)\,,$ and so on.

Note that Equation 1 can be naturally extended to incorporate a *transform matrix* rather than a vector. In this case, $\boldsymbol{a}$ becomes a $T \times S$ matrix A formed by S different column vectors $\boldsymbol{a}_s$, $s \in 0, \ldots S-1$. The $S \times 1$ correlation coefficient vector $\boldsymbol{\rho}_{\boldsymbol{xh}}\left(A,\, \boldsymbol{b}\right)$ is then given by evaluating Equation 1 for all $\boldsymbol{a}_s$ and concatenating the results. This form covers (amongst other linear transforms) any FIR filter: A consists of the filter coefficient vector that is shifted by s positions for row s, that is, A is a Toeplitz matrix [21].

Computational Complexity. The form of Equation 1 is useful when the correlation coefficient is to be computed for fixed $\boldsymbol{x}_i$ and h_i but for (many) different $\boldsymbol{a}$: first computing $\boldsymbol{a} \cdot \boldsymbol{x}_i$ for each trace and then evaluating the traditional formula for the correlation coefficient has a complexity of $\mathcal{O}\left(N\,T\right)$. For L different choices $\boldsymbol{a}^l$, $l = 0 \ldots L-1$, the overall effort is thus $\mathcal{O}\left(L\,N\,T\right)$. In contrast, to evaluate Equation 1, the computation of the covariance matrices needs $\mathcal{O}\left(N\,\left(T^2 + T\right)\right) = \mathcal{O}\left(N\,T^2\right)$ operations. The post-processing to obtain the desired correlation coefficients for all $\boldsymbol{a}^l$ is then $\mathcal{O}\left(L\,T^2\right)$ (which is independent of N). Hence, the total complexity is $\mathcal{O}\left(N\,T^2 + L\,T^2\right)$.

Complexity Reduction in Special Cases. The main drawback of Equation 1 is that it requires the covariance matrix $\Sigma_{\boldsymbol{xx}}$. For large T, the estimation of this matrix is problematic due to issues with the computational complexity. Thus, the application of the closed form of the correlation may become infeasible. Incidentally, the statistical efficiency is not an issue in this case — Equation 1 does not involve the inverse of $\Sigma_{\boldsymbol{xx}}$ and is fully valid for small sample size N.

Still, for a filter of order $S-1$, $\boldsymbol{a}$ only has S consecutive non-zero coefficients. Hence, in this case, it is sufficient to estimate $\Sigma_{\boldsymbol{xx}}$ as a band matrix with a bandwidth of S. The computational complexity is then reduced to $\mathcal{O}\left(S\,T\right)$, i.e., linear with respect to the length of the traces. Finally, for the optimisation approach proposed in the following Sect. 3, $\Sigma_{\boldsymbol{xx}}$ can be factored out when determining $\boldsymbol{a}$. Hence, if estimating $\Sigma_{\boldsymbol{xx}}$ becomes prohibiting for large T, one may still utilise the traditional approach and compute the dot products $\boldsymbol{a} \cdot \boldsymbol{x}_i$ before the CPA once the optimal $\boldsymbol{a}$ has been found.

3 Optimal Linear Transforms for CPA

Given the closed-form expression for the transformed correlation coefficient of Equation 1, we propose a method to find "optimal" filter coefficients $\boldsymbol{a}$ in the sense that the filter maximises the distinguishability of the correct key candidate.

To achieve this goal, we regard Equation 1 as a multivariate function in $\boldsymbol{a}$ and employ standard numerical optimisation algorithms. As we aim to maximise the distinguishability rather than the correlation itself, a suitable optimisation criterion has to be defined. We assume a (semi-)profiled scenario in which an adversary possesses an instance of the DUT for which he knows the secret key. Note that the adversary is not necessarily able to change the key — only the knowledge of the correct key is required for the optimisation of the filter coefficients. In contrast to template attacks, which have been shown to be highly sensitive to process variations [19], we expect the filter coefficients to be less sensitive in this regard. This conjecture is based on the fact that a filter modifies the frequency spectrum (which should be less device-dependent than e.g. the signal amplitude), while the actual key recovery is still carried out by a (more robust) differential technique like CPA.

In our experiments, directly maximising Equation 1 gave rise to overfitting of the coefficients $\boldsymbol{a}$. As a result, the correlation is maximised for one specific problem instance (i.e., fixed input data, key, and traces), however, if any parameter changes, the determined coefficients no longer lead to the desired result. Hence, we devised the criterion given in Equation 2. The goal is to maximise the ratio between the absolute value of the correlation coefficient for the correct key k^{dut} and the average over the absolute value of the correlation coefficients for incorrect key candidates $k_{wrong} \in \mathcal{K}_{optim}$, $\mathcal{K}_{optim} = \mathcal{K}_{cand} \setminus \{k^{\mathrm{dut}}\}$.

$$f_{objective}\left(\boldsymbol{a}\right) = \frac{\left|\rho_{\boldsymbol{x}h_{k^{\mathrm{dut}}}}\left(\boldsymbol{a}\right)\right|}{{}^{1}\!/\!_{|\mathcal{K}_{optim}|}\left(\sum_{k \in \mathcal{K}_{optim}} \left|\rho_{\boldsymbol{x}h_k}\left(\boldsymbol{a}\right)\right|\right)} \tag{2}$$

Note that in Equation 2, every $\rho_{\boldsymbol{x}h}$ both in the numerator and denominator contains a positive factor of ${}^{1}\!/\!\sqrt{\boldsymbol{a}^T \cdot \Sigma_{\boldsymbol{xx}} \cdot \boldsymbol{a}}$ (independent of h_k) which can be cancelled. Equation 2 thus takes the form of Equation 3 (whereas the factor ${}^{1}\!/\!_{|\mathcal{K}_{optim}|}$ was left out).

$$f_{objective}\left(\boldsymbol{a}\right) = \frac{\left|{}^{1}\!/\!\sqrt{\Sigma_{h_{k^{\mathrm{dut}}} h_{k^{\mathrm{dut}}}}} \cdot \boldsymbol{a}^T \cdot \Sigma_{\boldsymbol{x}h_{k^{\mathrm{dut}}}}\right|}{\sum_{k \in \mathcal{K}_{optim}} \left|{}^{1}\!/\!\sqrt{\Sigma_{h_k h_k}} \cdot \boldsymbol{a}^T \cdot \Sigma_{\boldsymbol{x}h_k}\right|} \tag{3}$$

This eliminates the computationally most expensive part of Equation 1, namely the vector-matrix product with complexity $\mathcal{O}\left(T^2\right)$. Moreover – at least in the profiling step – the covariance matrix $\Sigma_{\boldsymbol{xx}}$ is not needed at all. Hence, as mentioned in Sect. 2.1, the optimisation of the weight coefficients can be carried out even with long traces for which computational issues make the estimation of the sample covariance matrix difficult or impossible. To numerically find an optimum of $f_{objective}$, we employ the function `fminunc` provided by the MATLAB optimization toolbox [23]. This function *minimises* a given objective function. In our case, we thus search for a minimum of $-f_{objective}$ (which is equivalent to a maximum of $f_{objective}$).

3.1 Relation to Other Techniques

Principal Component Analysis. The method of Principal Component Analysis (PCA) [22] transforms signals to a new (lower-dimensional) representation. Recently, Batina et al. proposed to use PCA as a pre-processing step for a CPA [4]. Their idea is based on the observation that a leakage signal and unrelated noise are often mapped to different principal components. PCA is a linear transform, i.e., a trace $\boldsymbol{x}_i$ is projected to the new representation using the vector-matrix product $\boldsymbol{y} = U^T \cdot \boldsymbol{x}_i$ with U the matrix of the (retained) eigenvectors of the covariance matrix $\Sigma_{\boldsymbol{xx}}$. Thus, as mentioned in Sect. 2.1, one point of the projected trace $\boldsymbol{y}$ is given as scalar product between $\boldsymbol{x}_i$ and one row of U^T. The rows of U^T can therefore be regarded as different choices for the weight vector.

Canonical Correlation Analysis. In contrast to PCA which picks principal components with maximum variance, Canonical Correlation Analysis (CCA) [10] finds a weight vector that maximises the correlation coefficient. Performing an eigenvector decomposition of the covariance matrix, CCA finds a vector $\boldsymbol{a}$ that maximises Equation 1. However, as mentioned in Sect. 3, in our experiments this lead to overfitting and resulted in non-applicable weight vectors.

SCA in the Frequency Domain. Transforming traces to the frequency domain and discarding the phase component has been shown to be beneficial for SCA [8,18]. This pre-processing step, also known as Differential Frequency Analysis (DFA), is both applicable to overcome misalignment in the traces and to spectrally isolate the leakage component. Note that the phase component is removed by taking the absolute value of the transformed traces, i.e., $|\mathrm{DFT}\left\{\boldsymbol{x}_i\right\}|$. Due to the absolute value operator, the transform is no longer linear, and hence cannot be described in terms of Equation 1 with a suitable $\boldsymbol{a}$. In Sect. 4 and Sect. 5, we provide a comparision of our proposed technique to DFA.

It should be taken into account that computing the Discrete Fourier Transform (DFT) over the complete trace is only suitable in special cases. In practice, a trace is usually split into windows of a given length which are processed separately [17]. As of today, determining the optimal window length is a somewhat heuristic process that either involves (educated) guessing or optimisation by testing many choices for the parameter.

Note that our proposed method can be combined with the frequency domain transformation. The weight vector is then applied to the transformed traces $|\mathrm{DFT}\{\boldsymbol{x}_i\}|$ and optimised according to Sect. 3. In cases where the leakage is distributed over multiple frequency bins, this approach can combine and thus presumably better utilise the overall side-channel information. Therefore, we also included this approach in our simulation and practical results in Sect. 4 and Sect. 5.

4 Simulation Results

In order to evaluate the effect of the optimised weight coefficients, we generated simulated traces for a 128-bit implementation of the Advanced Encryption Standard (AES) in MATLAB. The main purpose of this section is to demonstrate the basic effectiveness of the proposed approach. We do not aim to comprehensively examine every conceivable scenario, hence, the choice of the simulation parameters may appear somewhat arbitrary.

In our simulation, the clock frequency was set to $33.\overline{3}$ MHz, with the trace being sampled at 1 GHz. A clock cycle thus contains 30 samples. The i'th simulated trace for the clock cycle c is then generated as the sum of a "clock" signal multiplied by a leakage s_i^c and normally distributed noise as $\boldsymbol{x}_i^c = (1 + \sigma_{signal} \cdot s_i^c)\, \boldsymbol{t} + \boldsymbol{\mathcal{N}}(0, \sigma_{noise})$

We used $\sigma_{signal}^2 = \sigma_{noise}^2 = {}^1/_{1000}$. $\boldsymbol{t}$ was set to a rectangular pulse, that is, $\boldsymbol{t} = (1, 1, \ldots, 1, 0, 0, \ldots, 0)$, with the first seven entries set to 1 (corresponding to ${}^1/_4$ of the full cycle) and the remaining 23 entries set to 0. To form the final simulated trace $\boldsymbol{x}_i$, we concatenated four cycles $\boldsymbol{x}_i^c$. The leakage in the first cycle s_i^0 was generated as the HW of the 128-bit AES state after the initial key addition and SubBytes operation. In the remaining three cycles, $s_i^{1/2/3}$ was calculated as the HW of a uniformly distributed random 128-bit value.

Band-Limited Noise. To simulate the effect of a band-limited noise source, we added an additional noise term $\boldsymbol{\mathcal{N}}_{band}$ to the simulated trace. For our experiments, we selected a noise bandwidth of ±1 MHz around 24 MHz, i.e., the spectrum of $\boldsymbol{\mathcal{N}}_{band}$ is "white" between 23 and 25 MHz and zero otherwise. For a range of noise powers of $\boldsymbol{\mathcal{N}}_{band}$, we then performed (1) a time domain CPA, (2) a CPA on the frequency domain representation of the traces (cf. Sect. 3.1), (3) a time domain, and (4) a frequency domain CPA using optimised filter coefficients $\boldsymbol{a}$ as described in Sect. 3. For the profiling and the attack, we used different keys and different input data.

Fig. 1 depicts the respective (maximum) correlation for the cases (1), (2), (3), and (4) for a noise power (i.e., the average standard deviations $\sigma_{bandlimited}$) of 1. The average signal power of a trace was $\sigma_{trace} = 0.56$, i.e., the given $\sigma_{bandlimited}$ correspond to a "Trace-to-Noise Ratio" (TNR) of approximately 0.5. Because the simulated traces already contain white noise, we avoid the term SNR here.

As evident in Fig. 1, the CPA using optimised filter coefficients (3) outperforms the normal CPA (1) and the frequency domain CPA (2) in the presence of

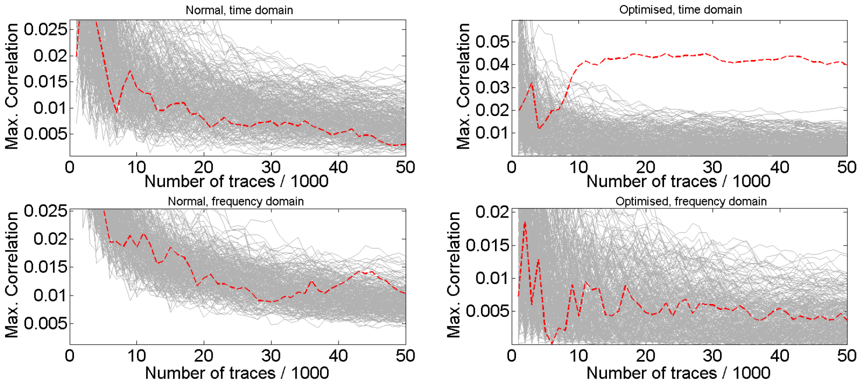


Fig. 1. Maximum correlation for band-limited noise, $\sigma_{bandlimited} = 1$. Top left: time domain (1), top right: time domain, optimised (3), bottom left: frequency domain (2), bottom right: frequency domain, optimised (4). Correct key candidate: dashed, red.

band-limited noise. Table 1 summarises the results, giving the absolute value of the correlation coefficient for the correct key after 50,000 traces (Table 1a) and the ratio between the correlation for the correct candidate and the maximum correlation for the wrong candidates (Table 1b). If this ratio is less than 1, the correct key can no longer be distinguished from the wrong candidates, i.e., the attack does not succeed.

Table 1. Comparision of evaluation methods (1) - (4) for simulated traces with band-limited noise. Best values in bold font.

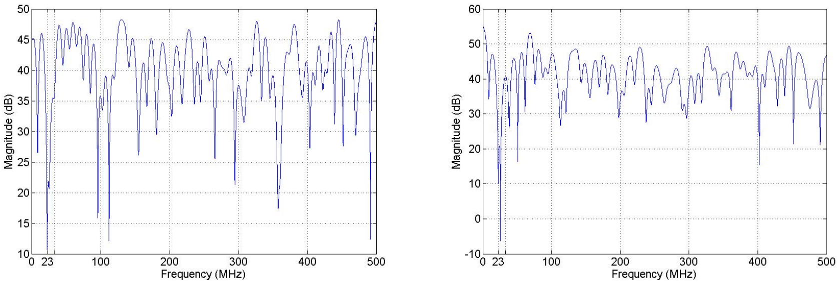
TNR	(1)	(2)	(3)	(4)
∞	**0.123**	0.09	0.051	0.017
1.7	0.009	0.019	**0.049**	0.014
1	0.005	0.012	**0.049**	0.007
0.5	0.003	0.01	**0.04**	0.004

(a) Correlation using 50k traces

TNR	(1)	(2)	(3)	(4)
∞	**3.73**	3.46	3	1.21
1.7	0.5	1	**2.88**	1.17
1	0.28	0.63	**2.72**	0.44
0.5	0.17	0.59	**1.9**	0.26

(b) Ratio between correct and maximum wrong candidate (50k traces)

With increasing $\sigma_{bandlimited}$ (i.e., decreasing TNR), the correlation coefficient for the correct key candidate is very close to or even below the correlations for the wrong key candidates using method (1), (2), or (4) after 50,000 traces. In contrast, the correlation for the correct key obtained with method (3) clearly exceeds the correlation for the wrong candidates after less than 5,000 traces in all cases. Computing the frequency response corresponding to the optimised coefficients, it turns out that the range from 23 to 25 MHz is attenuated, while the filter's transfer function is rather flat in the region of the clock frequency. The corresponding plot of the frequency response is given in Fig. 2a.



(a) Noise ($\sigma_{bandlimited} = 1$) (b) Noise + randomised timing (4 cycles)

Fig. 2. Magnitude frequency response of the optimised filter coefficients for the simulated traces

Interestingly, if no additional noise is present, method (3) provides worse distinguishability of the correct key than methods (1) and (2). In this case, the optimisation algorithm appears to overfit the weight coefficients – the coefficients then yield optimal distinguishability based on the data used in the profiling phase, but do not produce the desired effect in general. Thus, in the attack phase with a different set of traces, the distinguishabilty is reduced and not – as intended — increased. This problem could presumably be mitigated by techniques used in global optimisation, e.g., running the optimisation algorithm several times using different initial values and different subsets of the profiling data. For the purposes of this paper, we did not look further into this issue and leave it for future work.

Timing Randomisation. A randomisation of the algorithmic timing was realised by shuffling the four clock cycles for each trace, that is, by randomly selecting a uniformly distributed position for the clock cycle that corresponds to the actual AES state. For this case, we applied the same evaluation methods as above. We also combined the timing randomisation with the band-limited noise source, again considering the same range of noise powers as for the non-randomised traces.

The result for $\sigma_{bandlimited} = 1$, i.e., a TNR of approximately 0.5, is exemplarily depicted in Fig. 3. Table 2 subsumes the results like in Table 1, giving the maximum correlation and the ratio between the correlation for the correct and the highest wrong candidate for different TNRs. While the normal CPA and the frequency domain CPA fail to clearly distinguish the correct key candidate from the wrong ones after 50,000 traces, the optimisation approach determines filter coefficients that allow to extract the correct candidate after less than 5,000 traces. The according frequency response (Fig. 2b) again exhibits a band-stop characteristic eliminating the band-limited noise. In the time domain, the filter coefficients additionally resemble a comb filter, i.e., realise the averaging over multiple clock cycles [6].

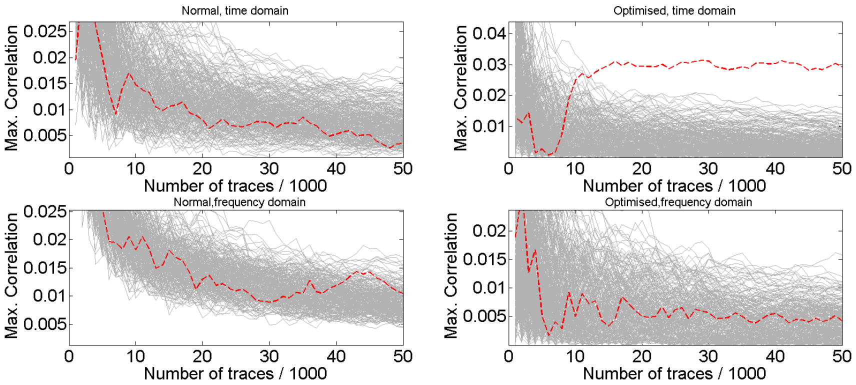


Fig. 3. Maximum correlation for jitter (4 cycles) and band-limited noise, $\sigma_{bandlimited} = 1$. Top left: time domain (1), top right: time domain, optimised (3), bottom left: frequency domain (2), bottom right: frequency domain, optimised (4). Correct key candidate: dashed, red.

Table 2. Comparision of evaluation methods (1) - (4) for simulated traces with band-limited noise and timing randomisation (4 cycles). Best values in bold font.

TNR	(1)	(2)	(3)	(4)
∞	0.038	**0.089**	0.04	0.02
1.7	0.005	0.018	**0.04**	0.014
1	0.004	0.011	**0.038**	0.007
0.5	0.003	0.01	**0.029**	0.004

(a) Correlation using 50k traces

TNR	(1)	(2)	(3)	(4)
∞	1.9	**3.42**	2.5	1.33
1.7	0.28	1	**2.22**	1.08
0.5	0.22	0.58	**2.11**	0.54
1	0.17	0.59	**1.81**	0.25

(b) Ratio between correct and maximum wrong candidate (50k traces)

5 Practical Results

In order to evaluate the efficiency of our findings in a real-world setting, we applied our methods to the traces provided in the second DPA contest [7]. The traces were recorded for a hardware implementation of the AES on the Sasebo GII [16] at a sample rate of $f_s = 5\,\text{GHz}$. We focused on the last round of the encryption process and accordingly only used the respective part from time point 2300 to 2700 of the 3253-point original traces.

For the profiling purposes, we used 15,000 raw traces of the “public database” (DPA_contest2_public_base_diff_vcc_a128_2009_12_23) belonging to the encryption with the AES key $k_{profiling}$ = 0x37d0d724d00a1248db0fead349f1c09b. For the attack phase, i.e., to evaluate the effect of the optimised filter coefficients, we used 15,000 traces for k_{attack} = 0x000000000000000003243f6a8885a308d3. These keys lead to different subkeys for the first S-Box in the final round (0xdc for $k_{profiling}$, 0x53 for k_{attack}).

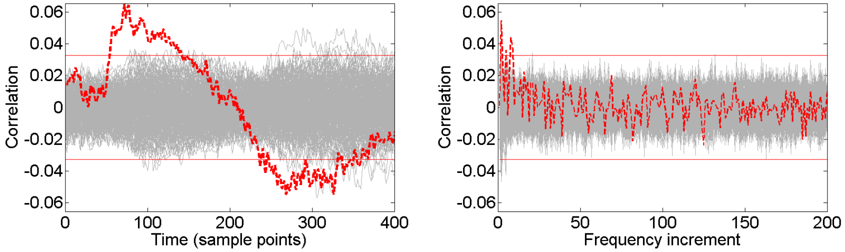


Fig. 4. Correlation coefficients (DPA contest v2 AES) for the first byte after 15,000 traces, left: time domain, right: frequency domain. Correct key candidate: dashed, red.

As a first step, we performed a standard CPA targeting the (bytewise) Hamming distance between the input of the last SubBytes operation and the encryption result, following the reference attack of the DPA contest. As depicted in Fig. 4, the highest correlation coefficient clearly occurs for the correct key candidate after 15,000 traces, with a magnitude of 0.064 in the time domain and 0.055 in the frequency domain, respectively. However, significant "ghost peaks" occur at the end of the trace (around point 300).

At this point, we want to emphasise that we primarily use the DPA contest traces to demonstrate the general effectivity of our approach in a practical setting. Hence, we did not employ the full range of evaluation metrics provided by the contest. We applied the same evaluation methods as in Sect. 4, that is, CPA (1), frequency domain CPA (2), CPA with optimised coefficients (3), and frequency domain CPA with optimised coefficients (4).

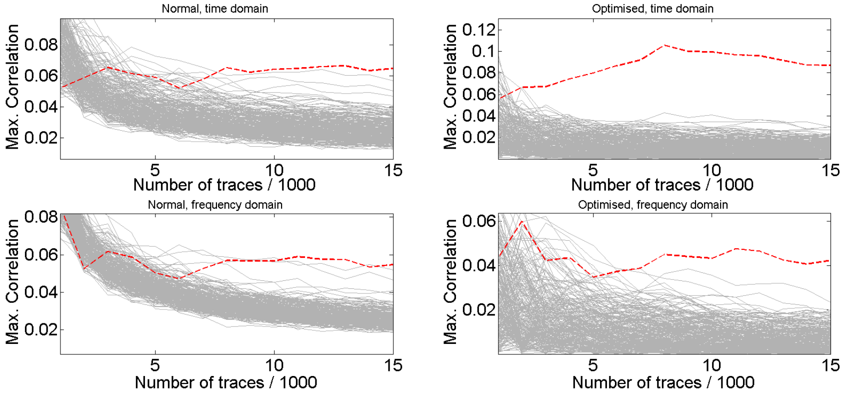
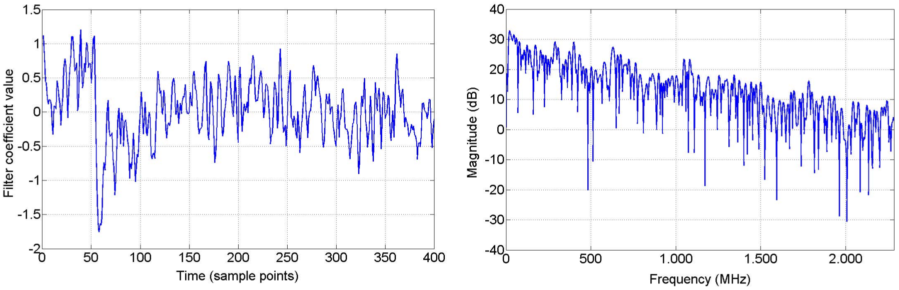


Fig. 5. Maximum correlation coefficients (DPA contest v2 AES) for the first byte. Top left: time domain (1), top right: time domain, optimised (3), bottom left: frequency domain (2), bottom right: frequency domain, optimised (4). Correct key candidate: dashed, red.



(a) Time domain

(b) Frequency response for (a)

Fig. 6. Optimised filter coefficients for the DPA contest v2 traces

As evident in Fig. 5, the optimised coefficients decrease the number of required traces both for the time and the frequency domain CPA: for the normal CPA, the correct key candidate yields the highest correlation, however, the ratio with the second highest is rather small, i.e., $^{0.064}/_{0.057} = 1.12$ in the time domain and $^{0.055}/_{0.052} = 1.06$ in the frequency domain. In contrast, with the optimised filter coefficients, these ratios are increased to $^{0.087}/_{0.03} = 2.9$ and $^{0.042}/_{0.023} = 1.83$, respectively. Accordingly, the (approximate) minimum number of traces needed to distinguish the correct and the wrong key candidate is reduced from 8,000 to 3,000 in the time domain and 11,000 to 8,000 in the frequency domain.

The optimised coefficients reduce the influence of the "ghost peaks" mentioned above on the results of the CPA, i.e., improve the distinguishability of the correct key candidate. As evident in Fig. 6a, the optimised coefficients obtained with method (3) put the highest weight on the maximum of the leakage at around time point 50, followed by decaying weight according to the shape of the correlation depicted in Fig. 4. The according frequency response (Fig. 6b) shows a lowpass characteristic in general. However, certain frequencies are selectively attenuated, for example, narrow regions around 70 MHz, 160 MHz, 227 MHz, 327 MHz, 388 MHz, 422 MHz, and 480 Mhz.

We experimentally verified that these results obtained for the first byte equivalently hold for the other bytes. Table 3 gives the ratio between the correlation for the correct and the highest wrong candidate after 15,000 traces for a normal CPA (1) and a CPA with optimised coefficients (3) in the time domain. For all

Table 3. Ratio between correct and maximum wrong candidate for normal (1) and optimized CPA (3) in the time domain after 15,000 DPA contest v2 traces

Byte	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
CPA	1.12	1.84	1.47	1.67	1.02	1.77	1.5	1.45	1.68	2.17	1.24	1.83	1.63	1.68	1.53	1.81
opt. CPA	2.9	2.74	2.48	3.83	1.71	3.88	3.12	3.15	3.44	3.79	2.16	3.31	2.49	3.24	2.54	4.44

bytes, the optimised coefficients lead to a higher distinguishability and allow for extracting the key with less traces.

6 Summary

We presented a closed form to perform CPA on traces under a linear transform. In contrast to traditional approaches, our method does not require to re-compute the CPA for each particular choice of the transform parameters. Thus, in cases where many different parameters are to be tested, our method allows for a substantially faster evaluation. Consequently, we derived an optimisation criterion allowing to find an "optimal" transform in the sense that it maximises the distinguishability of the correct key candidate. Using both simulated and real-world traces, we demonstrated that this technique performs better than traditional methods and offers a systematic way to derive linear filters for SCA. Especially when designing countermeasures against SCA, our method allows to give a more comprehensive (and objective) assessment regarding the effectiveness of protection mechanisms.

Future Work. Our work offers several starting points for further research: first of all, the employed numerical optimisation algorithm was used "out-of-the-box". We believe that an algorithm adapted to the specific requirements of our method may lead to better results and avoid the problem of overfitting. Besides, the proposed optimisation criterion could be replaced, utilising a different metric for the distinguishability of the correct key candidates. It would also be interesting to investigate whether an analytical solution for the present or a different suitable optimisation criterion can be computed efficiently. In this regard, the applicability of statistical methods like CCA in a side-channel context would deserve some attention.

We limited our experiments to the weight coefficients applied to the traces. However, equivalently, the prediction could also be subject to a linear transform. This essentially corresponds to finding a suitable model for the contribution of single bits to the overall leakage, i.e., relates to SCA with stochastic models [20]. Finally, we focused on CPA only. However, distinguishers like Mutual Information Analysis (MIA) [9] have been shown to be superior in certain cases. Thus, finding a similar technique to compute the mutual information of transformed traces without re-executing the complete MIA is worth further research.

Acknowledgements. The work described in this paper has been supported in part by the European Commission through the ICT program under contract ICT-2007-216676 ECRYPT II and by the German Federal Ministry of Education and Research BMBF (grant 01IS10026A, Project EXSET).

References

1. Agrawal, D., Archambeault, B., Rao, J.R., Rohatgi, P.: The EM Side-Channel(s). In: Kaliski Jr., B.S., Koç, Ç.K., Paar, C. (eds.) CHES 2002. LNCS, vol. 2523, pp. 29–45. Springer, Heidelberg (2003)
2. Barenghi, A., Pelosi, G., Teglia, Y.: Improving First Order Differential Power Attacks through Digital Signal Processing. In: Proceedings of the 3rd International Conference on Security of Information and Networks, SIN 2010, pp. 124–133. ACM, New York (2010)
3. Barenghi, A., Pelosi, G., Teglia, Y.: Information Leakage Discovery Techniques to Enhance Secure Chip Design. In: Ardagna, C.A., Zhou, J. (eds.) WISTP 2011. LNCS, vol. 6633, pp. 128–143. Springer, Heidelberg (2011)
4. Batina, L., Hogenboom, J., van Woudenberg, J.G.J.: Getting More from PCA: First Results of Using Principal Component Analysis for Extensive Power Analysis. In: Dunkelman, O. (ed.) CT-RSA 2012. LNCS, vol. 7178, pp. 383–397. Springer, Heidelberg (2012)
5. Brier, E., Clavier, C., Olivier, F.: Correlation Power Analysis with a Leakage Model. In: Joye, M., Quisquater, J.-J. (eds.) CHES 2004. LNCS, vol. 3156, pp. 16–29. Springer, Heidelberg (2004)
6. Clavier, C., Coron, J.-S., Dabbous, N.: Differential Power Analysis in the Presence of Hardware Countermeasures. In: Koç, Ç.K., Paar, C. (eds.) CHES 2000. LNCS, vol. 1965, pp. 252–263. Springer, Heidelberg (2000)
7. COMELEC department, Télécom ParisTech. DPA Contest v2. Website, http://www.dpacontest.org/v2/index.php
8. Gebotys, C.H., Ho, S., Tiu, C.C.: EM Analysis of Rijndael and ECC on a Wireless Java-Based PDA. In: Rao, J.R., Sunar, B. (eds.) CHES 2005. LNCS, vol. 3659, pp. 250–264. Springer, Heidelberg (2005)
9. Gierlichs, B., Batina, L., Tuyls, P., Preneel, B.: Mutual Information Analysis – A Generic Side-Channel Distinguisher. In: Oswald, E., Rohatgi, P. (eds.) CHES 2008. LNCS, vol. 5154, pp. 426–442. Springer, Heidelberg (2008)
10. Hardoon, D.R., Szedmak, S., Shawe-Taylor, J.: Canonical Correlation Analysis: An Overview with Application to Learning Methods (May 2003)
11. Kasper, T., Oswald, D., Paar, C.: Side-Channel Analysis of Cryptographic RFIDs with Analog Demodulation. In: Juels, A., Paar, C. (eds.) RFIDSec 2011. LNCS, vol. 7055, pp. 61–77. Springer, Heidelberg (2012)
12. Kocher, P.C., Jaffe, J., Jun, B.: Differential Power Analysis. In: Wiener, M. (ed.) CRYPTO 1999. LNCS, vol. 1666, pp. 388–397. Springer, Heidelberg (1999)
13. Mangard, S., Oswald, E., Popp, T.: Power Analysis Attacks: Revealing the Secrets of Smart Cards. Springer, Secaucus (2007)
14. Messerges, T.S., Dabbish, E.A., Sloan, R.H.: Investigations of Power Analysis Attacks on Smartcards. In: USENIX Workshop on Smartcard Technology, pp. 151–162 (1999)
15. Moradi, A., Barenghi, A., Kasper, T., Paar, C.: On the Vulnerability of FPGA Bitstream Encryption against Power Analysis Attacks: Extracting keys from Xilinx Virtex-II FPGAs. In: Chen, Y., Danezis, G., Shmatikov, V. (eds.) ACM Conference on Computer and Communications Security (CCS 2011), pp. 111–124 (2011)
16. National Institute of Advanced Industrial Science and Technology (AIST). Side-channel Attack Standard Evaluation Board SASEBO-GII Specification, 1.01 edition (2009)
17. Oswald, D., Paar, C.: Breaking Mifare DESFire MF3ICD40: Power Analysis and Templates in the Real World. In: Preneel, B., Takagi, T. (eds.) CHES 2011. LNCS, vol. 6917, pp. 207–222. Springer, Heidelberg (2011)

18. Plos, T., Hutter, M., Feldhofer, M.: Evaluation of Side-Channel Preprocessing Techniques on Cryptographic-Enabled HF and UHF RFID-Tag Prototypes. In: Dominikus, S. (ed.) Workshop on RFID Security — RFIDSEC 2008, pp. 114–127 (2008)
19. Renauld, M., Standaert, F.-X., Veyrat-Charvillon, N., Kamel, D., Flandre, D.: A Formal Study of Power Variability Issues and Side-Channel Attacks for Nanoscale Devices. In: Paterson, K.G. (ed.) EUROCRYPT 2011. LNCS, vol. 6632, pp. 109–128. Springer, Heidelberg (2011)
20. Schindler, W., Lemke, K., Paar, C.: A Stochastic Model for Differential Side Channel Cryptanalysis. In: Rao, J.R., Sunar, B. (eds.) CHES 2005. LNCS, vol. 3659, pp. 30–46. Springer, Heidelberg (2005)
21. Smith, J.O.: General LTI Filter Matrix. In: Introduction to Digital Filters with Audio Applications. Center for Computer Research in Music and Acoustics (2007), `http://www.dsprelated.com/dspbooks/filters/`
22. Standaert, F.-X., Archambeau, C.: Using Subspace-Based Template Attacks to Compare and Combine Power and Electromagnetic Information Leakages. In: Oswald, E., Rohatgi, P. (eds.) CHES 2008. LNCS, vol. 5154, pp. 411–425. Springer, Heidelberg (2008)
23. The MathWorks, Inc. MATLAB R2011b Documentation, Optimization Toolbox, fminunc. Website (Online; accessed February 28, 2012)
24. Weisstein, E.W.: Variance. Mathworld - A Wolfram Web Resource (December 2010), `http://mathworld.wolfram.com/Variance.html`

SCA with Magnitude Squared Coherence

Sebastien Tiran and Philippe Maurine

University of Montpellier / LIRMM
161 Rue Ada
34392 Montpellier France

Abstract. Magnitude Squared Coherence (MSC) is a signal processing tool that indicates how well two time domain signals match one with the other by tracking linear dependencies in their spectral decomposition. Spectral Coherence ANalysis (SCAN) was the first way to use it as a Side-Channel Attack (SCA). This paper introduces two ways of using the Magnitude Squared Coherence in side-channel analyses. The first way is to use it as a distinguisher while the second consists in using it to transform the side-channel traces in a worthwhile manner. Additionally, an algorithm for fast computation of the SCAN is provided.

Keywords: Secure Circuits, Side-Channel Attacks, Frequency Domain, Distinguisher.

1 Introduction

Following [7], many side-channel attacks have been proposed in the literature. Most of them directly work with time domain traces, and aim at analysing each time sample independently to retrieve the secret key. However, the power consumption and the electromagnetic (EM) emanations of a cryptographic algorithm, are such that the leakage is spread over many time samples. Consequently, these analyses cannot exploit the leakage in its whole.

By contrast, much less attention has been paid to side-channel analyses performed in the frequency domain that could bring a solution to this problem. To the best knowledge of the authors, Gebotys, Ho and Tiu [5] were the first to propose a differential attack after the application of a Fast Fourier Transform (FFT) to side-channel traces. This work was then extended towards Correlation Power Analysis like attacks (CPA) in [2]. Various similar approaches were described in [9]. These works underlined the advantages of frequency domain analyses against misaligned traces but they didn't focus on the fact that they can also capture more efficiently a leakage that is spread over time.

The Magnitude Squared Coherence being a tool that works in the frequency domain to estimate the similarity between two signals, it can be used in the context of side-channel analysis to retrieve the secret key [4,13]. In some cases, it can provide better results than the time domain distinguishers. The advantages of MSC are that it can exploit the leakage scattered in time and fully use it by exploiting several harmonics.

S. Mangard (Ed.): CARDIS 2012, LNCS 7771, pp. 234–247, 2013.

This paper aims at showing the efficiency of the MSC as a tool for side-channel analysis. First, it shows that the MSC is a really interesting distinguisher and provides some explanations related to its efficiency. Second, a new way of using the MSC is introduced. It consists in transforming the traces to get a wider source of information before exploitation by statistical means.

The rest of this paper is organized as follow. Section 2 reminds some basics about Magnitude Squared Coherence. Section 3 briefly recalls its first use in the context of side-channel analysis. It also provides additional information such as an efficient coding of the SCAN. Section 4 presents a method to transform the side-channel traces and introduces various solutions to exploit the resulting source of information. In section 5, experimental results of these attacks are shown. Finally, a conclusion is drawn.

2 Magnitude Squared Coherence

The Magnitude Squared Coherence is a signal processing tool that returns real values between 0 and 1 to indicate how well two time domain signals *x(t)* and *y(t)* match one with the other. It provides scores, *MSC(f)*, allowing to estimate their similarity at various frequencies. The result of the Magnitude Squared Coherence at a given frequency, *f*, is obtained by computing :

$$MSC(f) = \frac{|P_{xy}(f)|^2}{P_{xx}(f).P_{yy}(f)}. \tag{1}$$

where P_{xy} is the cross-power spectral density of *x(t)* and *y(t)* and P_{xx}, P_{yy} are the auto-power spectral densities of *x(t)* and *y(t)*, respectively. To calculate the cross-power spectral density, the Welch's average method is typically used [10]. This consists in : dividing the signals in several overlapping segments of the same length, computing the cross-power spectral density between each pair of segments and finally, computing the average.

$$P_{x,y}(f) = \sum_{i=1}^{n} FFT_{x_i}(f).FFT_{y_i}(f) \tag{2}$$

with $\{x_1(t), x_2(t), ..., x_n(t)\}$ and $\{y_1(t), y_2(t), ..., y_n(t)\}$ the segments of *x(t)* and *y(t)*, respectively.

3 Spectral Coherence ANalysis

Typically in side-channel analysis, an adversary has to collect a set of traces, $\{T_1, T_2, ..., T_n\}$, corresponding to the encryption / decryption of n messages, $\{m_1, m_2, ..., m_n\}$. He then sorts these traces according to a given selection function, f_k, that predicts some intermediate values computed during the algorithm execution and that depend on a part of the key.

$$\begin{aligned} f_k : \{0,1\}^q \times \{0,1\}^p &\rightarrow \{0,1\}^w \\ (m_i, k) &\rightarrow f_k(m_i, k) = c_i \end{aligned} \tag{3}$$

If the Hamming Weight (HW) model is chosen, then f_k predicts a word c_i of w bits processed by the cryptographic algorithm according to the value of a clear / ciphered message m_i and according to a key guess $k \in K$. If the Hamming Distance (HD) model is preferred, then f_k rather provides a word c_i indicating which of the w bits have switched during a given clock cycle.

To apply a SCAN as defined in [4,13], one have to proceed as for a DPA. For each guess on the key and for each of the w bits, the adversary must sort the traces in two subsets depending on the values provided by f_k and compute the means of the two resulting subsets. But instead of computing the difference of the means (DoM), one may first compute the Magnitude Squared Coherence between the two mean traces. Finally, one have to compute the mean, of all obtained *MSC(f)* values in order to fix a score for the considered key guess.

For wrong key guesses, as the traces are not well sorted, the two mean traces are expected to be similar while they should be significantly different for the correct key. This is to say that the correct key is identified by searching the guess with the minimum score. For detailed information about efficient implementation of the SCAN, see Appendix A.

As one may conclude from this brief description, the SCAN is obtained from the DPA simply by replacing the DoM by the MSC distinguisher. This is an intuitive and straightforward way of using the MSC within the context of SCA. However this approach could be far from being optimal. Indeed, the MSC can be used differently. For example, one may use it to transform the side-channel traces containing the leakage as explained below.

4 Transformation of the Leakage

The basic idea of leakage transformation is to construct from the set of available traces, a new set of data on which side-channel attacks are more efficient. The MSC offers the possibility of applying this idea.

4.1 Preprocessing Step

Indeed, given a set of traces, one may compute the *MSC(f)* between each pair of curves. By doing so, one gets several *MSC(f)* for each pair of curves. To exploit fully the leakage, which is scattered on many frequencies, one may then computes the mean of all the *MSC(f)* values to get a score, *Coher(T_i,T_j)*, that will represent, in the rest of the paper, the difference of leakage between two traces, namely T_i and T_j.

$$Coher(T_i, T_j) = \frac{1}{nbf} \cdot \sum_{f=fmin}^{fmax} MSC_{T_i,T_j}(f) \tag{4}$$

where T_i and T_j are two time domain signals and *nbf* is the number of harmonics falling in the bandwidth of interest, ie between f_{min} and f_{max}, the cut-off frequencies of the used equipments .

At the end of this leakage transformation step, one obtains for a set of n traces, $n.\frac{n-1}{2}$ *Coher(T_i, T_j)* scalars, that is to say $\frac{n-1}{2}$ times as much informers, the latter being related to the leakage difference of trace pairs. Such a multiplication of data constitutes an interesting advantage while attacking systems in which keys are regularly refreshed. The open questions are then:

- what is the relevant leakage model?
- how to efficiently extract the secret key from the statistical distribution of the *Coher(T_i, T_j)* values?

4.2 Leakage Model

When working in time domain with now classical attacks (DPA, CPA ...), an adversary typically uses the HW and HD models. The basic idea on which the HW model relies is that computations ending by a '1' (Vdd) usually consume more energy than computations ending by a '0' (Gnd). Similarly, the HD model is based on the idea that a state change burns much more energy than a calculus ending by the same result than the preceding one.

Considering that the power consumption or EM emanations are additive quantities, the leakage model we adopted is based on the following idea : the incoherence (coherence) of two traces is an increasing (decreasing) function of the difference of their HW or HD. That is to say: the greater the difference of the Hamming Weights is, the more incoherent (less coherent) the corresponding traces are. This choice, which relies on the shape of the traces rather than on the amplitude of samples, can be improved. However, it has lead to interesting experimental results given in the next sections.

4.3 Specific Selection Function

With such a leakage model, the question is now : how to sort coherence values? In other words, what are the relevant selection functions? Considering (3) and following the same reasoning as for the previous leakage model, we defined the following selection function :

$$\begin{aligned} \Delta f_k : \{0,1\}^w \times \{0,1\}^w &\rightarrow \{0,1\}^w \\ (c_i, c_j) &\rightarrow \Delta f_k(c_i, c_j) = f_k(m_i, k) \oplus f_k(m_j, k) = \Delta c_{i,j} \end{aligned} \tag{5}$$

with $\Delta c_{i,j}(l)$ the l^{th} bit of $\Delta c_{i,j}$ that represents the difference between the l^{th} bits of c_i and c_j. Δf_k has thus been deduced from the difference of the selection function f_k (3), the latter providing the values of c_i and c_j.

4.4 Mean and Variance Analyses

Following (5), let us define $\{C_k|\Delta c_{i,j}(l) = 0\}$ and $\{C_k|\Delta c_{i,j}(l) = 1\}$ as the two subsets of coherence values for which the l^{th} bits of c_i and c_j are respectively equal and different. Because $\{C_k|\Delta c_{i,j}(l) = 0\}$ ($\{C_k|\Delta c_{i,j}(l) = 1\}$) gathers coherence values associated to pair of traces with a given bit having the same and different HD or HW values, the expectation $E(C_k|\Delta c_{i,j}(l) = 0)$ should be higher than

$E(C_k|\Delta c_{i,j}(l) = 1)$. By averaging these results for every bits from 1 to w, an adversary may expect disclosing the secret key using the following distinguisher :

$$\max_{k \in K} \left\{ \sum_{l=1}^{w} (E(C_k|\Delta c_{i,j}(l) = 0) - E(C_k|\Delta c_{i,j}(l) = 1)) \right\} \tag{6}$$

Similarly, the variances $V(C_k|\Delta c_{i,j}(l) = 0)$ and $V(C_k|\Delta c_{i,j}(l) = 1)$ should have greater values for wrong guesses than for the secret key, k_g. Thus, an adversary may also identify k_g with:

$$\min_{k \in K} \left\{ \sum_{l=1}^{w} V(C_k|\Delta c_{i,j}(l) = 0) \right\} \tag{7}$$

Let us denote by *mean+MSC* and *var+MSC* these two attacks afterwards.

4.5 Correlation Analysis

According to the adopted leakage model, the coherence of two traces is a decreasing function of the difference of $\Delta c_{i,j}(l)$. Assuming additionally that this function is linear, is equivalent to assume that the expectations of $(C_k| \sum_{l=1}^{w} \Delta c_{i,j}(l) = q)$ are decreasing with the increase value of q. This is to say that the more the words c_i and c_j associated to the two traces are different, the less these traces are similar. Thus, an adversary may analyse the correlation between C_k and $\sum_{l=1}^{w} \Delta c_{i,j}$ and identify the secret key by searching the guess with the maximum absolute score.

Let us denote by *corr+MSC* this attack afterwards.

4.6 Non-parametric Tests

In 4.4, we explained why the probability density functions associated to the values $\{C_k|\Delta c_{i,j}(l) = 0\}$ or $\{C_k|\Delta c_{i,j}(l) = 1\}$ must have different values of expectation and variance for a correct guess of the secret key. Let us generalize this reasoning and more precisely let us assume:

- that for k_g, the secret key, the Cumulative Density Function (CDF) constructed with all values $\left\{C_{k_g}|\Delta c_{i,j}(l) = 0\right\}$ is unique and different from all the others,
- that for wrong guesses, k, the CDFs constructed with all values $\{C_k|\Delta c_{i,j}(l) = 0\}$ are similar in shape.

With these assumptions, the secret key can then be identified, using the Kolmogorov - Smirnov test [14,15] of goodness of fit, with:

$$\max_{k_1 \in K} \left\{ \sum_{l=1}^{w} \sum_{k_2 \neq k_1} \delta \left(CDF[C_{k_1}|\Delta c_{i,j}(l) = 0], CDF[C_{k_2}|\Delta c_{i,j}(l) = 0] \right) \right\} \tag{8}$$

with:

$$\delta(F_1, F_2) = \frac{g_1 . g_2}{g_1 + g_2} . Sup_x \left| F_1(x) - F_2(x) \right| . \tag{9}$$

the maximal distance between F_1 and F_2, two Cumulative Distribution Functions.

Let us denote by *KS+MSC* this attack afterwards.

5 Experimental Results

In order to verify the efficiency of the SCAN, and of the leakage transformation with its derived attacks, we have applied them on a set of 5000 traces collected at the surface of an unprotected implementation of the DES. This FPGA implementation operates at 50 MHz. We also compared the results obtained with MSC based attacks to those of well known distinguishers. (It is to notice that the following attacks are noted with a P for power, to keep their usual name, however they are applied on electromagnetic curves.) Among them, we selected:

- The Bravais-Pearson correlation used in the time domain [3] and which is perfectly adapted to a linear leakage,
- the Difference of Means and more precisely the multi-bit Differential Power Analysis (DPA) that also works in the time domain [1] and sum the Difference of Means for each bit,
- the multi-bit DPA (DPAabs) which sums the absolute value of the Difference of Means of each bit [11],
- the Correlation Power Frequency Analysis (CPFA) described in [2] and further analysed [8], to provide a comparison with previous attempts to exploit the frequency domain,
- and two different implementations of the Mutual Information Analysis [6] based on kernel estimations. The first one (MIA) calculates the mutual information between the traces and the sum of all the output bits $\sum_{l=1}^{w} c_i(l)$ (see eq.3). The second one (MIA mb) calculates the mutual information between the traces and the values of each output bits $c_i(l)$ and then computes the average of all results.

Our evaluations have followed the framework proposed in [12]: we computed a global Success Rate taken on the eight sub-keys of the last round of the DES. It is to note that all EM traces were acquired with a Lecroy oscilloscope featuring a 20 GS/s sampling rate and using a low noise 63db amplifier with a 1 GHz bandwidth.

5.1 Efficiency: Number of Traces Required

Table 1 gives for each attack based on a HD model, the number of curves required to reach a given value of Success Rate. In this table 'mb' and 'word' allow identifying the attacks that work at the word level and the ones that work on each bit separately before combining the results obtained for all bits. As can be seen, all attacks are able to disclose the secret key with this limited set of traces. However, in this case, frequency domain analyses have given better results than time domain analyses, especially those applied after the transformation of the leakage.

It is to notice that the two MIA require less curves than CPA to reach a Success Rate of 80% and 100%. This may suggest that the leakage has a linear behavior but not only. It is also important to notice that all analyses based on the MSC, including the SCAN, are the only ones to reach a Success Rate of 80% after the processing of less than 1000 curves, while all time domain analyses have reach a Success Rate of ∼ 10%, only (except DPA abs). Additionally, one can note that all proposed analyses with the leakage transformation have allowed disclosing the key with fewer traces than the SCAN (50% less in the best case).

We can therefore conclude from Table 1 that analyses in the frequency domain may provide better results than time domain analyses and that transforming the traces does not suppress information but seems to increase significantly the number of informers.

Table 1. Number of processed traces vs Success Rate (HD model)

		Success Rate	10%	20%	40%	60%	80%	100%
time domain	word	CPA	775	1075	1525	2150	4475	5000
		MIA	1650	1850	2450	2900	3300	4150
	mb	DPA	850	1175	1750	2800	4250	4975
		DPAabs	500	550	720	825	1075	1400
		MIA mb	950	1150	1250	1600	1750	2100
frequency domain	word	CPFA	1110	1205	1410	1630	2025	3150
		corr+MSC	320	410	480	532	660	730
	mb	SCAN	375	390	420	480	615	1200
		mean+MSC	230	260	310	440	495	650
		var+MSC	440	450	535	670	780	1135
		KS+MSC	350	370	440	455	540	690

Table 2 gives the same results than Table 1 but this time in case of an adversary adopting the HW model. Only the MSC based analyses the DPA abs and one of the two MIA (MIA mb) are able to retrieve entirely the key with this set of 5000 traces. From Tables 1 and 2 we may thus conclude that MSC based analyses provide the best results. However that doesn't explain why these analyses outperform the time domain attacks. One first explanation could be that they work in the frequency domain, but this is not sufficient! Indeed, one of the MIA remains efficient and the DPA abs also.

5.2 Efficiency: CPU Times

It is necessary to notice that all our attacks were coded in C and were launched on a standard computer with a CPU running at 3 GHz. The CPU time costs of

Table 2. Number of processed traces vs Success Rate (HW model)

		Success Rate	10%	20%	40%	60%	80%	100%
time domain	word	CPA	fail	fail	fail	fail	fail	fail
		MIA	fail	fail	fail	fail	fail	fail
	mb	DPA	fail	fail	fail	fail	fail	fail
		DPAabs	2800	3175	3925	4400	4800	4950
		MIA mb	3550	3700	3900	4350	4550	4900
frequency domain	word	CPFA	fail	fail	fail	fail	fail	fail
		corr+MSC	2375	2515	2705	3495	3990	4810
	mb	SCAN	1750	1900	2300	2950	3625	4200
		mean+MSC	2430	2510	2685	3460	3980	4855
		var+MSC	4250	4400	fail	fail	fail	fail
		KS+MSC	2120	2580	3310	3710	4070	4495

Table 3. CPU times of the attacks with a step of 10 curves

Number of traces :	500	1000
CPA	13s	26s
DPA and DPAabs	15s	30s
MIA	4m	8m
MIA mb	13m	25m
CPFA	13s	27s
SCAN	15s	31s
MSC based analyses	1h5m	4h20m

the different attacks were measured. Table 3 gives the results obtained for two sets of 500 and 1000 traces, respectively. The step refers to the number of traces between which the computation of the distinguisher is done to retrieve the key.

As can be seen, the application of a well implemented SCAN requires roughly the same CPU time than a CPA. However all MSC based analyses that work on pairs of curves are time consuming. Indeed, the increase of the number of informers comes at the cost of an increase of time computation which seems quadratic (for a set of n traces, the number of coherences to compute is proportional to $n.\frac{n-1}{2}$). Consequently, such attacks are to be used on a limited set of traces; i.e. on systems embedding a frequent refreshing of the keys.

5.3 Advantages of the Frequency Domain

One main advantage of working in the frequency domain is the ability to catch the leakage spread over time, while time domain attacks that aim at analysing each time sample independently don't fully use it. Additionally, analysing several samples at a time, as frequency domain analyses do, may provide a certain level of robustness against the eventual existence of a time sample with an outlier

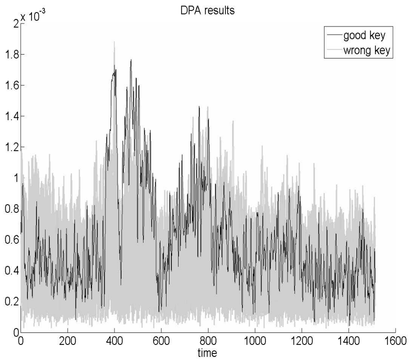


Fig. 1. Results of a DPA targeting the four output bits of the sbox $n°4$ after the processing of 1000 curves

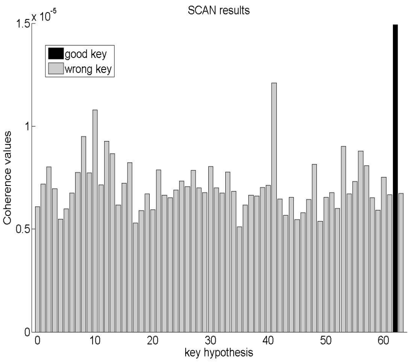


Fig. 2. Results of a SCAN targeting the four output bits of the sbox $n°4$ after the processing of 1000 curves

behavior with respect to the leakage model. We verified this potential advantage. Figure 1 shows the results, obtained for the fourth sbox, of the multi-bit DPA based on a HD model after the processing of 1000 curves. As can be seen, there is a peak corresponding to a wrong guess of the key that prevents from finding the good key with 1000 curves while the SCAN, and the MSC based analyses, performed with 512 consecutive samples, are not disturbed by its occurrence as shown by Figure 2. This figure gives the coherence value obtained by each key hypothesis after the processing of the same 1000 traces with the SCAN. Thus, the 'filtering' of few peaks with an outlier behavior constitutes a first advantage of MSC based analyses.

The reading of Tables 1 and 2 highlights that the MIA and the analyses based on the Magnitude Squared Coherence give better results, on this set of traces, than the CPA when the HW and the HD models are adopted. Thus one may wonder about the correctness of the assumption according to which the leakage (an EM leakage in our case) depends linearly either on the Hamming Distance or the Hamming Weight, even if we were expecting to observe a strong linear dependency of the leakage with the HD because of the iterative implementation of the DES.

We thus tried to find the degree of the polynomial representing at best the evolution of the leakage according to the Hamming Distance and to the Hamming Weight. Figures 3.a and 3.b are scatter plots vs the HD and the HW of a single sample of the traces sorted according to the output value of the fourth sbox: $\sum_{l=1}^{w} c_i$. Figures 3.c and 3.d show the polynomial representing at best the leakage. These polynomials were obtained with the least squared method. It should be noticed that similar figures were obtained for neighbouring samples.

The best modelling of leakage found when the traces are sorted according to the HD model, considered at the word level, is obviously linear as it was expected. This explains why analyses based on the HD models at word level, such as the CPA and the MIA, work well.

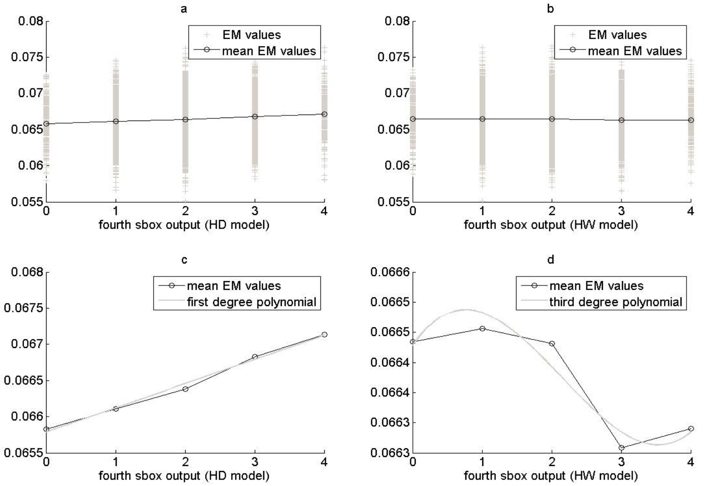


Fig. 3. Figures a and b show the values of a single sample of EM traces sorted according to guesses on the output of the fourth sbox. Figures c an d show the polynomials with the lowest degree representing at best the leakage.

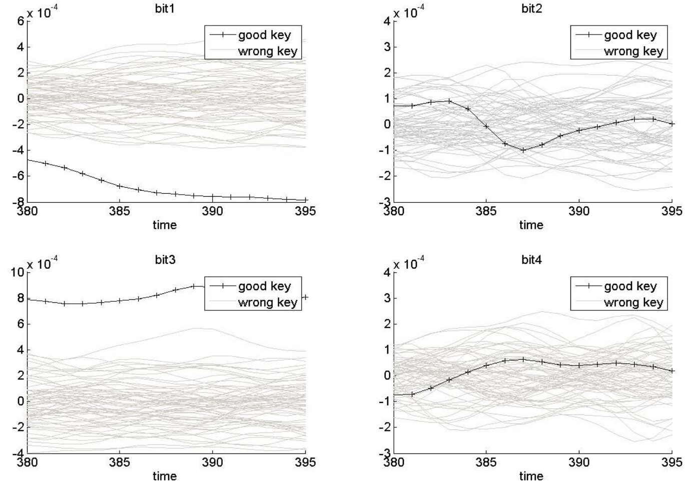


Fig. 4. Results of mono bit DPA targeting each of the four output bits of the fourth sbox

However, when traces are sorted according to a HW model, the leakage is clearly not linear, and a polynomial of degree three is at least needed to describe it correctly. It is to notice that the ordinates of Figures 3.c and 3.d are not the same, and that compared to 3.c, 3.d seems quite constant, suggesting that there is no leakage when working with $\sum_{l=1}^{w} c_i$. These observations explain why analyses based on the HW model at word level, such as the CPA and the MIA, do not succeed in disclosing the key.

On the contrary, when we observed each output bit separately (represented figure 4), results showed that the four output bits of the fourth sbox have different behaviors. Two of them don't seem to leak data dependent information, while the two others leak opposite information. These two last bits, when working at the sum level, cancel each other and this explains why the attacks, such as the multi-bits DPA, can't retrieve the key. However, attacks that work at the bit level, and give a positive value to the score of each bit, such as the DPA absolute Sum, the SCAN the MIA mb, are able to find the key.

From all the above analyses, we therefore conclude that attacks based on the MSC offer the advantages:

- to work at bit level and thus to offer a significant resistance to the eventual non linearity of the leakage model at word level,
- to score the leakage with a positive real value ranging between 0 and 1 so that the cancellation of the various bit contributions is avoided (DPA Absolute Sum and MIA mb do exactly the same),
- to be able to filter some peaks with outlier behaviors by working on several consecutive samples,

6 Conclusion

From all the above analyses and results, one may conclude that Magnitude Squared Coherence is an efficient tool for SCA. Indeed, it can be directly used as a distinguisher characterized by an interesting robustness to the occurrence of outlier behaviors on few samples of the leakage traces. Additionally, working with this distinguisher at bit level, confers a significant robustness against an eventual non linearity of the leakage at word level. The Magnitude Squared Coherence can also be used to transform a reduced set of traces into a wider set of scalar data without loss of information and even with a significant increase of the amount of information. The resulting set of data can then be advantageously used to obtain the secret key by statistical means. One may now wonder how to mount an higher order analysis with such a tool.

References

1. Bévan, R., Knudsen, E.: Ways to enhance differential power analysis. In: Lee, P.J., Lim, C.H. (eds.) ICISC 2002. LNCS, vol. 2587, pp. 327–342. Springer, Heidelberg (2003)
2. Bohl, E., Hayek, J., Schimmel, O., Duplys, P., Rosenstiel, W.: Correlation power analysis in frequency domain. In: COSADE, Darmstadt, Germany (2010)

3. Brier, E., Clavier, C., Olivier, F.: Correlation power analysis with a leakage model. In: Joye, M., Quisquater, J.-J. (eds.) CHES 2004. LNCS, vol. 3156, pp. 16–29. Springer, Heidelberg (2004)
4. Dehbaoui, A., Tiran, S., Maurine, P., Standaert, F.-X., Veyrat-Charvillon, N.: Spectral coherence analysis - first experimental results -. Cryptology ePrint Archive, Report 2011/056 (2011), `http://eprint.iacr.org/`
5. Gebotys, C.H., Ho, S., Tiu, C.C.: EM Analysis of rijndael and ECC on a wireless java-based PDA. In: Rao, J.R., Sunar, B. (eds.) CHES 2005. LNCS, vol. 3659, pp. 250–264. Springer, Heidelberg (2005)
6. Gierlichs, B., Batina, L., Tuyls, P., Preneel, B.: Mutual information analysis – A generic side-channel distinguisher. In: Oswald, E., Rohatgi, P. (eds.) CHES 2008. LNCS, vol. 5154, pp. 426–442. Springer, Heidelberg (2008)
7. Kocher, P.C., Jaffe, J., Jun, B.: Differential power analysis. In: Wiener, M. (ed.) CRYPTO 1999. LNCS, vol. 1666, pp. 388–397. Springer, Heidelberg (1999)
8. Mateos, E., Gebotys, C.H.: A new correlation frequency analysis of the side channel. In: WESS, p. 4 (2010)
9. Meynard, O., Réal, D., Guilley, S., Flament, F., Danger, J.-L., Valette, F.: Characterization of the electromagnetic side channel in frequency domain. In: Lai, X., Yung, M., Lin, D. (eds.) Inscrypt 2010. LNCS, vol. 6584, pp. 471–486. Springer, Heidelberg (2011)
10. Welch, P.D.: The use of fast fourier transform for the estimation of power spectra: A method based on time averaging over short. IEEE Trans. Audio Electroacoustics 15, 70–73 (1967)
11. Standaert, F.-X., Gierlichs, B., Verbauwhede, I.: Partition *vs.* Comparison side-channel distinguishers: An empirical evaluation of statistical tests for univariate side-channel attacks against two unprotected CMOS devices. In: Lee, P.J., Cheon, J.H. (eds.) ICISC 2008. LNCS, vol. 5461, pp. 253–267. Springer, Heidelberg (2009)
12. Standaert, F.-X., Malkin, T.G., Yung, M.: A unified framework for the analysis of side-channel key recovery attacks. In: Joux, A. (ed.) EUROCRYPT 2009. LNCS, vol. 5479, pp. 443–461. Springer, Heidelberg (2009)
13. Tiran, S., Dehbaoui, A., Maurine, P.: Magnitude squared coherence based SCA. Cryptology ePrint Archive, Report 2012/077 (2012), `http://eprint.iacr.org/`
14. Veyrat-Charvillon, N., Standaert, F.-X.: Mutual information analysis: How, when and why? In: Clavier, C., Gaj, K. (eds.) CHES 2009. LNCS, vol. 5747, pp. 429–443. Springer, Heidelberg (2009)
15. Whitnall, C., Oswald, E., Mather, L.: An exploration of the kolmogorov-smirnov test as a competitor to mutual information analysis. In: Prouff, E. (ed.) CARDIS 2011. LNCS, vol. 7079, pp. 234–251. Springer, Heidelberg (2011)

A SCAN Implementation

A.1 Naive Implementation of SCAN

During the application of a side-channel attack on a set of curves, one often wants to see the results progressively and not only after the processing of all the available traces. This usually allows stopping an attack as soon as this one may be considered as successful. For that, an adversary have to compute the attack by steps, i.e. each time a given number of additional traces has been processed.

Algorithm 1 gives a first way of implementing the SCAN, based on a multi-bits DPA approach. It consists in computing a mean trace for each key hypothesis and for each bit of the predicted output. At each step, instead of doing the difference of means, the MSC is computed between the averaged traces associated to the two possible values of each predicted output bit.

However the computation of the Magnitude Squared Coherence implies the calculation of a significant number of Fast Fourier Transforms (see equations (1-2)). This number depends on the segmentation of the signals when the Welch method is adopted. For an algorithm such as the AES, retrieving the entire key implies the computation of the MSC for each 16 sbox, for each 256 possible key hypotheses and for each 8 output bits. Thus one have to compute 32768 MSC at each step.

A.2 Fast Implementation of SCAN

To reduce this number, in case of a low step value, one may compute directly the FFT of each trace. Due to the linearity of the Fourier Transform, he can then compute the mean of the Fourier Transform of the traces instead of calculating the mean trace. This result in a significant reduction of the CPU time needed to process a set of traces with a low step value. Algorithm 2 represents the pseudocode of this new approach.

Algorithm 1. SCAN pseudocode

```
Input : messages $T_i$
Output : guessed key $k_g*$
for $i = 1$ to $nbcurves$ do // loop on the number of curves
  for $k = 0$ to $nbk$ do // loop on the number of key hypothesis
    for $l = 1$ to $w$ do // loop on the number of predicted output bits
      if $c_i(l) = 1$ then // value of a predicted output bit
        $T_k^{c_i(l)=1} += T_i$;
        $cpt_k^{c_i(l)=1}++$;
      else
        $T_k^{c_i(l)=0} += T_i$;
        $cpt_k^{c_i(l)=0}++$;
      end if
    end for
  end for
  if i mod step == 0 then
    for $k = 0$ to $nbk$ do
      for $l = 1$ to $w$ do
        $M_k^{c_i(l)=1} = T_k^{c_i(l)=1} / cpt_k^{c_i(l)=1}$;
        $M_k^{c_i(l)=0} = T_k^{c_i(l)=0} / cpt_k^{c_i(l)=0}$;
      end for
    end for
    $k_g* = \min_{k \in K} \left\{ \frac{1}{w} \frac{1}{nbf} \sum_{l=1}^{w} \sum_{f} Coher((M_k^{c_i(l)=1}), (M_k^{c_i(l)=0}))(f) \right\}$
  end if
end for
```

It is to notice that the spectral components of the Fourier Transform of each traces are averaged. Thus the Magnitude Squared Coherence is computed on this mean instead of being computed at each frequency before being averaged. However experimental results have shown that these two methods lead approximately to the same results and enable to retrieve the key with nearly the same number of curves. Averaging the spectral components at the beginning greatly speeds up the algorithm as this reduces the number of MSC that have to be computed (one coherence on a mean frequency instead of a coherence at each frequency).

Algorithm 2. Fast SCAN pseudocode

```
Input : messages $T_i$
Output : guessed key $k_g*$
for $i = 1$ to $nbcurves$ do
   for $wind = 1$ to $nbwind$ do // loop on the number of sub-segments of the trace
      $F(wind) = \frac{1}{nbf} \sum_{f} FFT(T_{i,wind})(f);$
   end for
   for $k = 0$ to $nbk$ do
      for $l = 1$ to $w$ do
         if $c_i(l) = 1$ then
            for $wind = 1$ to $nbwind$ do
               $F_k^{c_i(l)=1}(wind) += F(wind);$
            end for
            $cpt_k^{c_i(l)=1} ++;$
         else
            for $wind = 1$ to $nbwind$ do
               $F_k^{c_i(l)=0}(wind) += F(wind);$
            end for
            $cpt_k^{c_i(l)=0} ++;$
         end if
      end for
   end for
   if i mod step == 0 then
      for $k = 0$ to $nbk$ do
         for $l = 1$ to $w$ do
            $P_{1,0}^{k,c_i(l)} = \sum_{wind} F_k^{c_i(l)=1}(wind).F_k^{c_i(l)=0}(wind);$
            $P_{1,1}^{k,c_i(l)} = \sum_{wind} F_k^{c_i(l)=1}(wind).F_k^{c_i(l)=1}(wind);$
            $P_{0,0}^{k,c_i(l)} = \sum_{wind} F_k^{c_i(l)=0}(wind).F_k^{c_i(l)=0}(wind);$
         end for
      end for
      $k_g^* = \min_{k \in K} \left\{ \frac{1}{w} \sum_{l=1}^{w} \left\{ \left| P_{1,0}^{k,c_i(l)} \right|^2 / (P_{1,1}^{k,c_i(l)}.P_{0,0}^{k,c_i(l)} \right\} \right\}$
   end if
end for
```

Strengths and Limitations of High-Resolution Electromagnetic Field Measurements for Side-Channel Analysis

Johann Heyszl[1], Dominik Merli[1],
Benedikt Heinz[1], Fabrizio De Santis[2], and Georg Sigl[2]

[1] Fraunhofer Research Institution AISEC, Munich, Germany
{johann.heyszl,dominik.merli,benedikt.heinz}@aisec.fraunhofer.de
[2] Technische Universität München, Munich, Germany
{desantis,sigl}@tum.de

Abstract. The electromagnetic field as a side-channel of cryptographic devices has been linked to several advantages in past contributions. We provide a comprehensive study using high-resolution horizontal and vertical magnetic field probes at close distance to an integrated circuit die. We configured an FPGA device with two uncorrelated digital structures showing similar leakage behavior as symmetric cryptography implementations. We found that measurements from the frontside of the die using a horizontal probe lead to the highest signal-to-noise ratios. Further, high sampling rates are required and no trace compression should be applied. Contrary to previous contributions, we successfully demonstrate that the leakage of design parts is locally restricted and matches their placement. This proves the feasibility of localized side-channel analysis after a profiling phase, however, also means that other locations will lead to inferior results, which is an important limitation. Our analysis confirmed an advantage of measuring localized electromagnetic fields instead of current consumption due to the fact that less parasitic capacitances are involved.

Keywords: EM, high-resolution, side-channel, localization, CPA, SNR.

1 Introduction

The past years have seen many publications describing the use of the ElectroMagnetic (EM) side-channel, mostly the magnetic near-field, and cartography thereof [10] to find locations where side-channel analyses lead to the best results [5,12,11]. The magnetic field is vectored and measured using coil sensors. Different coil angles capture different parts of the fields. Agrawal et al. [1] as well as by Standaert and Archambeau [13] provide evidence for this in the context of side-channel analysis. Gandolfi et al. [4] state that inductive probes with high spatial resolutions can be used to locally restrict measurements to specific circuit parts if they are placed close to the surface of an integrated circuit. A variety of magnetic probes have been used in past contributions. Large, hand-crafted ones are used

S. Mangard (Ed.): CARDIS 2012, LNCS 7771, pp. 248–262, 2013.

by Mulder et al. [3] for global measurements of a chip. Peeters et al. [9] use a custom designed probe with a coil diameter of 0.7 mm at a fixed position and close distance to an integrated circuit after partly removal of the package. Sauvage et al. [12] use laboratory equipment with a coil diameter of 0.5 mm outside the chip's package. As a conclusion, they state that observed areas of signal leakage do not coincide with the placement of the leaking design parts on the floorplan of the FPGA. We suggest that the measurement equipment and distance to the die surface have been insufficient leading to mainly observing the magnetic field of bonding wires. Kirschbaum and Schmidt [7] present evidence for successfully localizing EM leakage and performed cartographic measurements. However, they use a hand-crafted coil with 0.5 mm diameter, which has a comparably coarse resolution in our opinion. Heyszl et al. [6] use a high-resolution probe and show that the information leakage significantly depends on the measurement location. They provide first results, however, clear evidence for the feasibility of localizing leakage of circuit parts is lacking.

We fill this gap by performing a comprehensive study of the electromagnetic near-field side-channel using high-resolution measurement equipment at close distance to a decapsulated integrated circuit die. We employ magnetic probes with horizontal, and vertical coils and discuss important parameters of the measurement setup. We analyze a design-under-test configured into an FPGA consisting of a register with a loop feedback through the AES substitution function. Therefore, our results allow conclusions about the side-channel analysis of symmetric cryptography implementations. We conclude, that measurements from the frontside of an integrated circuit using a horizontal probe lead to the highest signal-to-noise ratios. Further we argue, that high sampling rates are required and no trace compression should be applied. Hence, as a main contribution, we clearly demonstrate the feasibility of matching localized electromagnetic fields with placed design parts. This proves the feasibility of restricting side-channel analysis to parts of a design after finding the correct positions through profiling. However, this also demonstrates that incorrect positioning of high-resolution equipment leads to inferior signal-to-noise-ratios. We compared the achieved signal-to-noise ratios against results that we derived from analyzing conventional current consumption measurements. Leakage signals in the EM field are observed within short times after the active clock edge, making local EM measurements favorable for analyzing devices with high clock frequencies since less parasitic capacitances influence the observation.

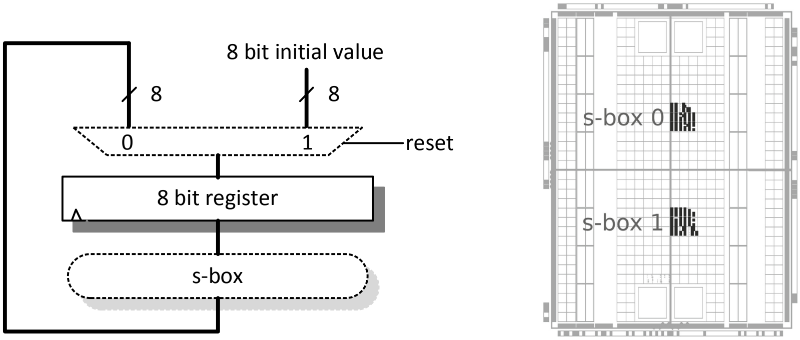
We describe the equipment, design and analysis method in Sect. 2. In Sect. 3, we present and discuss our measurement results and derive conclusions which are summarized in Sect. 4.

2 Practical Study

In this section, we present our device-under-test, measurement equipment and analysis methods.

2.1 Device-Under-Test

We use a *Xilinx Spartan 3A XC3S200A* FPGA in a *VQ100* package, as device-under-test. The device is manufactured in a 90 nm technology, uses a 1.2 V supply for the internal logic, and the die measures 4100 × 4300 µm. To perform semi-invasive measurements close to the surface of the chip, we decapsulated the FPGA from the front-, and backside using fuming nitric acid, i.e., with a concentration of $> 95\%$.



(a) S-box-structure

(b) Placement

Fig. 1. Design-under-test

The FPGA is configured with a hardware design-under-test. We use a specific, simple design to simplify the acquisition of measurements, while being able to draw meaningful conclusions about the side-channel leakage of cryptographic designs. This design is depicted in Fig. 1(a) and contains a feedback loop structure including an 8-bit register and an implementation of the AES substitution function, s-box, as published by Canright [2]. The 8-bit register is loaded with a fixed initial value at synchronous reset and updates the register with the value's s-box-substitution in every cycle. Therefore, every clock cycle contains a value update, i.e. Hamming distance. The design always performs the same operation. Hence, there are no operation-dependencies and the design serves to analyze data-dependent side-channel leakage. This is according to implementations of symmetric cryptographic algorithms which are primarily subject to differential side-channel attacks relying exclusively on data-dependent leakage. According to our opinion, this structure exhibits similar side-channel behavior as implementations of the AES algorithm, because the same non-linear function is used and the amount of combinational logic is representative for such implementations.

To analyze localized aspects of the electromagnetic side-channel leakage, we use two instantiations of this register-s-box structure. We used constraints to place the s-box structures 0 and 1 on the FPGA at a certain distance and to restrict both structures to the same area. The placement on the floorplan of the *Xilinx Spartan 3A* FPGA is depicted in Fig. 1(b). Since both structures are active at the same time, they consume power at the same time and contribute to the electromagnetic field jointly. To analyze the contributions of the two structures separately, they need to be statistically independent. The two instantiations use different initial values for their feedback loop. If values from a limited space are repeatedly replaced by a substitution function projecting into the same space, the initial values are eventually derived since the number space is limited. The number of substitutions, thus, length of the sequence of values depends on the number space, substitution function and the generating initial value. We achieve an independence, or de-correlation of both structures by using sequences with different initial values and different lengths for both structures. Hence, the offset between the two sequences is different for every repetition of either one. *The s-box structure with index* 0 *has an initial value of* ***0x1d****, resulting in a sequence length of* 87*. The s-box structure with index* 1 *has an initial value of* ***0x09****, resulting in a sequence length of* 81*.* Obviously, neither of the two sequences contains values from the respective other sequence. The design additionally includes a 16-bit counter to generate an external trigger for the oscilloscope and synchronously reset both structures. Every measurement contains 2^{16} consecutive clock cycles, thus, 753 repetitions of the sequence with length 87 and 809 repetitions of the sequence with length 81.

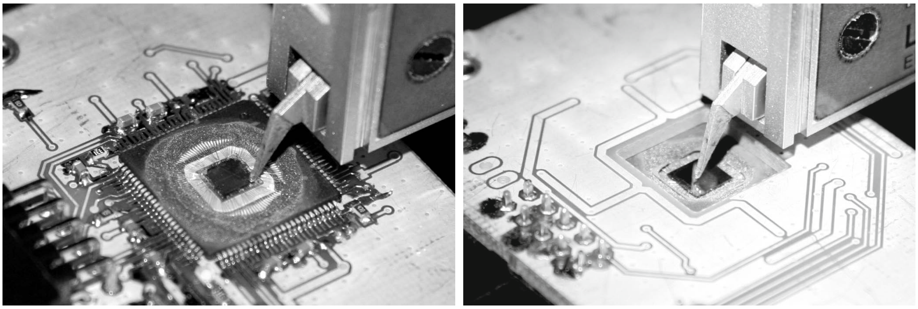
2.2 Measurement Setup

Magnetic fields are vector-fields and magnetic coils only capture components which are orthogonal to the coil. It is not obvious which probe, or which coil direction leads to the best results for side-channel analysis. We used the following magnetic probes to measure the magnetic near-field:

1. Magnetic field probe with 150 µm shielded horizontal coil, 6 windings, 100 µm resolution, and 2.5 MHz – 6 GHz frequency span.
2. Magnetic field probe with 150 µm shielded vertical coil, 6 windings, 80 µm resolution, and 2.5 MHz – 6 GHz frequency span.

The horizontal coil probe will measure the vertical components of the superposed magnetic field generated by the circuit. The vertical probe will record horizontal magnetic field components and provides a choice of direction of the probe. We limited our analysis to x- and y-directions since conductors in integrated circuits are limited in these directions due to manufacturing stability reasons. We also took measurements using a high-resolution electric field probe. *However, the measurements did not reveal any detectable signals, thus, we conclude that this probe is unsuitable for side-channel analysis.* The reason might be that the electric field is shielded by the conductors within the circuit. Ferromagnetic conductors within an integrated circuit also influence magnetic fields, however, our results indicate that the field is still detectable with high SNRs.

We moved the probes over the front- and backside surface of the FPGA die using a stepping table at a resolution of 100 µm and recorded one measurement at every position. The backside measurement is depicted in Fig. 2(b). The probe



(a) Frontside

(b) Backside

Fig. 2. Probe positioned and moved over the surface of the FPGA die

touches the surface of the circuit and we recorded 43 × 41 measurements. The frontside measurement is depicted in Fig. 2(a). To prevent damaging probe and die, the probe to surface distance is $\approx 50\,\mu\text{m}$. The bonding wires prevent measurements over the complete surface and we recorded 27 × 27 measurements in the area enclosed by the bonding wires.

All probes contain a built-in 30 dB amplifier with a noise figure of 4.5 dB. Additionally, we use a 30 dB amplifier with a bandwidth of 3 GHz and noise figure of 4.5 dB. Our *LeCroy WavePro 715Zi* oscilloscope has an analog input bandwidth of 1.5 GHz at 50 Ω impedance. As an approximate upper boundary for the sampling rate, twice the bandwidth of the equipment, thus, 5 GS/s seems reasonable. We used a zero offset for all measurements and a vertical resolution of 50 mV/DIV and confirmed that all measurements stay within scale. The noise contribution from the measurement setup, i.e., the probes, the two amplifiers, and the oscilloscope was determined by turning of the clock and voltage supply and recording a trace containing noise exclusively. This resulted in noise with a standard deviation of $\approx 22.3\,\mu\text{V}$ for the horizontal magnetic probe, and noise with a standard deviation of $\approx 20\,\mu\text{V}$ for the vertical one.

We use a 20 MHz clock signal for the design on the FPGA. Through synchronization of the oscilloscope and the function generator, we prevent frequency jitter and drift in the measurements. Every measurement contains 16384000 byte samples for the 2^{16} recorded clock cycles.

We took a current consumption measurement to compare its quality for side-channel analysis to high-resolution EM measurements. We use a *LeCroy* active differential probe with a bandwidth of 500 MHz and a 10 Ω measurement resistor.

2.3 Analyses

In every measurement, the two different s-box structures contribute a repeating sequence of value updates of different length. To determine these two independent signal components, every measurement trace is split into sub-traces in two ways according to the different sequence lenghts. First it is split into sequences containing 87 clock cycles each, and second, it is split into sequences containing 81 clock cycles each. Both sets of sequences are averaged separately, thus, the noise is removed from both signals at a statistical sample size of 753, and 809 respectively. The two independent signals remain separately. After subtracting one of those two signals from the trace the noise remains. It includes power supply noise, clock supply noise, measurement noise, quantization error and switching, or algorithmic noise from parts of the circuit which did not contribute to the signal. In our case, the counter as well as the respective other s-box structure, which exhibits an uncorrelated sequence with different length, contribute to this switching noise. By comparing all clock cycles, the noise is observed as a Gaussian mixture with zero mean and standard deviation σ_{noise} at every relative sample index within the clock cycle.

We regard the constant influence of clocking the registers, i.e., the clock tree and common logic parts, in every cycle as an operation-dependent part of the signal. The variance between the clock cycles in the two derived signal sequences is due to the processed data, thus, the data-dependent part of the signal. We estimate the data-dependent signal part by averaging the clock cycles from each signal sequence and subtracting this operation-dependent part from the sequence. The data-dependent signal remains. By comparing all cycles in the sequence, we get a data-dependent Gaussian distribution at every relative sample index within the clock cycle and we describe it using the standard deviation σ_{data}. We compute the Signal-to-Noise Ratio (SNR) over the clock cycle in decibel as the quotient between the signal and noise standard deviations, SNR $= 20 * \log(\frac{\sigma_{data}}{\sigma_{noise}})$ dB. The SNR is derived for both identical signal sources and it depends on the location, which one results in a higher SNR.

We performed a Correlation Power Analysis (CPA) to evaluate the quality of the measurements for differential power analysis. We use the Hamming distance leakage model between values from consecutive cycles. The sample size n for the correlation equals the number of recorded clock cycles, 2^{16}, and a correlation coefficient of 0 results in values of $\pm 4/\sqrt{n} = \pm 0.015625$ with a confidence level of 99.99% [8]. Therefore, absolute correlation coefficients below this significance level are disregarded.

3 Discussion of Measurement Results

In this section, we discuss measurement results and derive conclusions. Since the measurements from the frontside with the horizontal coil led to the best results, we provide details for this case and use it as a base for comparison.

3.1 Signal and Noise

Figure 3 depicts the mean and standard deviation of all clock cycles from one measurement at position $(x, y) = (24, 17)$. This position is approximately above s-box structure 1 and will serve as an interesting example. The figure spans the time of one clock cycle, thus, 50 ns. The mean represents the operation-dependent part of the signal which is for instance due to clocking the registers

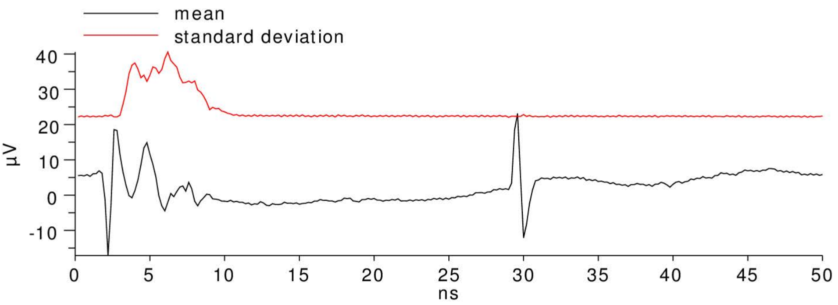


Fig. 3. Mean and standard deviation of all clock cycles at position (24, 17) (frontside, horizontal coil)

and the active and inactive clock edges can be observed as significant peaks. The standard deviation is constant throughout most of the cycle which can be explained by constant noise factors from the measurement setup and corresponds well to the measured noise floor mentioned in Sect. 2.2. The standard deviation is significantly higher during a time after the active clock edge which is due to data-dependent switching activity in the circuit. At this stage, this switching activity cannot be attributed to specific parts of the design and contains contributions from all circuit parts.

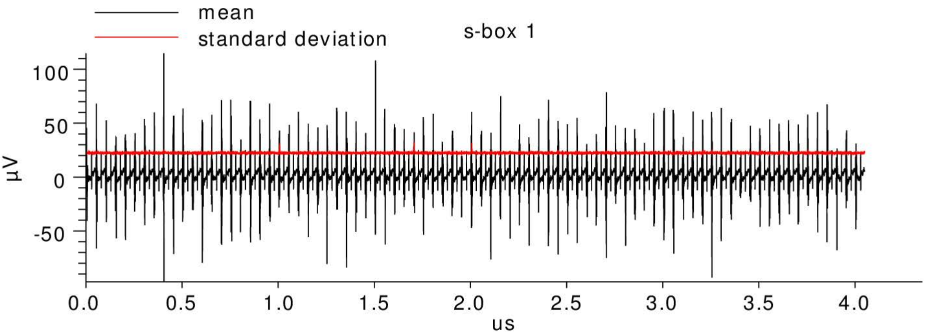


Fig. 4. Mean and standard deviation of repeated s-box 1 sequence at position (24, 17) (frontside, horizontal coil)

We determined the data-dependent signal and noise for both structures in every measurement as described in Sect. 2.3. Figure 4 shows the mean and standard deviation of the repeated sequence of values processed by s-box structure 1 which contains 81 clock cycles. The standard deviation trace is similar to the one depicted in Fig. 3, however, there are no times with significantly high standard deviations like in the previous figure. This is due to the fact that the data-dependent signal parts are now included in the mean trace. This is clearly

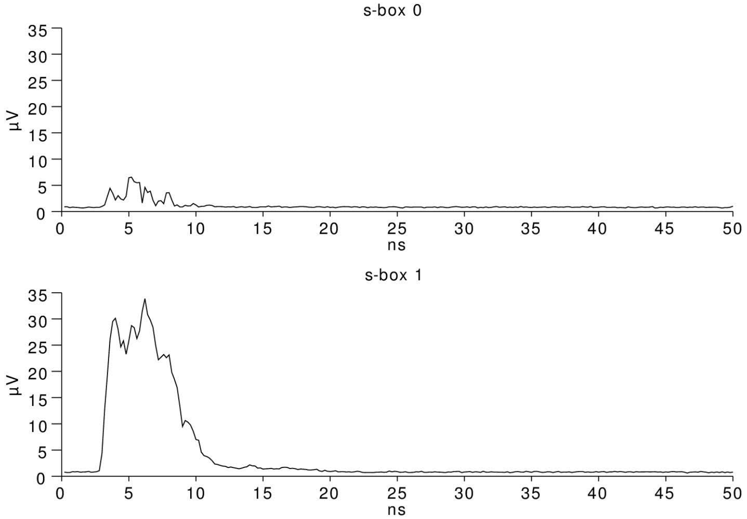


Fig. 5. Data-dependent signal standard deviation σ_{data} over clock cycle for s-box 0 and 1 at position $(24, 17)$ (frontside, horizontal coil)

visible in Fig. 4, where the mean exhibits significantly different amplitudes. To determine the data-dependent part, we repeatedly subtracted the mean clock cycle (Fig. 3) from the mean depicted in Fig. 4. Figure 5 depicts the result, which is the data-dependent signal within the clock cycle of s-box structure 1 in the bottom diagram. The maximum signal amplitude is clearly significant when compared to the noise level mentioned before. The constant floor of $\approx 1\,\mu$V seems to be due to statistical artifacts. We performed the same procedure using the same measurement for s-box structure 0 resulting in the upper diagram in Fig. 5. The signal from s-box structure 0 exhibits a significantly low amplitude which is explained by the fact that the measurement position is close to s-box structure 1 and further away from s-box structure 0. *We conclude that the distance to parts of the design in x- and y-directions is important for the detection of leakage signals.*

Significant signal amplitudes are observed within the first 10 ns after the positive, active clock edge. The synthesis tool reported 12.5 ns delay as the longest combinational path of our design. *It is an important observation, that signal leakage is exclusively restricted to a time-span as short as the combinational path after the active clock edge when analyzing local EM measurements close to the source of the leakage.*

The SNR of the signal leakage of an s-box structure is computed as the maximum of the ratio of data-dependent signal amplitude σ_{data} over noise σ_{noise} within the clock cycle. Strong signal components of an s-box structure add to the algorithmic noise for the respective other s-box structure. The measurements from the frontside using the horizontal coil led to the highest SNRs. Figure 6

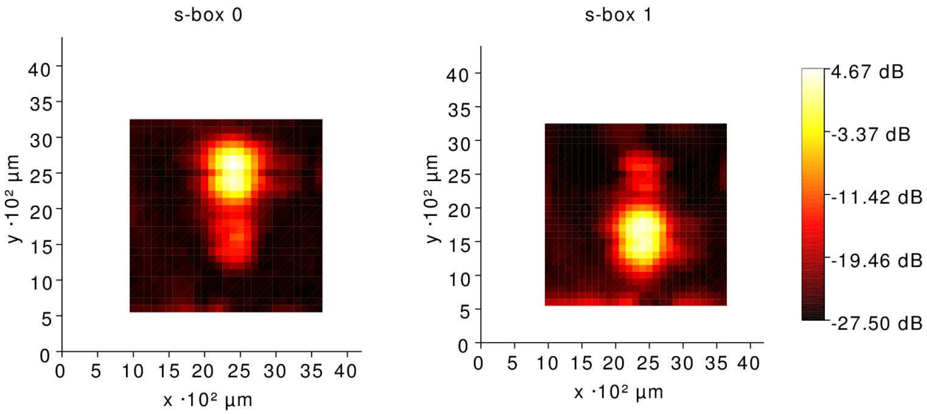


Fig. 6. SNR for both signals (frontside, horizontal coil)

depicts a map of those SNR values for all positions and both s-box structures. We emphasize that a maximum of $\approx 4.7\,\text{dB}$ represents a significant signal strength. *It is remarkable how the signals from the s-box structures are significant in areas above the placed logic of the structures as depicted in Fig. 1(b). This is an important result of our study.* Surprisingly, the SNRs are also higher close to the respective other structure. We suspect that this is due to modulation effect as described by Agrawal et al. [1]. *As another important conclusion, we realize that it is only possible to achieve high SNRs using a high-resolution probe when it is correctly positioned. This requires that adversary is able to find such positions, e.g., through profiling.*

The two s-box structures only occupy a very small area on the FPGA as depicted in Fig. 1(b). We took a measurement where the same s-box structures were distributed over a broader area, thus, requiring longer routing wires. We observed a significantly higher SNR for this case, thus, we conclude that the SNR highly depends on the design and placement.

3.2 CPA and Localization

We performed CPA as described in Sect. 2.3. Figure 7 depicts the correlation coefficient over the clock cycle for both s-box structures at the position $(24, 17)$. We observe high correlation values, positive as well as negative, for s-box structure 1 and insignificant correlation values for s-box structure 0. Correlation peaks are only 1 to 3 samples wide. Therefore, we estimate a minimal required sampling rate of $\approx 2\,GS$ in this case. *This will vary for other devices, but it can be generally expected, that a high sampling rate is required for localized EM measurement.*

Figure 8 depicts a map of maximum absolute correlation coefficients for every measurement on the map. *We strongly emphasize how perfectly distinct the areas with high correlations of the two s-box structures are. This provides the perfect precondition for localized CPA attacks where only a single s-box structure is*

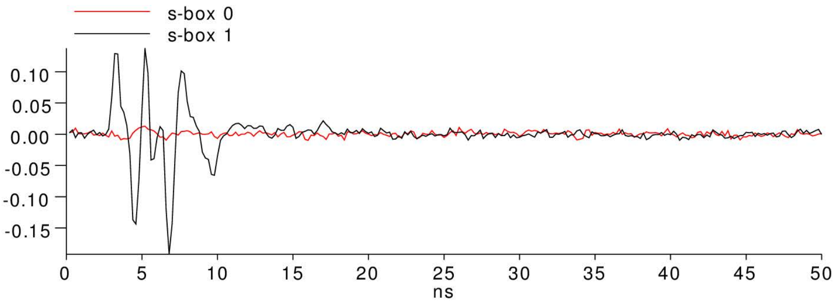


Fig. 7. CPA over cycle at position (24, 17) (frontside, horizontal coil)

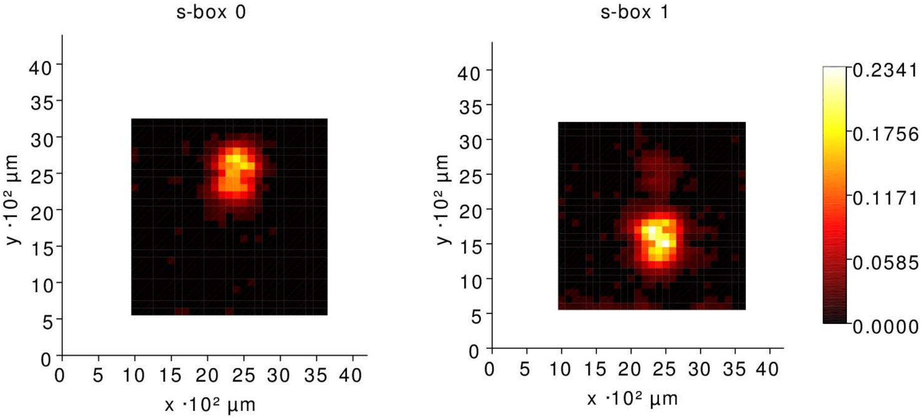


Fig. 8. CPA coefficients (frontside, horizontal coil)

targeted. However, it must be noted, that an adversary must be able to find those positions.

Unfortunately, there is no available information about the physical size of the FPGA cells. During the scanning of the surface, we used a step size of 100 µm. In Fig. 8 and Fig. 6 we observe distinct leakage regions corresponding to the two s-box structures. We assume that the distance between the centers of the leakage regions equals the distance of the structure centers in the placement. From this, we derive an assumed distance of the s-box structure centers of $\approx$ 900 µm and an assumed logic area of the two structures of $\approx 250 \times 250$ µm. In Fig. 8, we observe that the regions with significant correlation coefficients do not overlap. *We assume from the presented evidence that there are non-overlapping regions of significant correlation coefficients even when the two s-box structures are adjacent to each other, thus, when the centers are only* $\approx$ 250 µm *apart.* This is strongly dependent on the logic structure of the device, hence, we do not suggest generalization. *An interleaved placement of s-box structures will render this significantly more difficult, if not impossible, because of entirely overlapping* *regions of correlation coefficients. However, measurement equipment with higher resolution may still support localization.*

3.3 Backside versus Frontside Measurement

Decapsulating a chip from the backside can be achieved with less effort than from the frontside. Hence, it is an important question which preparation leads to better results. Figure 9 depicts the SNR map from using the horizontal coil from the backside. In the middle of the device, the localized signal leakage of both structures is clearly visible in distinct, confined regions. Additionally, regions with high SNRs are observed on the edges of the die. Those regions have not been covered by the frontside measurements and exhibit signals from *both* s-box structures. However, since those regions contain contributions from both structures, they are useless from a localized perspective. We assume that the magnetic field in those regions is caused by bonding wires or parts of the chip supply. The maximum SNR is ≈ 15 dB lower than in case of frontside measurement. This might be due to the silicone substrate and the fact that the conductors within the integrated circuit, which carry the exploitable signals, are on upper metal layers and therefore, further away when measuring from the backside. *We conclude that backside measurements lead to significantly lower SNRs.*

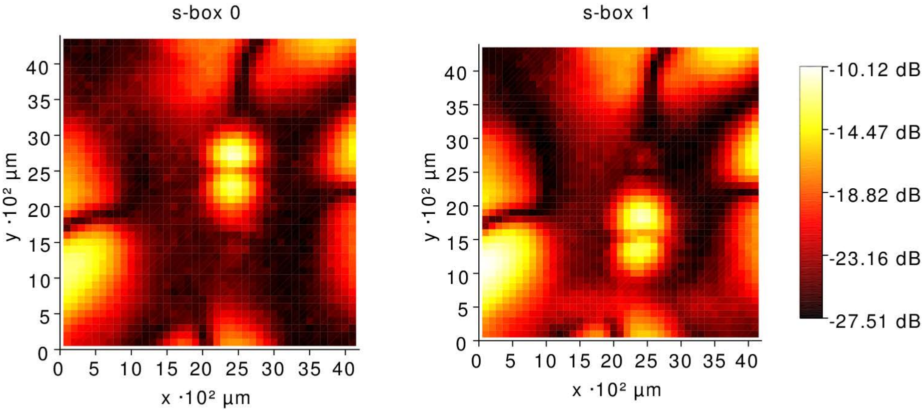


Fig. 9. SNR is ≈ 15 dB lower on backside (horizontal coil)

3.4 Probe-to-Chip Distance

An important question is whether high-resolution EM measurements require semi-invasive decapsulation to achieve minimal probe-to-die distances. We repeated the measurement from the frontside using the horizontal coil and increased the distance of the probe to the surface of the chip by 300 µm which roughly equals the package thickness above the die. The measurements lead to a significantly lower maximum SNR of −16.5 dB. This is ≈ 21 dB lower than the maximum SNR observed in the original measurement depicted in Fig. 6. *We conclude that the semi-invasive decapsulation is important to achieve high SNRs.*

3.5 Vertical Coil

We took measurements using the vertical coil in x-, and y-direction. Figure 10 depicts the SNR maps for the x-direction. A maximum SNR of ≈ -8.9 dB is achieved which is significantly lower than in case of using the horizontal coil. However, the coil seems to capture the magnetic field more selectively than the horizontal coil. We observe, that the regions with high SNR corresponding to the s-box structures have a different, more compact shape than in case of using the horizontal coil which is depicted in Fig. 6. This corresponds to the expectation that different coil orientations 'select' different parts of the field. The results from the y-direction resulted in even lower SNRs and the two regions with high SNRs are dispersed over a wider area, thus, making localization more difficult. *Given those observations, we conclude that the probe with the vertical coil leads to lower SNRs compared to horizontal coils. However, a better selectivity can be achieved for better localization if the coil direction supporting this is known.*

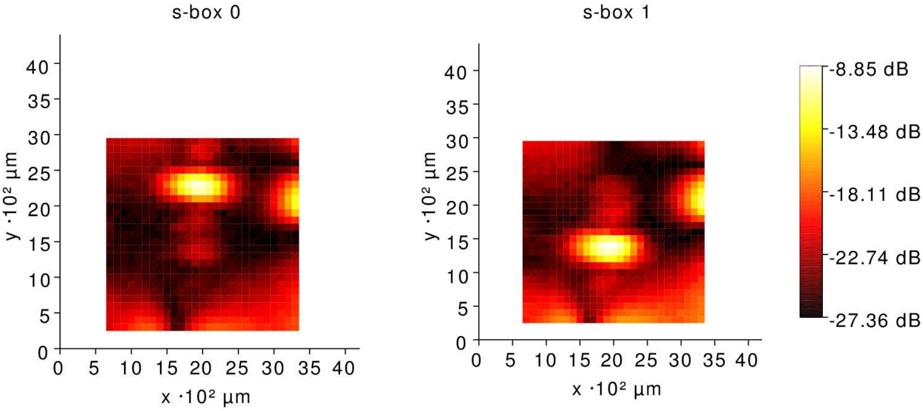


Fig. 10. SNR using vertical coil in the x-direction (frontside)

3.6 Frequency Domain

Frequency filtering of EM signals is generally promising for selecting data-dependent signals. We used the Fast Fourier Transform (FFT) to calculate frequency spectra at several positions on the die using the measurements from the backside using the horizontal coil. We cut the measurement into 753 parts corresponding to the repeated s-box 0 value sequence. Then, we performed FFT transformations and decreased phase and amplitude noise by averaging the frequency spectra. Figure 11 shows the resulting power spectral density at three different positions on the die. Our results show frequency components up to 2.5 GHz because of the sampling rate of 5 GS/s. The first position is close to the s-box structure 0. The second one is close to an area where we assume a big influence of the power supply, and the third is at a position which exhibited a low

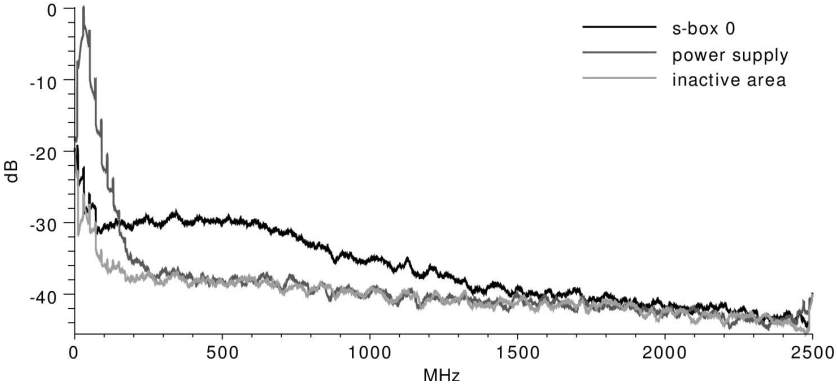


Fig. 11. Frequency spectra at different positions (backside, horizontal coil)

SNR in Fig. 9. A comparison of the position close to s-box 0 to the position with minimum SNR leads to the observation, that the information leakage is contained in a frequency span between 100 MHz and 1.5 GHz. The upper frequency limit might be determined by the bandwidth of our measurement equipment. The position close to the power supply exhibits higher amplitudes, thus, information leakage, in a frequency span lower than 200 MHz. We argue that this is due the low-pass characteristic of the series of on-chip capacitances between the leaking s-box structure and the power supply. Therefore, a high-pass filter with a cut-off frequency of 200 MHz could be applied to focus on localized signals and this fact could be used as a heuristic to find exploitable positions on the circuit.

3.7 Trace Compression

Trace compression is popular to reduce data and computational complexity during side-channel analysis. We evaluated four methods which reduce 250 samples per clock cycle to a single value and repeated the CPA from Sect. 3.2 to benchmark the outcome. During *maximum extraction*, one sample index is selected which exhibits the maximum mean value over all cycles. This resulted in a maximum correlation of 0.086. *Peak-to-peak extraction* derives the distance between the two values with highest and lowest mean over all cycles. This resulted in a maximum correlation of 0.030. *Sum-of-absolutes* integrates absolute values over whole clock cycles. This resulted in a maximum correlation of 0.063. *Sum-of-squares* integrates squared values over whole clock cycles and resulted in a maximum correlation of 0.086. The original traces lead to a maximum correlation coefficient of 0.234 in Sect. 3.2, Fig. 8.

Hence, all compression methods resulted in significantly lower correlation coefficients and we conclude that trace compression is generally inadvisable when analyzing high-resolution EM measurements. However, a compression method, which simply removes unimportant parts in each clock cycle, e.g., samples between 75 ns and 250 ns in Fig. 5, will not influence the outcome. This becomes obvious from Fig. 7, where the correlation is detectable in the first part of the cycle only.

3.8 Current Consumption versus Electromagnetic Field

In our current consumption measurement, the signals from consecutive clock cycles interfere which decreases the SNR due to additional data-dependent switching noise from other cycles. This is caused be the low bandwidth in the supply network containing on- and off-chip capacitances and inductances [8]. We excluded the lower bandwidth of the differential probe as a cause by analyzing the field of bonding wires of the supply exhibiting the same interference. As expected we found that the signal leakage is detectable over the whole clock cycle almost constantly instead of just during a short time after the active edge. This makes current consumption measurements more robust against misalignment in differential attack settings. We detected a maximum SNR of 0.9 dB and a maximum correlation coefficient of 0.094. This is significantly lower than the maximum observed SNR of 4.7 dB and maximum correlation coefficient of 0.234 in the case of high-resolution EM measurement. We assume that this is due to the overlap and additional switching noise from other circuit parts. *We conclude, that localized EM measurements prevent interference of signals across multiple cycles at high clock frequencies of the design under test and provides significantly higher SNRs.* However, this requires to be able to position the probe correctly.

4 Conclusion

We suggest that some of our conclusions about high-resolution EM measurements can be generalized to other integrated circuits such as FPGAs or ASICs. Localized measurements are only superior, if correct positions for measurement are known. Then, the signal of circuit parts can be recorded selectively and with higher SNRs without interference due to low-pass behavior of the supply network. Measurements from the die frontside lead to better results, and we generally recommend semi-invasive decapsulation to achieve minimal probe-to-die distances. The horizontal probe provided higher SNRs while the vertical coil could be used to increase selectivity. High sampling rates, e.g., at least > 1 GS/s will be required in most cases and compression of traces is generally not recommended.

References

1. Agrawal, D., Archambeault, B., Rao, J., Rohatgi, P.: The EM side-channel(s). In: Kaliski Jr., B.S., Koç, Ç.K., Paar, C. (eds.) CHES 2002. LNCS, vol. 2523, pp. 29–45. Springer, Heidelberg (2003)
2. Canright, D.: A very compact S-box for AES. In: Rao, J.R., Sunar, B. (eds.) CHES 2005. LNCS, vol. 3659, pp. 441–455. Springer, Heidelberg (2005)

3. De Mulder, E., Buysschaert, P., Ors, S., Delmotte, P., Preneel, B., Vandenbosch, G., Verbauwhede, I.: Electromagnetic analysis attack on an fpga implementation of an elliptic curve cryptosystem. In: The International Conference on Computer as a Tool, EUROCON 2005, vol. 2, pp. 1879–1882 (November 2005)
4. Gandolfi, K., Mourtel, C., Olivier, F.: Electromagnetic analysis: Concrete results. In: Koç, Ç.K., Naccache, D., Paar, C. (eds.) CHES 2001. LNCS, vol. 2162, pp. 251–261. Springer, Heidelberg (2001)
5. He, W., de la Torre, E., Riesgo, T.: An interleaved EPE-immune PA-DPL structure for resisting concentrated EM side channel attacks on FPGA implementation. In: Schindler, W., Huss, S.A. (eds.) COSADE 2012. LNCS, vol. 7275, pp. 39–53. Springer, Heidelberg (2012)
6. Heyszl, J., Mangard, S., Heinz, B., Stumpf, F., Sigl, G.: Localized electromagnetic analysis of cryptographic implementations. In: Dunkelman, O. (ed.) CT-RSA 2012. LNCS, vol. 7178, pp. 231–244. Springer, Heidelberg (2012)
7. Kirschbaum, M., Schmidt, J.M.: Learning from electromagnetic emanations - a case study for iMDPL. In: Workshop Proceedings COSADE 2011, pp. 50–55 (2011)
8. Mangard, S., Oswald, E., Popp, T.: Power Analysis Attacks: Revealing the Secrets of Smart Cards (Advances in Information Security). Springer-Verlag New York, Inc., Secaucus (2007)
9. Peeters, E., Standaert, F.X., Quisquater, J.J.: Power and electromagnetic analysis: improved model, consequences and comparisons. Integr. VLSI J. 40(1), 52–60 (2007)
10. Quisquater, J.-J., Samyde, D.: Electromagnetic analysis (EMA): Measures and counter-measures for smart card. In: Attali, S., Jensen, T. (eds.) E-smart 2001. LNCS, vol. 2140, pp. 200–210. Springer, Heidelberg (2001)
11. Real, D., Valette, F., Drissi, M.: Enhancing correlation electromagnetic attack using planar near-field cartography. In: Design, Automation Test in Europe Conference Exhibition, DATE 2009, pp. 628–633 (April 2009)
12. Sauvage, L., Guilley, S., Mathieu, Y.: Electromagnetic radiations of fpgas: High spatial resolution cartography and attack on a cryptographic module. ACM Trans. Reconfigurable Technol. Syst. 2, 4:1–4:24 (2009)
13. Standaert, F.-X., Archambeau, C.: Using subspace-based template attacks to compare and combine power and electromagnetic information leakages. In: Oswald, E., Rohatgi, P. (eds.) CHES 2008. LNCS, vol. 5154, pp. 411–425. Springer, Heidelberg (2008)

Efficient Template Attacks Based on Probabilistic Multi-class Support Vector Machines

Timo Bartkewitz and Kerstin Lemke-Rust

Department of Computer Science,
Bonn-Rhine-Sieg University of Applied Sciences,
Grantham-Allee 20, 53757 Sankt Augustin, Germany
{timo.bartkewitz,kerstin.lemke-rust}@h-brs.de

Abstract. Common template attacks are probabilistic relying on the multivariate Gaussian distribution regarding the noise of the device under attack. Though this is a realistic assumption, numerical problems are likely to occur in practice due to evaluation in higher dimensions. To avoid this, a feature selection is applied to identify points in time that contribute most information to an attack. An alternative to common template attacks is to apply machine learning in form of support vector machines (SVMs). Recent works brought out approaches that produce comparable results, respectively better in the presence of noise, but still not optimal in terms of efficiency and performance. In this work we show how to adapt the SVM template approach in order to considerably reduce the effort while carrying out the attack and how to better exploit the side-channel information under the assumption of an attack model with a strict order, *e.g.* Hamming weight model.

Keywords: Power Analysis, Template Attacks, Machine Learning, Support Vector Machines.

1 Introduction

Side-channel attacks are still, even after years of intensive research, a serious threat to cryptographic devices. New attacks challenge new algorithms and respective countermeasures. Generally, side-channel attacks rely on the assumption that any electronic device provides physically observable information on secrets, usually a cryptographic key, embedded in the device. The spot from which the information is extracted is referred to as a side-channel. There exist several distinct kinds of side-channels, for instance the power consumption, electromagnetic emanation, or timings. In this work we focus on template attacks (TAs) [5], a powerful side-channel attack since they are supposed to retrieve the most information of a side-channel leakage. TAs require a profiling phase to model the noise of the device, assuming the noise to be multivariate Gaussian distributed. In the subsequent characterization phase any dependency can be found using a likelihood approach for similarities between different points in time of recorded

S. Mangard (Ed.): CARDIS 2012, LNCS 7771, pp. 263–276, 2013.

power consumption traces. Therefore, TAs are dependent on these points in time which are assumed to contain the most information given by the maximum key-dependent variance. Besides the selection of such points which is often denoted *feature selection*, one is also concerned with potential numerical problems and the question whether the assumed noise model is adequate or not in order to mount a successful TA.

To overcome these issues recent works investigated alternatives to the multivariate Gaussian approach. Machine learning in the form of support vector machines (SVMs) is one of these promising alternatives. The SVM is a linear binary classifier that decides to which of two classes an input vector belongs, based on classified training data. Further, the SVM is independent of a certain noise distribution. In [12] the authors focused on SVMs amongst other machine learners and present attacks that predict key bits of a DES implementation. In [10, 11] the authors exclusively focused on SVMs concentrating on the applicability for template attacks. However, they did not provide an attack. In [9] the authors extend the SVM approach from a single-bit model to a multi-bit model and consequently introduce probabilistic multi-class SVMs. They showed that the SVM based template attack outperforms the Gaussian approach in the presence of noise. Nevertheless, their approach is not optimal in terms of efficiency and exploited side-channel leakage.

Our contribution: In this work we carry the SVM template attacks forward. We first show how a tailored multi-class strategy can considerably reduce the effort during the profiling and characterization phase. Second, we show how to better exploit the side-channel leakage, including the introduction of a dedicated feature selection, and compare it against several other TA approaches. We therefore assume an attack model with a strict order, *e.g.* Hamming weight model.

Organization of the paper: This paper is organized as follows: Section 2 briefly introduces template attacks based on the Gaussian approach. Section 3 provides a broad overview on support vector machines. In Section 4 we describe how SVM based template attacks can be improved by means of efficiency and performance. Finally, Section 5 reports our experimental results before we conclude in Section 6.

2 Template Attacks

Template attacks usually consist of three steps. The first step is to select points in time (often called *points of interest* or *features*) that are supposed to contain a considerable proportion of the leakage information. Afterwards templates are built involving the power consumption of a reference device, similar to the target device, that is under the full control of the attacker. Eventually, the attack on the target device is carried out by matching its power consumption leakage to the templates.

Feature Selection. The selection of points of interest within power traces is the first issue in TAs we are concerned with. There are several methods to obtain a set of points that could lead to a successful attack. Primarily, the points are selected according to their key-dependent variability, including known-key DPA [18], pair-wise distance to the mean vectors [5], or using the sum of squared pair-wise T-differences [8]. A more systematic approach is the principal subspace-based TA where the principal component analysis (PCA) is applied to transform the side-channel data into a low-dimensional subspace, figuring out the optimal linear combination of points in time which show maximum variance with respect to the side-channel leakage [1].

Template Building. The application of templates implies two successive phases. In the first phase templates are built according to N_p selected points of interest from several measured power traces $\{\mathbf{t}^i_{d,k}\}^{N_\mathbf{t}}_{i=1}$ that are correlated to a function which involve both, known input data d and a key k, respectively a part of it. The traces are assumed to be drawn from a multivariate Gaussian distribution $\mathcal{N}(\mathbf{t}^i_{d,k}|\boldsymbol{\mu}_{d,k}, \boldsymbol{\Sigma}_{d,k})$. Therefore, a single template $\boldsymbol{\tau}_{d,k}$ is equivalent to an estimation of the mean $\boldsymbol{\mu}_{d,k}$ and the covariance matrix $\boldsymbol{\Sigma}_{d,k}$ based on the selected points, and corresponding to different pairs of $\{d, k\}$.

Key Recovery Attack. In the second phase, given a new power trace $\mathbf{t}^{\text{new}}_{d,k}$ to be characterized, the multivariate Gaussian probability density function

$$p(\mathbf{t}^{\text{new}}_{d,k}|\boldsymbol{\tau}_{d,k}) = \frac{1}{\sqrt{(2\pi)^{N_p}|\boldsymbol{\Sigma}_{d,k}|}} \exp\left(-\frac{1}{2}(\mathbf{t}^{\text{new}}_{d,k} - \boldsymbol{\mu}_{d,k})^\top \boldsymbol{\Sigma}^{-1}_{d,k}(\mathbf{t}^{\text{new}}_{d,k} - \boldsymbol{\mu}_{d,k})\right) \quad (1)$$

is evaluated for each template. The maximum likelihood approach provides the best fit, hence the higher the probability density the better the trace $\mathbf{t}_{d,k}$ fits the respective template.

In order to avoid numerical problems in practice, mainly due to the inversion of $\boldsymbol{\Sigma}_{d,k}$, one can omit the covariances (off-diagonal values of $\boldsymbol{\Sigma}_{d,k}$) to obtain so called *reduced templates* [14]. This leads to a univariate approach since (1) can then be rewritten as the product of the probability densities at each point of interest

$$p(\mathbf{t}^{\text{new}}_{d,k}|\boldsymbol{\tau}_{d,k}) = \prod_{i=1}^{N_p} \frac{1}{\sqrt{2\pi\sigma_i^2}} \exp\left(-\frac{1}{2}\frac{(t^{\text{new}}_{i,d,k} - \mu_{i,d,k})^2}{\sigma_i^2}\right) \quad (2)$$

considering them as being independent and thus uncorrelated.

In practice it is often not sufficient to recover the key, or a part of it, based on a single trace to be characterized but based on a few traces. Usually, we apply Bayes' theorem [14]

$$p(k^*|\mathbf{t}^{\text{new}}_{d,k}) = \frac{p(\mathbf{t}^{\text{new}}_{d,k}|\boldsymbol{\tau}_{d,k}) \cdot p(k^*)}{p(\mathbf{t}^{\text{new}}_{d,k})} \quad (3)$$

in order to determine by which key the traces to be characterized were generated. Here, $p(k^*)$ is the prior probability and $p(k^*|\mathbf{t}^{\text{new}}_{d,k})$ the posterior probability of each key candidate k^*.

3 Binary Support Vector Machines

Support vector machines are suitable for solving classification, regression, and pattern detection problems and belong to the category of *sparse kernel machines.* Originally described in [20], the SVM is a non-probabilistic, linear, binary-class decision machine whose output is a class label. The SVM is related to supervised learning methods whose determination of the model parameters correspond to convex optimization problems. This section follows the explanations in [2].

3.1 Mathematical Background of Binary SVMs

A binary-class classification problem can be described by a linear discriminant function of the form

$$y(\mathbf{x}) = \mathbf{w}^\top \mathbf{x} + b \tag{4}$$

where $\mathbf{w}$ is a *weight vector*, and b is a *bias.* An input vector $\mathbf{x}$ is assigned to class C^- if $y(\mathbf{x}) < 0$ and to C^+ otherwise. Hence, the decision boundary corresponds to the $(D-1)$-dimensional hyperplane within the D-dimensional input space, *i.e.* $y(\mathbf{x}) = 0$. Since $\mathbf{w}^\top \mathbf{x} = 0$ for every $\mathbf{x}$ lying on the decision boundary, $\mathbf{w}$ is orthogonal to every vector on the decision boundary and hence the normal vector of the hyperplane. With the same argument, bias $b = -\mathbf{w}^\top \mathbf{x}$ for every $\mathbf{x}$ on the decision boundary. Suppose the training set consists of N input vectors (row vectors) $\mathbf{x}_1, \ldots, \mathbf{x}_N$ (vectors with an index belong to the training set in the remainder) where each vector is associated to a class label $c_i \in \{-1, 1\}$. New vectors $\mathbf{x}$ are accordingly classified by the sign of $y(\mathbf{x})$. For the moment, it is assumed that the training set is linearly separable within the input space D, which means we can find a pair $(\mathbf{w}, b)$ such that (4) satisfies $c_i y(\mathbf{x}_i) > 0$ for all training vectors. That is, every training vector $\mathbf{x}_i$ is correctly classified. Naturally, we can find several pairs that separate the training set exactly but not every solution will give the smallest generalization error [2] that states the performance of classifying new vectors . In support vector machines this is solved by introducing the approach of the margin which embodies the smallest distance between the decision boundary to any input vector (Fig. 1). The best solution is given by the pair $(\mathbf{w}, b)$ for which this margin is maximized. The orthogonal distance of a vector $\mathbf{x}$ to the hyperplane is given by $y(\mathbf{x}) / \|\mathbf{w}\|$ ($\|\bullet\|$ denotes the *Euclidean norm*) and under the general constraint $c_i y(\mathbf{x}) > 0$ the maximum margin can be achieved by finding

$$\arg\max_{\mathbf{w},b} \{\min_i [c_i(\mathbf{w}^\top \mathbf{x}_i + b)]\}. \tag{5}$$

Finding a direct solution would be too complex. Since rescaling of $\mathbf{w}$ and b does not affect the distance from any input vector to the hyperplane, we set

$$c_i(\mathbf{w}^\top \mathbf{x}_i + b) \geq 1, \quad i = 1, \ldots, N \tag{6}$$

which means that for vectors that lie on the margin around the decision boundary the equality holds. Consequently, the optimization problem has been reduced

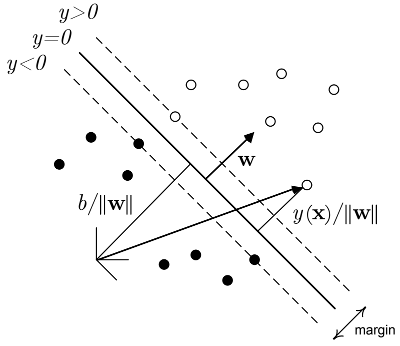


Fig. 1. Geometry of the separating hyperplane in support vector machines

to maximize $\|\mathbf{w}\|^{-1}$ which is equivalent to minimizing $\|\mathbf{w}\|^2$. This is a *quadratic programming* problem that can be solved by applying the method of *Lagrange multipliers* a_i, with respect to the *Karush-Kuhn-Tucker* (KKT) conditions [2]. The optimal solution of this Lagrangian optimization problem yields a representation of (4), *s.t.* $\mathbf{w} = \sum_{i=1}^{N} a_i c_i \mathbf{x}_i$ where the vectors $\mathbf{x}_i$ for which $a_i > 0$ are called *support vectors*. Hence, to classify new vectors $\mathbf{x}$ we obtain

$$y(\mathbf{x}) = \sum_{i \in \mathcal{S}}^{N} a_i c_i \mathbf{x}_i^\top \mathbf{x} + b \quad \text{where} \quad b = \frac{1}{N_{\mathcal{S}}} \sum_{i \in \mathcal{S}} (c_i - \sum_{j \in \mathcal{S}} a_j c_j \mathbf{x}_i^\top \mathbf{x}_j) \tag{7}$$

and $\mathcal{S}$ is the set of indices of the support vectors, respectively $N_{\mathcal{S}}$ the number of support vectors.

3.2 Non-linear Classification: Introduction of a Kernel

So far it was assumed that the training set is linearly separable within the input space D. If that does not hold the training set may be separable in the higher dimensional *feature space* $F > D$, not to be confused with feature selection. Therefore, the input vectors are transformed into that feature space by $\Phi(\mathbf{x})$ which gives a vector product of the form $\Phi(\mathbf{x}_1)^\top \Phi(\mathbf{x}_2)$ for $\mathbf{x}_1, \mathbf{x}_2 \in \mathbb{R}^D$. A direct solution is computationally very intensive. Hence, a kernel function is applied that behaves exactly like the vector product in F without even knowing the concrete feature space, but also without having influence on the resulting dimension.

3.3 Non-separable Case: Introduction of a Soft-Margin

A linear separation, in the input space or in the feature space, can lead to a poor generalization (large generalization error) in the case of overlapping class distributions. Therefore, it makes sense to allow for misclassification of some training vectors to achieve a separation anyway. To do so, slack variables and a trade-off parameter γ, often denoted *box constraint*, are introduced to penalize misclassification leading to two kinds of support vectors (see [2] for details). Those where $a_i < \gamma$, support vectors which lie on the margin, and those where $a_i = \gamma$ which are support vectors that are either correctly classified but inside the margin or misclassified. For $\gamma \to \infty$ the penalty prohibits misclassified vectors and thus recovers the strict margin.

3.4 SVM Training and Classification

Training a support vector machine means solving the Lagrangian optimization problem for the given training set. There exist a specific approach called *sequential minimal optimization* (SMO) [16] that breaks down the optimization problem into many smallest problems where each of which only considers two Lagrange multipliers at a time. The Lagrange multipliers are jointly optimized under a linear equality constraint. The subsequent multipliers to be optimized are then chosen heuristically. The SVM classification is done through the evaluation of (7) making use of the parameters of the SMO training.

4 Template Attacks Using Support Vector Machines

The approach is similar to common template attacks. The posterior key probabilities are successively updated with each characterization trace applying Bayes' theorem. Since TAs are a multi-class classification problem it is essential to turn the actual binary SVM into a probabilistic multi-class SVM. Contrary to previous work [9] which follows a general probabilistic approach, we subsequently present probabilistic multi-class SVMs tailored to fit template attacks.

4.1 Probabilistic Support Vector Machines

In order to use SVMs in an aggregated probabilistic approach, probabilistic decisions of the class label c for new vectors are necessary. Maintaining the sparseness property of SVMs it is proposed in [17] to fit a logistic sigmoid function to the outputs $y(\mathbf{x})$ of an already trained binary SVM to give the posterior conditional probability

$$p(c = 1|\mathbf{x}) = \frac{1}{1 + \exp(A \cdot y(\mathbf{x}) + B)} \tag{8}$$

that $\mathbf{x}$ belongs to the class $c = 1$. Clearly, $p(c = -1|\mathbf{x}) = 1 - p(c = 1|\mathbf{x})$. A second training set should be involved to avoid severe overfitting (this will be discussed in Section 5). The parameters A and B are found by minimizing the cross-entropy error of the training set

$$\arg\min_{A,B} - \sum_{i=1}^{N} t_i \log(p_i) + (1 - t_i) \log(1 - p_i) \tag{9}$$

where $t_i = (\text{sign}[y(\mathbf{x}_i)] + 1)/2$ and $p_i = p(c = 1|\mathbf{x}_i)$. Nevertheless, solving this optimization problem in a numerical stable way is proposed in [13].

4.2 Our Probabilistic SVM Multi-class Approach

Previous work [7] investigates methods based on two strategies to turn a binary SVM into a multi-class SVM. The *one-versus-all* strategy uses binary SVMs for separating one multi-class from the joint set of all other multi-classes, whereas the *one-versus-one* strategy applies binary SVMs for pair-wise distinguishing the multi-classes. In [9] the authors preferred the *one-versus-one* strategy in order to mount their attacks. In this work, we suggest our own method tailored for template attacks with regards to the attack model.

Suppose the TA classification problem implies M distinct classes. Then, given a new input vector $\mathbf{x}$, we aim for posterior conditional probabilities $p(\Omega_l|\mathbf{x})$ for $l = 1, \ldots, M$ and Ω_l is the l_{th} multi-class label. The classification is enabled by a training set $\widetilde{\mathbf{X}}$ (traces) where for each training vector $\tilde{\mathbf{x}}_j$ for $j = 1, \ldots, N$ the correct multi-class label Ω_l, relying on a certain attack model, is known.

Attack Model with a Strict Order. We assume that the side-channel leakage leads to splitting the measurement samples into a strict order according to the known multi-class labels at the leakage-dependent points in time. More precisely, let $x_{\Omega_l,t}$ be an arbitrary scalar at point t of a trace that belongs to label Ω_l, then for each leakage-dependent point t we assume a strict ordering of the labels, *i.e.* either $x_{\Omega_1,t} < x_{\Omega_2,t} < \ldots < x_{\Omega_M,t}$ or, alternatively, $x_{\Omega_1,t} > x_{\Omega_2,t} > \ldots > x_{\Omega_M,t}$. A well-known example of such an ordered leakage is the Hamming weight model [14]. With such an attack model we train $M-1$ SVMs using the training set $\widetilde{\mathbf{X}}$ and introduce binary-class helper labels $c_{i,j}$ to the training vectors, such that

$$c_{i,j} = \begin{cases} 1, & \{\tilde{\mathbf{x}}_j|\tilde{\mathbf{x}}_j \text{ belongs to } \Omega_l \text{ with } l \leq i \\ -1, & else \end{cases}, \quad \begin{matrix} j = 1, \ldots, N \\ l = 1, \ldots, M \end{matrix} \tag{10}$$

when the i_{th} SVM is under training, $i = 1, \ldots, M-1$. These helper labels convert the attack model multi-classes to binary-classes as requested by the SVM classification model. Using the i_{th} SVM afterwards to classify a new vector $\mathbf{x}$ we get the multi-class overlapping probabilities $p(\bigcup_{l=1}^{i} \Omega_l|\mathbf{x})$. That is, the i_{th} SVM gives the probability that $\mathbf{x}$ belongs to the classes before the i_{th} hyperplane. Figuratively, we construct the hyperplanes between the multi-class labels from the left to the right as depicted in Figure 2. Recalling that the probabilities rely on a distance measure between $\mathbf{x}$ and the separating hyperplane, each consecutive probability $p(\bigcup_{l=1}^{i+1} \Omega_l|\mathbf{x}), p(\bigcup_{l=1}^{i+2} \Omega_l|\mathbf{x}), \ldots, p(\bigcup_{l=1}^{M-1} \Omega_l|\mathbf{x})$ is even higher once $\mathbf{x}$ was classified to belong to a multi-class before the i_{th} hyperplane with a non-negligible probability (*cf.* Fig. 2). This is due to the fact that the distance grows in a positive manner and thus the probability grows since the binary-class separation regarding $\mathbf{x}$ becomes even clearer. However, this is of course an undesired result. We can overcome this drawback by training, again, $M-1$ SVMs but this time starting with the last multi-class label Ω_M, *i.e.* using (10) with reversed signs. With this approach, going from the left to the right first and then vice versa, we surround the correct multi-class label by two consecutive hyperplanes. In fact we do not need to train new SVMs since the separating hyperplanes

1	1	1	-1	-1	-1	-1		-1
Ω_1	Ω_2	Ω_3	Ω_4	Ω_5	Ω_6	Ω_7	$\dots$	Ω_M

Fig. 2. For instance, the *third* support vector machine is trained which corresponds to the hyperplane separating $\bigcup_{l=1}^{3} \Omega_l$ and $\bigcup_{l=4}^{M} \Omega_l$. The binary-class helper labels $c_{3,j}$ for the training vectors $\mathbf{x}_j$ are given on top. Training vectors that belong to the multi-classes before the hyperplane are classified with helper label 1, all others with -1.

did not change, but merely use the complementary probabilities deferred by one multi-class due to the surrounding. Suppose $\mathbf{x}$ indeed belongs to Ω_i, then $\mathbf{x}$ is right-bounded by the hyperplane i and left-bounded by the hyperplane $i-1$ and thus related to the probabilities $p(\bigcup_{l=1}^{i} \Omega_l|\mathbf{x})$ and $1 - p(\bigcup_{l=1}^{i-1} \Omega_l|\mathbf{x})$. Hence, we suggest using

$$p(\Omega_i|\mathbf{x}) = p(\bigcup_{l=1}^{i} \Omega_l|\mathbf{x}) \cdot \left[1 - p(\bigcup_{l=1}^{i-1} \Omega_l|\mathbf{x})\right] \tag{11}$$

$$= \frac{1 - \frac{1}{1+\exp(A_{i-1}\cdot y_{i-1}(\mathbf{x})+B_{i-1})}}{1 + \exp(A_i \cdot y_i(\mathbf{x}) + B_i)}, \quad 1 < i < M, \tag{12}$$

$$p(\Omega_1|\mathbf{x}) = p(\bigcup_{l=1}^{1} \Omega_l|\mathbf{x}) = \frac{1}{1 + \exp(A_1 \cdot y_1(\mathbf{x}) + B_1)}, \tag{13}$$

$$\text{and } p(\Omega_M|\mathbf{x}) = 1 - p(\bigcup_{l=1}^{M-1} \Omega_l|\mathbf{x}) = 1 - \frac{1}{1 + \exp(A_{M-1} \cdot y_{M-1}(\mathbf{x}) + B_{M-1})} \tag{14}$$

being the posterior conditional class probabilities.

Attack Model without a Strict Order. Technically speaking, if an attack model with a strict order is not applicable, *i.e.* scalars that belong to different multi-classes are equal on average, a separation is impossible no matter which method is used. Nevertheless, in practice one could try to combine (leave out) equivalent multi-classes or use a one-versus-one strategy. The latter case means separating each pair (Ω_i, Ω_j) for $i < j \leq M$ which in turn results in training $M(M-1)/2$ SVMs. The posterior conditional class probabilities are then given by

$$p(\Omega_i|\mathbf{x}) = \prod_{j \neq i} p[(\Omega_i, \Omega_j)|\mathbf{x} \text{ belongs to } \Omega_i]. \tag{15}$$

4.3 SVM Based Templates

As in the multivariate Gaussian approach, templates are built on the reference traces first, here called the training set, by training $M-1$ or $M(M-1)/2$ support vector machines where M denotes the number of classes according to our attack model. Afterwards, we fit the sigmoid function with classification values from the SVMs involving a second set of reference traces. Hence, a single template contains the Lagrange multipliers $\mathbf{a}_i$, the support vectors $\mathbf{X}_{\mathcal{S}i} \subset \widetilde{\mathbf{X}}$, and the bias b_i plus the values A_i and B_i for the templates $i, \ldots, M-1$, respectively for $i, \ldots, M(M-1)/2$. Please note that in the SVM approach templates are to be characterized by the class separators and not by the class representatives.

4.4 Considering Feature Selection

Primarily, a prior feature selection is a dimensionality reduction to help figuring out the most discriminative features in a given data set. However, this generally means a loss of information that in turn affects the prediction performance of classifiers.

For the Gaussian TA approach a priorly executed feature selection is certainly essential since it avoids numerical problems in practice which render the evaluation of probability densities impossible. Further, it is assumed that the exploitable side-channel leakage is hidden locally in the variability of only a few points in time with respect to such probability densities [14]. Hence, the loss of information is accepted and considered as loss of noise, respectively loss of information that marginally contributes to the attack.

This looks different for the SVM approach where we aim for inter-class separation instead of intra-class densities. On the one side numerical problems due to high dimensioned data do not occur and on the other dimensionality reduction methods such as PCA likely jeopardize the optimal performance of SVMs in other applications [21]. In contrast to previous works [9–12] we suppose using the linear kernel with a dedicated subsequent feature selection, called *normal-based feature selection*, is optimal while presuming a linear attack model in template attacks (*cf.* Section 4.2). The normal-based feature selection retains points in time according to the weight vector $\mathbf{w}$ (*cf.* Section 3.1). It can be shown [3] that a feature j which corresponds to a higher absolute value $|w_j|$ are more influential in determining the optimal margin and thus improves classification performance. Since we train several SVMs according to the attack model we disregard features by setting the respective weights of $\mathbf{w}$ to zero instead of removing them from the data set. One may argue that prior feature selection has the same effect but the Lagrange multipliers $\mathbf{a}_i$ that significantly determine the weight vector are still found using the complete data set.

5 Experimental Results

For our experiments we used a Microchip PIC18F2520 microcontroller [15] running at 3.68 MHz. The power traces were acquired with a PicoScope *5203* and a sampling rate at 125 MS/s using a 1 Ω-resistor in the ground line. The traces were compressed with peak extraction [14] representing the full substitution layer (S-boxes) in the first round of the AES (Advanced Encryption Standard). Therefore, we chose the attack model to be the Hamming weight of the S-box output. We thus have nine classes with a strict order (*cf.* Sec. 4.2).

To produce comparable results to [9] we choose the *guessing entropy* [19] to evaluate the attack performance. It states the average position of the correct key within a descending ranking of the probabilities of all possible keys. However, in order to introduce our adaptations we used our own SVM implementation as described throughout this paper (see Appendix A for pseudo code) instead of the C-SVC implementation of the LIBSVM library [4]. C-SVC also applies SMO for training and the same probability model for classification, thus both implementations are comparable.

In the following, we validate our SVM template attack against variants of it, the SVM TA from [9], the common TA, and the common TA with prior PCA. Our approach implies the here proposed multi-class method and the linear kernel with the subsequent normal-based feature selection. As variants we replaced the normal-based feature selection with prior feature selection, namely known-key DPA (kkDPA) where the points in time with the highest correlation were taken, respectively with the application of PCA. Further, we include the common template attack with both known-key DPA and prior PCA. The SVM TA from [9] uses the RBF kernel and known-key DPA. They also suggest an empirical determined cost factor (box constraint) of $\gamma = 10$, respectively $\gamma = 1$ for noisy measurements, and an empirical determined termination criterion of 0.02 that states the fraction of vectors that are allowed to violate the KKT conditions. In our experiments, however, we involve $\gamma = 1$ and termination criterion of zero, as recommended in [6], except for the attack from [9]. In order to simulate noisy measurements we add Gaussian noise to the power traces, in particular Gaussian noise with a standard deviation of $\sigma_{n_g} = 5$. We determined the intrinsic noise of our measured power traces to be $\sigma_{n_0} \approx 0.7$.

Initially, we want to show how to disregard features with the help of normal-based feature selection. As can be seen in Figure 3 the weights are linearly

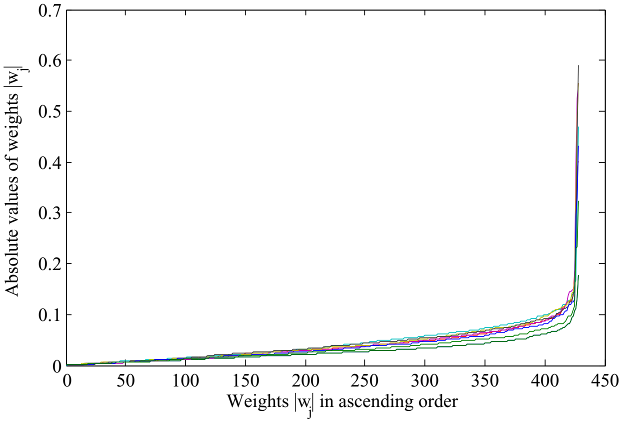


Fig. 3. Absolute values of the weights in ascending order. The weight vectors were obtained from training 8 SVMs (HW attack model).

increasing except the last few weights which increase exponentially. These are the weights we retain and thus all the others were set to zero. In our experiments we retain 8 weights. We also used 8 components from PCA and 8 points in time with the highest correlation from known-key DPA.

Our performance comparison considers each template attack involving the required profiling, respectively characterization traces to reach a guessing entropy of one. Thus, it states the minimum effort to always recover the correct key with respect to our experiments. Furthermore, we depict the guessing entropies of the attacks while our attacks possesses a guessing entropy of one.

Table 1. Comparison of template attacks depicting the required amount of characterization traces along an increasing profiling base (traces per HW) to reach a guessing entropy of one. The lower table depicts the guessing entropies while our TA reaches a guessing entropy of one. The traces were involved with their intrinsic noise $\sigma_{n_0} \approx 0.7$.

Profiling	SVM based TA				Common TA	
base	Our TA	linear & kkDPA	linear & PCA	Heuser et al. [9]	kkDPA	PCA
10	33	98	71	289	–	–
20	22	67	33	147	92	53
40	21	63	26	139	63	37
60	19	61	21	121	59	23
80	15	44	17	116	55	19
100	13	39	15	111	51	16
10	1	2.59	1.17	18.54	–	–
20	1	2.57	1.15	16.75	7.51	1.62
40	1	2.58	1.15	18.72	6.58	1.15
60	1	2.58	1.08	18.10	4.91	1.10
80	1	2.08	1.08	17.53	3.46	1.10
100	1	2.08	1.08	18.43	3.74	1.08

Table 1 shows the results considering the original traces. It is observable that template attacks based on SVMs perform better than common TAs. As expected each attack requires less characterization traces with an increasing profiling base where best performance is almost reached with a profiling base containing 100 traces per Hamming weight. However, with a too small profiling base (10 traces per HW) the common attacks fail due to numerical problems caused by the matrix inversion with the Gauss-Jordan algorithm. Our approach performs well, especially with a small profiling base, whereas the attack proposed in [9] using the RBF-kernel performs only suboptimal. This observation is even more aggravated when comparing the guessing entropies. When our TA reaches a guessing entropy of one the other attacks reduce the key space to at most four except the RBF kernel based attack that reduces the key space to about 18.

Next, we evaluate the performance in the presence of higher noise. Table 2 depicts that in this case PCA is not the optimal choice for feature selection. Both TA approaches lead to inferior results when using PCA instead of known-key

Table 2. Comparison of template attacks depicting the required amount of characterization traces along an increasing profiling base (traces per HW) to reach a guessing entropy of one. The lower table depicts the guessing entropies while our TA reaches a guessing entropy of one. The traces were involved with added noise σ_{n_g}, thus $\sigma_{n_1} \approx 5.7$.

Profiling	SVM based TA				Common TA	
base	Our TA	linear & kkDPA	linear & PCA	Heuser et al. [9]	kkDPA	PCA
10	81	121	149	1100	–	–
20	62	93	112	365	650	920
40	56	84	94	312	158	344
60	51	73	84	153	112	146
80	46	62	79	144	96	124
100	43	58	73	138	92	121
10	1	1.18	1.66	41.25	–	–
20	1	1.14	1.62	13.54	15.524	14.37
40	1	1.12	1.54	7.8	11.22	12.02
60	1	1.12	1.52	7.75	4.91	5.55
80	1	1.12	1.46	7.57	2.95	3.58
100	1	1.12	1.36	7.58	2.96	2.81

DPA. Our approach still performs well but the results altogether are also a bit closer now. The RBF kernel based attack performs slightly better and can finally reduce the key space to eight.

Eventually, our findings indicate that SVM based template attacks do not perform best with the RBF kernel. Admittedly, our results concerning the RBF kernel were obtained using our multi-class strategy instead of the usual one-versus-one strategy but we suggest this has no crucial negative impact. Actually, the good performance of a linear kernel should not be surprising since template attacks usually imply a linear classification problem whereas the RBF kernel is appropriate for non-linear problems. The linear kernel also performs well with prior feature selection, *i.e.* known-key DPA and PCA but the normal-based feature selection is very simple, and furthermore it provides better results with a small profiling base. The computational effort of our SVM based attack is in the range of seconds and hence negligible when compared to common TAs.

6 Conclusion

In this work we showed how to improve the performance of template attacks based on support vector machines. Although previous works already demonstrated the advantages of such template attacks, their approaches were not optimal in the sense of efficiency and performance. First of all we proposed a multi-class method tailored for TAs which lead to training less SVMs under a attack model with a strict order, *e.g.* the Hamming weight model. Next, we showed that the subsequent feature selection after the training called normal-based feature selection together with the linear kernel leads to superior results than using it with a prior feature selection, namely known-key DPA or PCA, respectively.

Acknowledgements. The work described in this paper was supported by the German Federal Ministry of Education and Research (BMBF) through the EXSET project (promotional reference 01IS10026B). The information in this document reflects the views only of the authors.

References

1. Archambeau, C., Peeters, E., Standaert, F.-X., Quisquater, J.-J.: Template Attacks in Principal Subspaces. In: Goubin, L., Matsui, M. (eds.) CHES 2006. LNCS, vol. 4249, pp. 1–14. Springer, Heidelberg (2006)
2. Bishop, C.M.: Pattern Recognition and Machine Learning. Springer, New York (2006)
3. Brank, J., Grobelnik, M., Milic-Frayling, N., Mladenic, D.: Feature Selection Using Linear Support Vector Machines. No. MSR-TR-2002-63, Microsoft Research (2002)
4. Chang, C.C., Lin, C.J.: LIBSVM: A Library for Support Vector Machines. ACM Transactions on Intelligent Systems and Technology 2, 27:1–27:27 (2011), Software available at `http://www.csie.ntu.edu.tw/~cjlin/libsvm`
5. Chari, S., Rao, J., Rohatgi, P.: Template Attacks. In: Kaliski Jr., B.S., Koç, Ç.K., Paar, C. (eds.) CHES 2002. LNCS, vol. 2523, pp. 13–28. Springer, Heidelberg (2003)
6. Cristianini, N., Shawe-Taylor, J.: An Introduction to Support Vector Machines and Other Kernel-Based Learning Methods. Cambridge University Press, Cambridge (2000)
7. Duan, K.-B., Keerthi, S.S.: Which Is the Best Multiclass SVM Method? An Empirical Study. In: Oza, N.C., Polikar, R., Kittler, J., Roli, F. (eds.) MCS 2005. LNCS, vol. 3541, pp. 278–285. Springer, Heidelberg (2005)
8. Gierlichs, B., Lemke-Rust, K., Paar, C.: Templates vs. Stochastic Methods. In: Goubin, L., Matsui, M. (eds.) CHES 2006. LNCS, vol. 4249, pp. 15–29. Springer, Heidelberg (2006)
9. Heuser, A., Zohner, M.: Intelligent Machine Homicide. In: Schindler, W., Huss, S.A. (eds.) COSADE 2012. LNCS, vol. 7275, pp. 249–264. Springer, Heidelberg (2012)
10. Hospodar, G., De Mulder, E., Gierlichs, B., Vandewalle, J., Verbauwhede, I.: Least Squares Support Vector Machines for Side-Channel Analysis. In: COSADE 2011. CASED, Darmstadt (2011)
11. Hospodar, G., Gierlichs, B., De Mulder, E., Verbauwhede, I., Vandewalle, J.: Machine Learning in Side-channel Analysis: A First Study. Journal of Cryptographic Engineering 1, 293–302 (2011)
12. Lerman, L., Bontempi, G., Markowitch, O.: Side Channel Attack: An Approach Based on Machine Learning. In: COSADE 2011. CASED, Darmstadt (2011)
13. Lin, H.T., Lin, C.J., Weng, R.C.: A Note on Platt's Probabilistic Outputs for Support Vector Machines, vol. 68, pp. 267–276. Kluwer Academic Publishers, Hingham (2007)
14. Mangard, S., Oswald, E., Popp, T.: Power Analysis Attacks: Revealing the Secrets of Smart Cards. Springer, Heidelberg (2007)

15. Microchip Technology Inc.: PIC18F2420/2520/4420/4520 Data Sheet (2008)
16. Platt, J.C.: Fast Training of Support Vector Machines Using Sequential Minimal Optimization. In: Advances in Kernel Methods, pp. 185–208. MIT Press, Cambridge (1999)
17. Platt, J.C.: Probabilistic Outputs for Support Vector Machines and Comparisons to Regularized Likelihood Methods. In: Advances in Large Margin Classifiers, pp. 61–74. MIT Press, Cambridge (1999)
18. Rechberger, C., Oswald, E.: Practical Template Attacks. In: Lim, C.H., Yung, M. (eds.) WISA 2004. LNCS, vol. 3325, pp. 440–456. Springer, Heidelberg (2005)
19. Standaert, F.-X., Malkin, T.G., Yung, M.: A Unified Framework for the Analysis of Side-Channel Key Recovery Attacks. In: Joux, A. (ed.) EUROCRYPT 2009. LNCS, vol. 5479, pp. 443–461. Springer, Heidelberg (2009)
20. Vapnik, V.N.: The Nature of Statistical Learning Theory. Springer, New York (1995)
21. Zhang, Y., Schneider, J.G.: Projection Penalties: Dimension Reduction without Loss. In: Fürnkranz, J., Joachims, T. (eds.) ICML 2010, pp. 1223–1230. Omnipress (2010)

A Algorithms for SVM Based Template Attacks

Required adaptations for an attack model without a strict order infer from Section 4.2.

Algorithm 1. SVM Template Building

```
Input: Training set $\widetilde{\mathbf{X}}$ with $N$ traces related to labels $\Omega_1, \ldots, \Omega_M$, and constraint $\gamma$
Output: $M-1$ templates $(\mathbf{a}_i, b_i, \mathbf{X}_{\mathcal{S}i} \subset \widetilde{\mathbf{X}}, A_i, B_i)$
for $i = 1$ to $M-1$ do
    for $j = 1$ to $N$ do
        if $\tilde{\mathbf{x}}_j$ belongs to $\Omega_l$ with $l \leq i$ then $c_j \leftarrow 1$ else $c_j \leftarrow -1$
    end for
    $\mathbf{a}_i, b_i, \mathbf{X}_{\mathcal{S}i} \subset \widetilde{\mathbf{X}} \leftarrow$ SMO-training [6] with $\widetilde{\mathbf{X}}$, $(c_1, \ldots, c_N)$, and $\gamma$
    $A_i, B_i \leftarrow$ Sigmoid-training [13] with $\mathbf{a}_i, b_i, \mathbf{X}_{\mathcal{S}i} \subset \widetilde{\mathbf{X}}$, and $\widetilde{\mathbf{X}}$
end for
```

Algorithm 2. SVM Template Classification

```
Input: $M-1$ templates $(\mathbf{a}_i, b_i, \mathbf{X}_{\mathcal{S}i} \subset \widetilde{\mathbf{X}}, A_i, B_i)$, and new trace $\mathbf{x}$
Output: Posterior probabilities $p(\Omega_1|\mathbf{x}), \ldots, p(\Omega_M|\mathbf{x})$
for $i = 2$ to $M-1$ do
    $p(\Omega_i|\mathbf{x})$ acc. to (12) with $(\mathbf{a}_j, b_j, \mathbf{X}_{\mathcal{S}j} \subset \widetilde{\mathbf{X}}, A_j, B_j)$ for $j \in \{i-1, i\}$
end for
$p(\Omega_1|\mathbf{x})$ acc. to (13) with $(\mathbf{a}_1, b_1, \mathbf{X}_{\mathcal{S}1} \subset \widetilde{\mathbf{X}}, A_1, B_1)$
$p(\Omega_M|\mathbf{x})$ acc. to (14) with $(\mathbf{a}_{M-1}, b_{M-1}, \mathbf{X}_{\mathcal{S}M-1} \subset \widetilde{\mathbf{X}}, A_{M-1}, B_{M-1})$
```

Defensive Leakage Camouflage

Eric Brier[1], Fortier Quentin[2], Roman Korkikian[3,4], K.W. Magld[5],
David Naccache[2,4], Guilherme Ozari de Almeida[3,4], Adrien Pommellet[2],
A.H. Ragab[5], and Jean Vuillemin[2,5]

[1] Ingenico
1, rue Claude Chappe, BP 346, F-07503 Guilherand-Granges, France
[2] École normale supérieure, Département d'informatique
45, rue d'Ulm, F-75230, Paris Cedex 05, France
[3] Altis Semiconductor
224 Boulevard John Kennedy, F-91100, Corbeil-Essonnes, France
[4] Sorbonne Universités – Université Paris II
12 place du Panthéon, F-75231, Paris Cedex 05, France
[5] King Abdulaziz University, Jeddah, Saudi Arabia

Abstract. This paper considers the transfer of digital data over *leaky and noisy* communication channels. We develop defensive strategies exploiting the fact that noise prevents the attacker from accurately measuring leakage.

The defense strategy described in this paper pairs each useful data element k with a camouflage value v and simultaneously transmits both k and v over the channel. This releases an emission $e(k,v)$. We wish to select the camouflage values $v(k)$ as a function of k in a way that makes the quantities $e(k,v(k))$ as *indistinguishable* as possible from each other.

We model the problem and show that optimal camouflage values can be computed from side-channels under very weak physical assumptions. The proposed technique is hence applicable to a wide range of readily available technologies.

We propose algorithms for computing optimal camouflage values when the number of samples per trace is moderate (typically ≤ 6) and justify our models by a statistical analysis.

We also provide experimental results obtained using FPGAs.

1 Introduction

In addition to its usual complexity postulates, cryptography silently assumes that secrets can be physically protected in tamper-proof locations. All cryptographic operations are physical processes where data elements must be represented by physical quantities in physical structures.

At any given point in the evolution of a technology, the smallest logic devices must have a definite physical extent, require a certain minimum time to perform their function and dissipate a minimal switching energy when transiting from one state to another. Energy is also dissipated statically, *i.e.* in the absence of any switching.

S. Mangard (Ed.): CARDIS 2012, LNCS 7771, pp. 277–295, 2013.

During the last twenty years, the research community devised many sophisticated methods for retrieving secret information from circuits by measuring their side-channel emanations.

A number of authors, *e.g.* [1], rely on the isotropic switching-model in which all bits dissipate identical switching energies. This work does not assume any *a priori* side-channel model and totally relies on the analysis of actually measured[1] emissions.

While most previous works analyzed leakage from complex cryptographic computations, we focus on one of the simplest forms of leakage: the emanations of a bus through which bits are being sent. We make only two physical assumptions:

- Emanations can be measured with equal (in)accuracy by both the attacker and the defender.
- Leakage is a global function of data plus noise. The proposed methods are thus unadapted to settings in which individual channel bits are probed with precision.

The proposed methodology is hence applicable to a wide range of circuits having *leaky* buses.

The proposed countermeasure pairs each useful data element k with a camouflage value v and simultaneously transmits k and v through the channel. This releases a physical side-channel emanation $e(k, v)$ that can be measured by both the attacker and the defender.

We address the following question:

How can a defender pair each value of k with a corresponding value $v(k)$ that makes the $e(k, v(k))$ as *indistinguishable* as possible from each other?

The crux of this paper is the definition of *indistinguishability* given the measured emissions.

Section 2 introduces algorithms for computing optimal camouflage values from actual power traces. These algorithms are efficient when each trace contains a few samples (typically ≤ 6). Section 3 presents a statistical analysis justifying the intuition that the best v values are those concentrating the $e(k, v)$ into the smallest possible sphere containing representatives of all k values. Section 4 provides experimental results.

In a way, this work achieves some sort of cryptographic key exchange based on the existence of ambient noise and on a gap in measurement accuracy between the legitimate receiver and the attacker.

2 Models and Algorithms

Let $e(d)$ represent the side-channel (*e.g.* power consumption) resulting from the transfer of an n-bit data element d over an n-bit channel (*e.g.* a bus). $e(d)$ can be measured with equal precision by both the attacker and the defender.

[1] Potentially anisotropic.

The defender builds a set of 2^n side-channel measurements $\mathcal{E}$. Each $e(d) \in \mathcal{E}$ is generated by transmitting an n-bit data element d. The defender assigns s channel bits to the useful information k, and devotes the remaining $n-s$ bits to the transmission of $(n-s)$-bit camouflage values $v(k)$. We denote $d = k|v$ and call the k's *"keys"* or *"colors"*. Note that key bits and camouflage bits are not necessarily adjacent and might be interleaved.

Let $e(k, v) = e(d)$ be the emanation released by transmitting $d = k|v$.

The vector $V = [v(0), \ldots, v(2^s - 1)]$ of all camouflage values must be chosen to make all emanations $e(k, v(k))$ look "as similar as possible". Our goal is to infer V from $\mathcal{E}$.

We assign a unique *color* $k = \text{color}(e(k|v))$ to each $e(i) \in \mathcal{E}$. $\mathcal{E}$ is hence analogous to a multidimensional cloud of 2^n colored points (*i.e.* 2^s sets of colored points; each of these 2^s sets contains 2^{n-s} identically colored points).

A *color-spanning sphere* is a subset $\mathcal{B} \subset \mathcal{E}$ containing at least one emission of each color.

The defender will use the 2^n elements of $\mathcal{E}$ to select 2^s transmittable $k|v(k)$ values forming a color-spanning sphere $\mathcal{A}(V) \subset \mathcal{E}$. The attacker will only get to see traces belonging to $\mathcal{A}(V)$:

$$\mathcal{A}(V) = \bigcup_{k=1,\ldots,2^s-1} \{e \in \mathcal{E} : \text{color}(e) = k\}$$

The defender's goal is to minimize the *size* of the color-spanning sphere $\mathcal{A}(V)$ exposed to the attacker. *i.e.* infer from $\mathcal{E}$ a *smallest color-spanning sphere* $\mathcal{A}_{\text{optimal}}$ such that

$$\|\mathcal{A}_{\text{optimal}}\| = \min_V \|\mathcal{A}(V)\|$$

$\mathcal{A}_{\text{optimal}}$ has thus the least size for all choices of V.

The next section considers the simplest setting where emanations are scalars[2]. In that case the difference $|e - e'|$ between two scalars $e, e' \in \mathcal{E}$ can be used as a similarity measure for constructing $\mathcal{A}_{\text{optimal}}$ efficiently.

2.1 One Dimension

Assume that the $e(d)$ are scalars (*e.g.* execution times or a unique power measurement per trace). Acquire the 2^n reference traces:

$$\mathcal{E} = \{e(0), \ldots, e(2^n - 1)\}$$

A given choice of $V = [v(0), \ldots, v(2^s - 1)]$ restricts the attacker's information to

$$\mathcal{A}(V) = \{e(0, v(0)), \ldots, e(2^{s-1}, v(2^{s-1}))\}$$

[2] *e.g.* execution times or a unique power measurement per trace.

The defender's goal is to minimize:

$$\|\mathcal{A}(V)\| = \max \mathcal{A}(V) - \min \mathcal{A}(V) = \max_k(e(k, v(k))) - \min_k(e(k, v(k)))$$

Let $\mathcal{P} = [p_0 \leq p_1 \leq \cdots \leq p_{2^n-1}]$ be the $e(i) \in \mathcal{E}$ sorted (with repetitions) by increasing scalar values. A *color-spanning segment* is an interval of $\mathcal{P}$ containing at least one p_i of each color.

A straightforward algorithm for finding $\mathcal{A}_{\text{optimal}}$ consists in working with two pointers start and end representing the beginning and the end of the segment under evaluation. When execution begins, start and end point at p_0. While [start,end] is not a color-spanning segment end is moved to the right. When end reaches p_{2^n-1} start is moved by one position to the right (*i.e.* from p_i to p_{i+1}) and end is moved back to start. Throughout this process, whenever a shorter color-spanning segment is found, it is recorded. The complexity of this algorithm is quadratic in the cardinality of $\mathcal{E}$, *i.e.* $O(2^{2n})$.

More clever approaches allow to solve the problem in $\tilde{O}(2^n)$. To do so build the 2^s sorted sequences (with repetitions) of emissions for each color:

$$\mathcal{P}^k = [p_0^k \leq \ldots \leq p_{2^{n-s}-1}^k] \text{ for } k = 1, \ldots, 2^s - 1$$

Represent the *color-spanning* segments by a binary search tree $\mathcal{T}$ of size 2^s.

At step 0, initialize the tree to $\mathcal{T}_0 = \{p_0^0, \ldots, p_0^{2^s-1}\}$ and proceed by 2^s-way merging.

At stage t, the color-spanning tree is

$$\mathcal{T}_t = \left\{ p_{\lambda_t^0}^0, \ldots, p_{\lambda_t^{2^s-1}}^{2^s-1} \right\}$$

where the λ_t^k denote the merge pointers.

Let $\underline{m}$ and $\overline{m}$ denote (respectively) the minimal and maximal scalars in $\mathcal{T}_t$. We denote by ϕ_t the minimal (*i.e.* best) segment length found at step t.

If $t = 0$ or $\overline{m} - \underline{m} < \phi_{t-1}$, then update $\phi_t = \overline{m} - \underline{m}$ else $\phi_t = \phi_{t-1}$.

Let $\underline{m} = p_{\lambda_t^c}^c$ and let $m = p_{\lambda_t^c+1}^c$ be the next emission of the same color. The next tree $\mathcal{T}_{t+1}$ is obtained by replacing $\underline{m}$ by m in $\mathcal{T}_t$. *i.e.* we increase $\lambda_{t+1}^c = \lambda_t^c + 1$ and stall all other merge pointers $\lambda_{t+1}^k = \lambda_t^k$ for $k \neq c$.

The algorithm terminates (at some step $\tau < 2^n$) when it fails to find a successor m to $\underline{m}$. The length of the minimal color-spanning segment is then ϕ_τ.

Complexity: Partitioning $\mathcal{E}$ to 2^s color subsets and sorting these subsets to get the $\mathcal{P}^k$ costs $O(n2^n)$.

Binary search trees [5] support the operations (insert, find-min, extract-min and find-max) required by the structure $\mathcal{T}$, each of these operations requires $O(s)$ time. It follows that the 2^s-way merge runs in $O(s2^n)$ and hence the above algorithm has an overall complexity of $\tilde{O}(2^n)$.

2.2 Higher Dimensions

We now consider the general case where e is a T-dimensional vector, *e.g.* a power consumption sampled at T different instants. $\mathcal{E}$ is now a T-dimensional cloud of colored points (Fig. 1) and the color spanning interval is a T-dimensional sphere. We need to determine the smallest sphere containing at least one point of each color *i.e.* the smallest color-spanning sphere $\mathcal{A}_{\text{optimal}}$ (Fig. 2, right).

The cloud of points is contained in some minimal enclosing T-dimensional rectangle $\mathcal{R}$, whose sides are parallel to the hyperspace's T axes (Fig. 3, right).

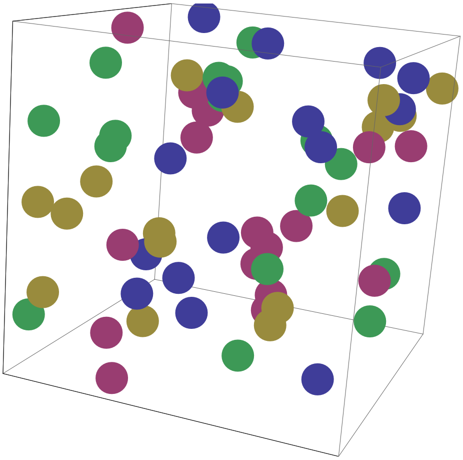


Fig. 1. Power trace representation in 3 dimensions

Divide and Conquer. This problem lends itself to divide and conquer resolution.

Let $\mathcal{B}$ be some[3] initial color spanning sphere of radius r. Let ℓ denote the length of the rectangle $\mathcal{R}$ along some dimension x. Split $\mathcal{R}$ along the x axis into two overlapping sub-rectangles of lengths $\frac{\ell}{2}+r$ as shown by Figure 4. Let $\mathcal{R}_{\text{right}}$ and $\mathcal{R}_{\text{left}}$ be the two equally sized sub-rectangles obtained that way (Fig. 5).

By construction, $\mathcal{A}_{\text{optimal}}$ is fully contained in either $\mathcal{R}_{\text{right}}$ or $\mathcal{R}_{\text{left}}$. So, we recursively apply the process to $\mathcal{R}_{\text{right}}$ and $\mathcal{R}_{\text{left}}$ until splitting diminishes the rectangles' sizes only negligibly[4]. At that point we solve each of the smaller instances (by any chosen method) and output the smallest solution of all, which is indeed the smallest color-spanning sphere in $\mathcal{R}$ *i.e.* the smallest color-spanning sphere $\mathcal{A}_{\text{optimal}}$ of the original problem.

[3] Not necessarily optimal, *cf.* Fig. 3, left.

[4] After the w-th splitting the rectangles' sides are of size $\ell_w = (\ell - 2r)2^{-w} + 2r$. Hence splitting can last forever. We suggest to stop splitting when $\ell_w < 3r$ *i.e.* after $\lfloor \log_2(\ell/r - 2) \rfloor$ iterations.

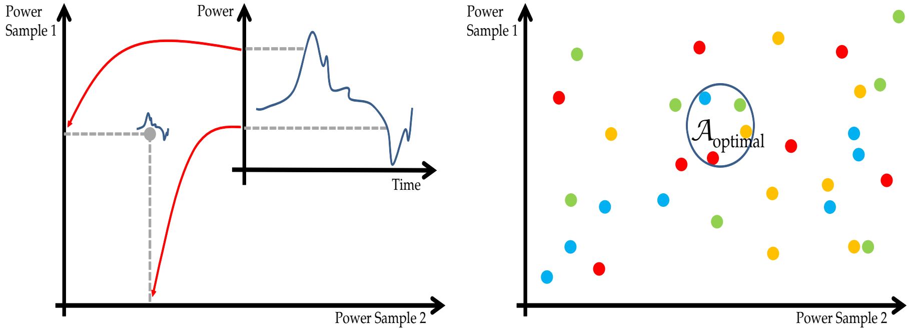


Fig. 2. Left: Mapping curves into points. Right: Problem instance and its optimal solution $\mathcal{A}_{\text{optimal}}$.

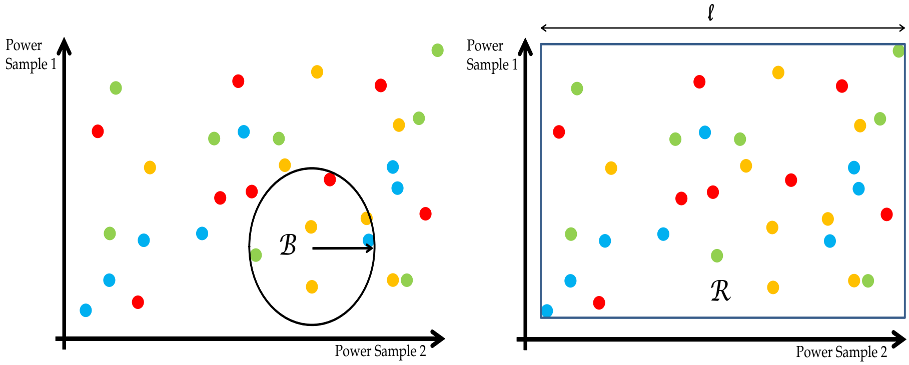


Fig. 3. Left: Step 1, find any color spanning sphere $\mathcal{B}$. Right: Step 2, define the rectangle $\mathcal{R}$.

Note that splitting can take place along several orthogonal axes simultaneously.

While practically very useful, this algorithm fails in a number of pathological cases (*e.g.* when $\mathcal{B}$ is too large to split $\mathcal{R}$). Luckily this is a well-studied problem: [2] describes a simple linear-time algorithm in two dimensions and Welzl [3] shows how to solve the problem in linear time for all dimensions, considering that the number of dimensions is a fixed problem parameter. Complexity is however exponential in the number of dimensions.

A key choice is the initial sphere $\mathcal{B}$: we want $\mathcal{B}$ to be small enough to significantly reduce the divide and conquer's search space. Yet, we want $\mathcal{B}$ to remain easy to compute.

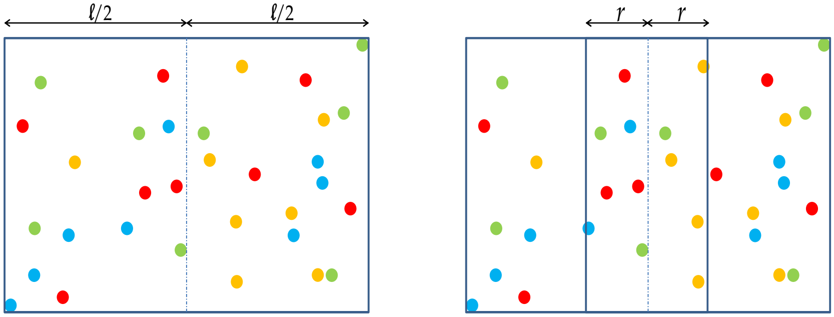


Fig. 4. Left: Step 3, split $\mathcal{R}$ into two overlapping rectangles $\mathcal{R}_{\text{right}}$ and R_{left} of length $\frac{\ell}{2} + r$

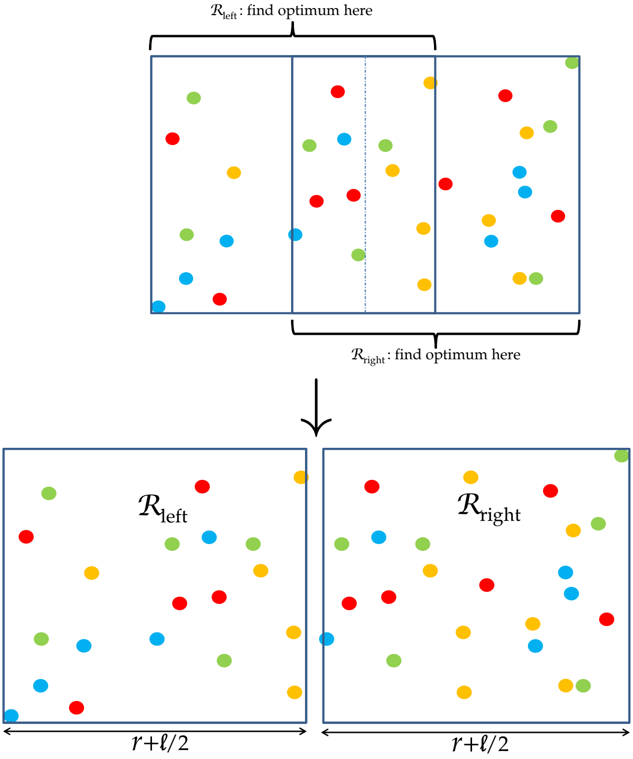

$\mathcal{R}_{\text{left}}$: find optimum here
$\mathcal{R}_{\text{right}}$: find optimum here
$\mathcal{R}_{\text{left}}$
$\mathcal{R}_{\text{right}}$
r+ℓ/2
r+ℓ/2

Fig. 5. Recursive problem size reduction

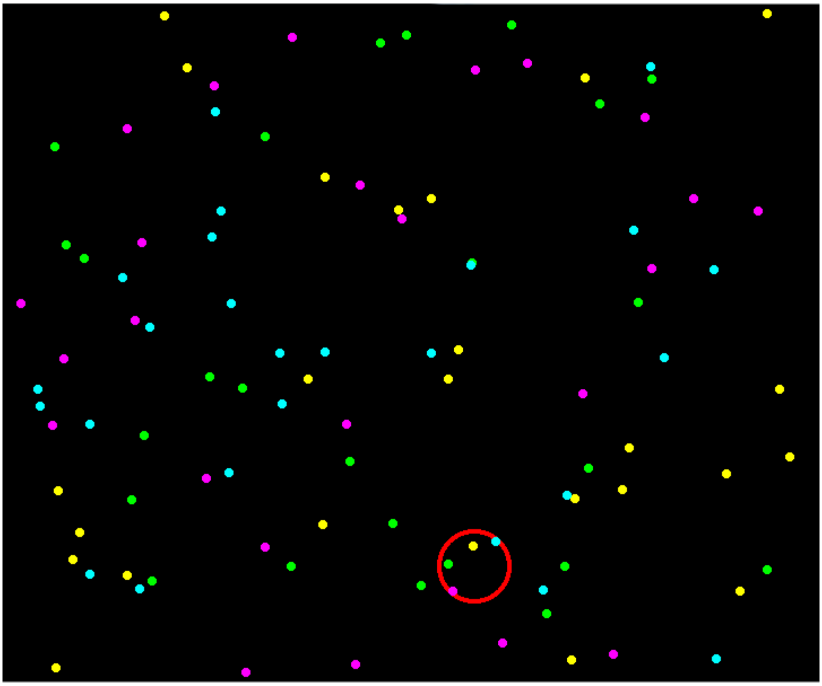


Fig. 6. Program output example in 2 dimensions

Heuristics: In our implementation we used the following method to construct $\mathcal{B}$: let p_0 be a point (for example the closest point to the center of $\mathcal{R}$) of color 0. After computing $p_1, \ldots, p_k$, we select as p_{k+1} the point of color $k+1$ at minimal distance from the barycenter of the cloud $p_1 \cdots p_k$. The resulting $\mathcal{B}$ is not necessarily optimal, (*cf.* Figure 7) but turns out to be much better than selecting any random color-spanning sphere.

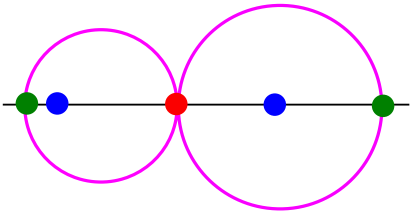


Fig. 7. The optimal sphere (left) is different from the sphere found by the barycenter heuristic (right) if the heuristic considers first the red, then the blue and finally the green points

2.3 Implementations

Algorithms were implemented in C++[5] in a straightforward manner. A function

```
bool smallest_ball(points, space, output)
```

splits space and points as explained above (using a sphere found by `find_ball_barycenter`) and calls recursively `smallest_ball` on the smaller spaces, until this process stops to significantly decrease the problem size. We then use Miniball[6], a C++ software for computing smallest enclosing spheres of points in arbitrary dimensions (without requiring spheres to be color spanning) using brute force. The description of Miniball can be found in [4,3].

Timings were measured on a Dell Inspiron 1520[7]. Code was compiled using Visual C++ 2008 with all optimization flags set for maximal speed.

Table 1. Running time for points randomly chosen in the 3-dimensional unit cube, averaged over 10 runs

Total number of points	2 colors	3 colors	4 colors	5 colors
10^2	8 ms	11 ms	43 ms	211 ms
10^3	96 ms	221 ms	833 ms	7 s
10^4	946 ms	3 s	11 s	81 s
10^5	10 s	31 s	145 s	953 s
10^6	109 s	327 s		

Table 2. Running time for points randomly chosen in the 4-dimensional unit cube, averaged over 10 runs

Total number of points	2 colors	3 colors	4 colors	5 colors
10^2	11 ms	39 ms	309 ms	2 ms
10^3	164 ms	1 s	10 s	147 s
10^4	2 s	16 s	160 s	
10^5	27 s	188 s	37 min	
10^6	287 s	32 min	> 1 hour	> 1 hour

Experimental running times seem to confirm that the algorithm is linear in the number of points and exponential in the number of colors.

[5] The code is available at `http://perso.ens-lyon.fr/quentin.fortier/color_ball.html`

[6] `http://www.inf.ethz.ch/personal/gaertner/miniball.html`

[7] Intel Core 2 Duo T7300 processor, 2.0GHz, 4MB L2 cache, 2Go memory.

3 Why Euclidean Distances?

Let $\{m_{0,t},\ldots,m_{n-1,t}\}$ be a database of n reference power consumption traces measured over some discrete time interval $t \in [0;T-1]$. Sample $m_{i,t}$ corresponds to the power consumption caused by the manipulation of data element i at instant t. Let μ_t be the average power consumption at time t and σ_t the standard deviation at time t:

$$\mu_t = \frac{1}{n}\sum_{i<n} m_{i,t} \qquad\qquad \sigma_t = \sqrt{\frac{1}{n}\sum_{i<n}(m_{i,t}-\mu_t)^2}.$$

Let a_t be an unidentified power measurement made by an attacker. The attacker's problem consists in finding the $m_{k,t}$ that *best reassembles* a_t. This section justifies why for doing so, an attacker would naturally compute for $i < n$ the quantities:

$$\text{score}(i) = \sum_{t<T}\frac{(a_t - m_{i,t})^2}{\sigma_t^2}, \tag{1}$$

and output the guess k corresponding to the $m_{k,t}$ whose score is the lowest *i.e.*:

$$\text{score}(k) = \min_{i<n}(\text{score}(i)).$$

This formula is justified in the next section for t-wise independent $m_{i,t}$'s.

In general, samples may be correlated, for instance when the same secret bit is manipulated at two different instants. We analyze this general case later and propose an explicit score minimization formula (2) taking into account intra-sample correlations.

3.1 Multivariate Normal Distributions

Equation (1) stems from the assumption that, for any fixed i, successive measurements of $m_{i,t}$ follow an independent normal distribution with mean μ_t and standard deviation σ_t, and hence abide by the probability density function:

$$f_{m_t}(x) = \frac{1}{\sigma_t\sqrt{2\pi}}\exp\left(-\frac{(x-\mu_t)^2}{2\sigma_t^2}\right)$$

When the $m_{i,t}$'s are independent, the probability density of all measurements $t < T$ can be expressed, for $\boldsymbol{x} = [x_0 \cdots x_{T-1}]$ as a T-dimensional multivariate distribution:

$$f_{\boldsymbol{m}}(\boldsymbol{x}) = \prod_{t<T} f_{m_t}(x_t) = \frac{1}{(2\pi)^{T/2}\prod\limits_{t<T}\sigma_t}\exp\Big(-\sum_{t<T}\frac{(x_t-\mu_t)^2}{2\sigma_t^2}\Big).$$

Note that in the previous equation μ_t and σ_t are the expected value and standard deviation of $m_{i,t}$ over *all* data elements i. For a measurement $m_{i,t}$ corresponding to a specific data element i, in addition, we also assume that $m_{i,t}$ follows a normal distribution with mean $\tilde{\mu_t} = m_{i,t}$ and standard deviation $\tilde{\sigma_t}$; we also assume that the standard deviation $\tilde{\sigma}_t$ around $m_{i,t}$ is the same for all data elements. In this case, the measurement m_t corresponding to data element i has the following distribution:

$$f_{\boldsymbol{m}}(\boldsymbol{x}) = \frac{1}{(2\pi)^{T/2} \prod\limits_{t<T} \tilde{\sigma}_t} \exp\Big(- \sum_{t<T} \frac{(x_t - m_{i,t})^2}{2\tilde{\sigma}_t^2} \Big)$$

Additionally, we assume that the standard deviation $\tilde{\sigma}_t$ of m_t around $m_{i,t}$ is proportional to the standard deviation σ_t of m_t when all data values are considered, *i.e.* we assume $\tilde{\sigma}_t = \alpha \cdot \sigma_t$ for all $0 \leq t \leq T-1$ for some $\alpha \in \mathbb{R}$. In this case, the probability density function of the m_t's for data i can be written as:

$$f_i(\boldsymbol{m}) = \frac{1}{(2\pi)^{T/2} \alpha^T \prod\limits_{t<T} \sigma_t} \exp\Big(- \sum_{t<T} \frac{(m_t - m_{i,t})^2}{2\alpha^2 \sigma_t^2} \Big) \; \propto \; \exp\Big(- \frac{\text{score}(i)}{2\alpha^2} \Big)$$

where score(i) is given by equation (1). The probability to obtain measurements m_t from data i is thus a decreasing function of score(i). Given measurement $\boldsymbol{m}$, the most probable candidate is therefore the one with the lowest score.

3.2 Multivariate Normal Distribution: Taking Correlation into Account

We denote by Σ the covariance matrix of the measurements, defined as follows:

$$\Sigma = \text{var}(\boldsymbol{m}) = \text{var} \begin{pmatrix} m_1 \\ \vdots \\ m_T \end{pmatrix} = \begin{pmatrix} \text{var}(m_1) & \text{cov}(m_1 m_2) \cdots & \text{cov}(m_1 m_T) \\ \text{cov}(m_1 m_2) & \ddots \quad \cdots & \vdots \\ \vdots & \vdots \quad \ddots & \vdots \\ \text{cov}(m_1 m_T) & \cdots \quad \cdots & \text{var}(m_T) \end{pmatrix}$$

where $\text{cov}(X, Y) = \text{E}(XY) - \text{E}(X)\text{E}(Y)$ and $\text{var}(X) = \text{cov}(X, X) = \text{E}(X^2) - \text{E}(X)^2$.

We assume that the measurements follow a T-dimensional multivariate distribution with mean $\boldsymbol{\mu}$ and covariance matrix Σ. The probability density function can then be expressed as:

$$f_{\boldsymbol{m}}(\boldsymbol{x}) = \frac{1}{(2\pi)^{T/2} |\Sigma|^{1/2}} \exp\Big(-\tfrac{1}{2} (\boldsymbol{x} - \boldsymbol{\mu})^{\text{tr}} \Sigma^{-1} (\boldsymbol{x} - \boldsymbol{\mu}) \Big).$$

where $|\Sigma|$ is the determinant of Σ and M^{tr} is the transposed of matrix M. The mean $\boldsymbol{\mu}$ is a T-vector and Σ is a $T \times T$-matrix.

Note that in the previous equation $\boldsymbol{\mu}$ and Σ are the expected value and covariance matrix of measurements for *all* data elements i. As previously for measurements corresponding to a specific data element i, we assume that these measurements follow a T-multivariate normal distribution with mean $\tilde{\mu_t} = m_{i,t}$ and covariance matrix $\tilde{\Sigma}$.

If we further assume that matrix $\tilde{\Sigma}$ is identical for all data elements, the measurement $\boldsymbol{m}$ for data i then obeys the multivariate distribution:

$$f_{\boldsymbol{m}}(\boldsymbol{x}) = \frac{1}{(2\pi)^{T/2}|\tilde{\Sigma}|^{1/2}} \exp\Big(- \tfrac{1}{2}(\boldsymbol{x} - \boldsymbol{m_{i,\cdot}})^{\mathrm{tr}}\tilde{\Sigma}^{-1}(\boldsymbol{x} - \boldsymbol{m_{i,\cdot}})\Big).$$

As previously, let us additionally assume that the covariance matrix satisfies $\tilde{\Sigma} = \alpha \cdot \Sigma$ for some $\alpha \in \mathbb{R}$. In this case, the probability density function is expressed by:

$$f_{\boldsymbol{m}}(\boldsymbol{x}) = \frac{1}{(2\pi\alpha)^{T/2}|\Sigma|^{1/2}} \exp\Big(- \tfrac{1}{2\alpha}(\boldsymbol{x} - \boldsymbol{m_{i,\cdot}})^{\mathrm{tr}}\Sigma^{-1}(\boldsymbol{x} - \boldsymbol{m_{i,\cdot}})\Big).$$

This can finally be written as

$$f_{\boldsymbol{m}}(x) = \frac{1}{(2\pi\alpha)^{T/2}|\Sigma|^{1/2}} \exp\Big(- \frac{\mathrm{score}(i)}{2\alpha}\Big)$$

where

$$\mathrm{score}(i) = (\boldsymbol{m} - \boldsymbol{m_{i,\cdot}})^{\mathrm{tr}}\Sigma^{-1}(\boldsymbol{m} - \boldsymbol{m_{i,\cdot}}) \tag{2}$$

It follows that equation (2) is a generalization of equation (1) where correlations are taken into account. In other words, to take correlations into account acquire a_t and compute for every i the score as per equation (2), sort the scores by increasing values and bet on the smallest.

Example. To illustrate the procedure, we consider the bivariate case where the covariance matrix between variables X and Y is:

$$\Sigma = \begin{bmatrix} \sigma_x^2 & \rho\sigma_x\sigma_y \\ \rho\sigma_x\sigma_y & \sigma_y^2 \end{bmatrix}$$

where $\mathrm{var}(X) = \sigma_x^2$, $\mathrm{var}(Y) = \sigma_y^2$, $\mathrm{cov}(X, Y) = \rho\sigma_x\sigma_y$ and ρ is the correlation between X and Y. In this case, we find:

$$\Sigma^{-1} = \frac{1}{1-\rho^2}\begin{bmatrix} \frac{1}{\sigma_x^2} & \frac{-\rho}{\sigma_x\sigma_y} \\ \frac{-\rho}{\sigma_x\sigma_y} & \frac{1}{\sigma_y^2} \end{bmatrix}$$

and the probability density function can be written as

$$f(x, y) = \frac{1}{2\pi\sigma_x\sigma_y\sqrt{1-\rho^2}} \exp\left(-\frac{1}{2(1-\rho^2)}\left[\frac{x^2}{\sigma_x^2} + \frac{y^2}{\sigma_y^2} - \frac{2\rho xy}{\sigma_x\sigma_y}\right]\right).$$

In this case, equation (2) gets simplified as follows:

$$s_i = \frac{(a_1 - m_{i,1})^2}{\sigma_1^2} + \frac{(a_2 - m_{i,2})^2}{\sigma_2^2} - \frac{2\rho(a_1 - m_{i,1})(a_2 - m_{i,2})}{\sigma_1\sigma_2}$$

where $\sigma_1 = \mathrm{var}(m_1)$, $\sigma_2 = \mathrm{var}(m_2)$ and ρ is the correlation between m_1 and m_2.

4 Experiments

4.1 Measurements

This section describes our experimental results using the Altera EP2C20F484C7N FPGA present on the Cyclone II Starter Development Kit (SDK). Fig.8 shows the circuit used to measure the power consumption of a memory read + register store operation. The circuit consisted of a RAM, a multiplexer, eight registers, slide switches (DIP) and buttons. Identical data was simultaneously written into eight identical registers to increase power signature.

Power was measured using a 1GHz oscilloscope (TDS 684B) and a Tektronix P6247 differential probe (1GHz bandwidth). The SDK's two GPIO pins (power and ground) were connected via the differential probe. Apart from DC signal rejection no filtering or power trace post processing was done.

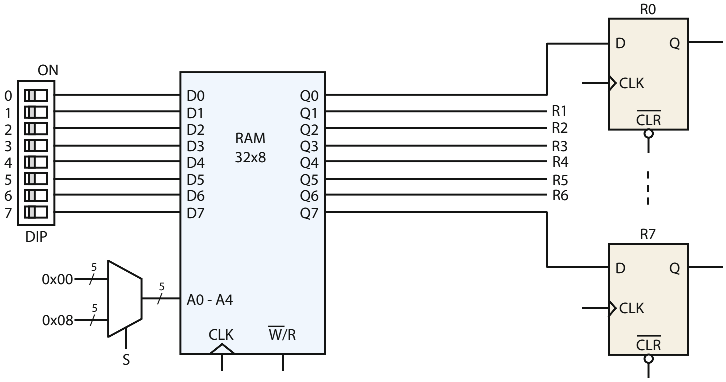


Fig. 8. The experimental circuit used for power consumption measurements

The experimental protocol was defined as follows:

- The DIP's eight slide switches were manually set to 0x00.
- Address 0x00 was latched on address bus A=[A0,...,A4] using the multiplexer's control bit S. This caused the value 0x00 to be written into RAM address 0x00.
- For $d = 0$ to 255:
  - The DIP's eight slide switches were manually set to d.
  - Pressing the board's KEY0 button triggered the following sequence of events 1000 times (averaged to remove noise):
    1. RAM write ($\overline{\texttt{W}}$) was activated and bit S was used to latch address 0x08 on bus A. This caused d to be written to RAM address 0x08 (1 cycle).
    2. RAM read was activated (R) and bit S was used to latch address 0x00 on bus A. This caused 0x00 to be read-out of RAM and clear all data previously present on the bus and in the registers (3 cycles).

3. The RAM's CLK signal was disabled.
4. Bit S was used to latch address 0x08 on bus A.
5. The oscilloscope was triggered.
6. The RAM's CLK signal was enabled for one cycle only causing d to appear on bus [R0,...,R7]. The RAM's CLK signal was immediately re-disabled to avoid a double-reads and freeze d on bus [R0,...,R7].
7. At the next clock cycle, d appeared at the Q output pins of the eight registers.
8. The clock was left running for one more cycle to acquire any signal tails due to capacitive discharges.

- A 2500-sample averaged power measurement $e'(d)$ was recorded.
- Three samples corresponding to instants t_0, t_1, t_2 were extracted from $e'(d)$ to form $e(d)$. $e(d)$ was recorded[8] as a file trace_d.d used for camouflage calculations.

The described state-flow could only be interrupted by power-off or by pressing KEY0. A finite state-machine (FSM) diagram will appear in the final version of this paper. A characteristic power trace is shown in Figure 9.

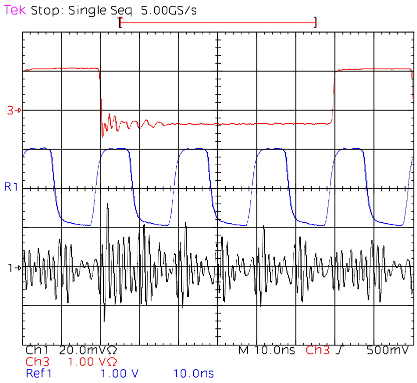


Fig. 9. Power trace of the circuit on Fig.8

The obtained results confirm very wall both our analysis and intuition. However, for various technical reasons, we are not entirely satisfied with this first measurement campaign. We thus plan to refine our setting and provide new experimental results in the final paper.

4.2 Analysis

Figure 12 represent the 256 values ($n = 8$) obtained experimentally as 8 color families (*i.e.* 32 points per family). The experimental data is available upon request.

[8] 3 big-endian values stored in ASCII in decimal format. Each sample is represented by two bytes (oscilloscope's precision).

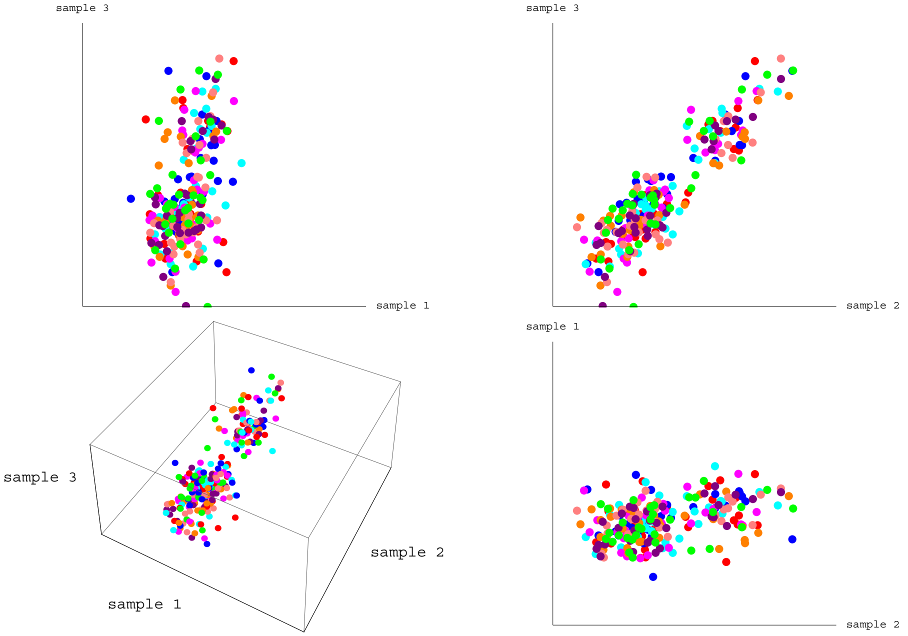


Fig. 10. Experimental results for $n = 8$. 3D and projected representations of the 256 experimental measurements (represented as 8 color families of 32 points).

Our goal is to consider this data as 2^i colors × 2^{8-i} points for $i = 1, \ldots, 7$, select the optimal bus bits on which k should be encoded, compute the $v(k)$ in all cases and check if the results indicate, as we conjecture, that similar Hamming weight words yield the best encoding.

For two colors (*i.e.* a 1-bit k) the two most similar bus values are `0x7F` and `0xF9` for which:

$$\text{distance}(e(\texttt{0x7F}), e(\texttt{0xF9})) = \text{distance}(\{28601, 28795, 28794\}, \{29115, 28789, 28876\}) = 26.94$$

For four colors (*i.e.* a 2-bit k) we get:

binary value of k	**optimal** k **and** $v(k)$	**observed side channel** $e(k, v(k))$
00	0xB4=101101$\underline{00}$	$e(\texttt{0xB4}) = \{28704, 28232, 28278\}$
01	0xD9=110110$\underline{01}$	$e(\texttt{0xD9}) = \{28652, 28107, 28315\}$
10	0x96=100101$\underline{10}$	$e(\texttt{0x96}) = \{28716, 28159, 28293\}$
11	0x6B=011010$\underline{11}$	$e(\texttt{0x6B}) = \{28670, 28280, 28380\}$

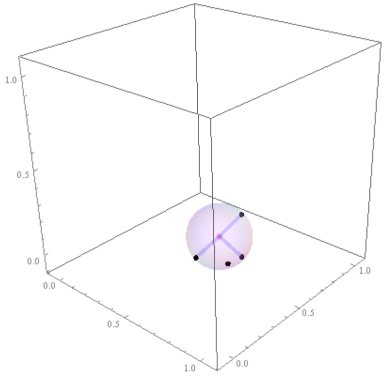


Fig. 11. Display of the re-scaled optimal solution rescale(e(0xB4)), rescale(e(0xD9)), rescale(e(0x96)), rescale(e(0x6B))

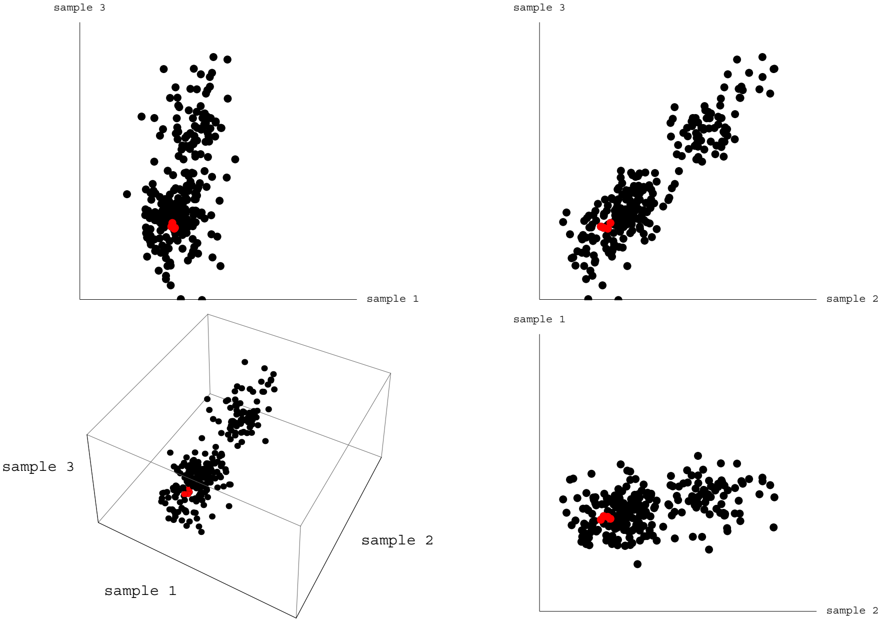


Fig. 12. Experimental results for $n = 8$. Position of the optimal solution e(0xB4), e(0xD9), e(0x96), e(0x6B).

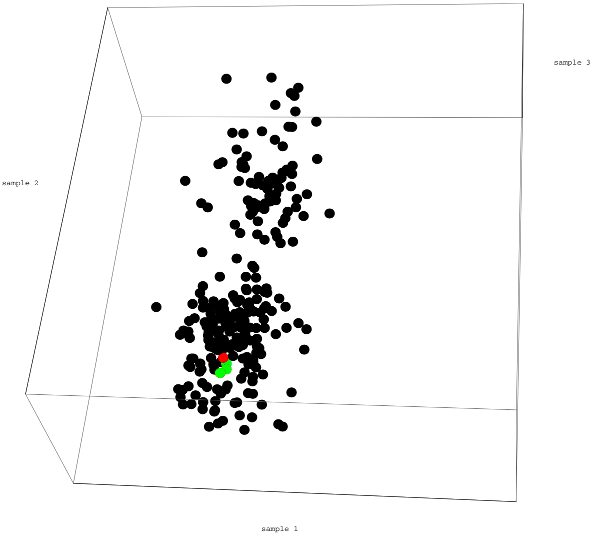


Fig. 13. Experimental results for $n = 8$. Position of the optimal solution $e(\texttt{0xB4}), e(\texttt{0xD9}), e(\texttt{0x96}), e(\texttt{0xD0})$.

$e(\texttt{0xB4}), e(\texttt{0xD9}), e(\texttt{0x96}), e(\texttt{0x6B})$ are contained in a sphere of radius $\sqrt{\frac{17239}{2}} \cong 92.84$ centered at $c = \{28661, 28193.5, 28347.5\}$ where:

$$\begin{array}{ll} \text{distance}(c, e(\texttt{0xB4})) = 90.34 & \text{distance}(c, e(\texttt{0xD9})) = 92.84 \\ \text{distance}(c, e(\texttt{0x96})) = 84.77 & \text{distance}(c, e(\texttt{0x6B})) = 92.84 \end{array}$$

The positions are illustrated in Figure 11 where points were re-scaled to $[0, 1]$ using the affine transform $\text{rescale}(\{x, y, z\}) = \{u(x), u(y), u(z)\}$ where $u(\ell) = (\ell - 28107)/609$:

$$\begin{array}{ll} \text{rescale}(e(\texttt{0xB4})) = \{0.98, 0.21, 0.28\} & \text{rescale}(e(\texttt{0xD9})) = \{0.89, 0.00, 0.34\} \\ \text{rescale}(e(\texttt{0x96})) = \{1.00, 0.09, 0.31\} & \text{rescale}(e(\texttt{0x6B})) = \{0.92, 0.28, 0.45\} \end{array}$$

5 Conclusion and Further Research

This works raises a number of interesting questions. A first natural generalization is the translation of our analysis to an infinite number of dimensions (in terms of metrics on function spaces and distances between functions).

A second line of research consists in introducing more complex information encoding schemes. Here the defender detects the 2^s most similar traces in $\mathcal{E} = \{e(0), \ldots, e(2^n - 1)\}$, *e.g.* using clustering. Let $\mathcal{L}$ be the subset (cluster) of these most similar traces:

$$\mathcal{L} = \{e(\ell(1)), \dots, e(\ell(2^s - 1))\} \subset \mathcal{E}$$

The communicating parties assign[9] to the transmitted information the encoding:

$$\ell(k) = \text{encode}(k) \qquad\qquad k = \text{decode}(\ell(k))$$

For four colors (*i.e.* a 2-bit k) using our experimental data, we get:

binary value of k	**optimal encode**(k)	**observed side channel** e(**encode**(k))
`00`	`0x96=10010110`	$e($`0x96`$) = \{28716, 28159, 28293\}$
`01`	`0xB4=10110100`	$e($`0xB4`$) = \{28704, 28232, 28278\}$
`10`	`0xD0=11010000`	$e($`0xD0`$) = \{28703, 28238, 28247\}$
`11`	`0xD9=11011001`	$e($`0xD9`$) = \{28652, 28107, 28315\}$

$e($`0x96`$)$, $e($`0xB4`$)$, $e($`0xD0`$)$, $e($`0xD9`$)$ are contained in a sphere of radius $\sqrt{\frac{12193}{2}} \cong 78.08$ centered at $c = \{28677.5, 28172.5, 28281\}$. This solution (shown in green in Figure 13) shares three points with the previous solution (Figure 12) shown in red.

Along the same line of ideas, a further refinement consists in buying an easier computation of camouflage values at the cost of extra assumptions on the power consumption model. Assume for instance an isotropic consumption model where emanations are proportional to the Hamming weight of the transmitted data. Here all $\binom{n}{w}$ emissions of weight w cause identical emanations. The largest binomial has weight $w = n/2$, and it is bounded by

$$\frac{2^n}{\sqrt{2n}} < b_n = \binom{n}{\frac{n}{2}} < \frac{2^n}{\sqrt{\pi n/2}}.$$

Assigning $c_n = \lceil \log_2 \sqrt{2n} \rceil$ implies that $2^{n-c_n} \leq 2^n/\sqrt{2n} < b_n$, *i.e.* $2^s < b_n$ for $s = n - c_n$. We can thus choose a distinct configuration of weight $n/2$ to encode each secret key k. It follows that $c_n = (3 + \log_2 n)/2$ bits are sufficient to perfectly hide the emanations from $s = n - c_n$ keys over the n bits of an isotropic bus.

If the noise level is high enough then the implementer may use the fact that

$$\log\left(\binom{n}{\frac{n}{2}}\right) \simeq \log\left(\binom{n}{\frac{n}{2} \pm \gamma}\right) \quad \text{for moderate } \gamma \text{ values}$$

and increase bandwidth at the cost of a carefully controlled security risk.

Acknowledgments. Part of this work was supported under grant 12-15-1432-HiCi from King Abdulaziz University.

[9] *e.g.* using a lookup table.

References

1. Brier, E., Clavier, C., Olivier, F.: Optimal Statistical Power Analysis. Cryptology ePrint Archive, Report 152 (2003), `http://eprint.iacr.org/`
2. de Berg, M., van Kreveld, M., Overmars, M., Schwarzkopf, O.: Computational Geometry: Algorithms and Applications. Springer (1997)
3. Welzl, E.: Smallest Enclosing Disks (Balls and Ellipsoids). In: Maurer, H.A. (ed.) New Results and New Trends in Computer Science. LNCS, vol. 555, pp. 359–370. Springer, Heidelberg (1991)
4. Gärtner, B.: Fast and Robust Smallest Enclosing Balls. In: Nešetřil, J. (ed.) ESA 1999. LNCS, vol. 1643, pp. 325–338. Springer, Heidelberg (1999)
5. Knuth, D.: The Art of Computer Programming, 2nd edn. Sorting and Searching, vol. 3. Addison Wesley (1998)

Author Index