KU LEUVEN

Radboud University Nijmegen

Implementation Aspects of Security and Privacy in Embedded Design

Josep Balasch

March 2014

KU LEUVEN

Radboud University Nijmegen

Implementation Aspects of Security and Privacy in Embedded Design

Josep Balasch

March 2014

D/2014/7515/39
ISBN 978-94-6018-815-2

Implementation Aspects of Security and Privacy in Embedded Design

Proefschrift

ter verkrijging van de graad van doctor
aan de Radboud Universiteit Nijmegen
op gezag van de rector magnificus prof. dr. S.C.J.J. Kortmann
volgens besluit van het college van decanen

en ter verkrijging van de graad van doctor in de ingenieurswetenschappen
aan de KU Leuven
op gezag van rector prof. dr. R. Torfs

in het openbaar te verdedigen op dinsdag 11 maart 2014
om 10:30 uur precies

door

Josep M. Balasch Masoliver

geboren op 3 juli 1982
te Vic, Spanje.

Implementation Aspects of Security and Privacy in Embedded Design

Doctoral Thesis

to obtain the degree of doctor
from Radboud University Nijmegen
on the authority of the Rector Magnificus, prof. dr. S.C.J.J. Kortmann,
according to the decision of the Council of Deans

and to obtain the degree of doctor of engineering science
from KU Leuven
on the authority of rector prof. dr. R. Torfs

to be defended in public on Tuesday, 11 March 2014
at 10:30 hours

by

Josep M. Balasch Masoliver

born in Vic, Spain
on 3 July 1982.

Supervisors:

Prof. dr. Ingrid Verbauwhede
KU Leuven, Belgium

Prof. dr. Bart P.F. Jacobs

Doctoral Thesis Committee:

Prof. dr. Lejla Batina

Prof. dr. Claudia Diaz
KU Leuven, Belgium

Dr. Benedikt Gierlichs
KU Leuven, Belgium

Prof. dr. Rudy Lauwereins
KU Leuven, Belgium

Prof. dr. François-Xavier Standaert
Université Catholique de Louvain, Belgium

Prof. dr. Eric R. Verheul

Prof. dr. Yves Willems
KU Leuven, Belgium

To the memory of my father

Acknowledgments

After seemingly endless months of typing and correcting, the time has come to write the last part of this thesis. I will take this opportunity to thank several people who, in one way or another, have greatly influenced the journey leading to this dissertation.

First of all, great appreciation goes to my supervisors Prof. Ingrid Verbauwhede and Prof. Bart Jacobs for giving me the chance to pursue a joint PhD degree. I am grateful for the freedom I have been given during my research, and the guidance and advice provided when needed.

I would like to extend my gratitude to Prof. Rudy Lauwereins, Prof. Lejla Batina and Prof. Claudia Diaz, my assessors during the PhD study, for many valuable advice along the way. My appreciation also to Prof. François-Xavier Standaert, Dr. Benedikt Gierlichs, Prof. Yves Willems and Prof. Eric Verheul for kindly accepting to be members of my Doctoral Examination Committee.

It has been (and still is) a pleasure to be part of a research group like COSIC. I will not attempt to do a one-by-one enumeration of colleagues and friends to whom I am indebted. Instead, I want to thank my multiple office and lab mates, for making working hours more enjoyable; the one o'clock Alma crew, for sharing lunches and stories; the occasional Friday beer gathering, for providing a fun space outside work; and the COSIC futsal team, for that ever present reminder that not everything is about winning. A particular thanks goes to Péla and Elsy for the countless times they have helped me with bureaucratic and financial issues.

Throughout my PhD I had the luck to work together with excellent researchers and to learn a lot from them. I am indebted to many co-authors with whom I share the credit of my works. I am particularly grateful to Carmela, Benedikt, Alfredo and Claudia not only for the close collaboration we had during our days in COSIC, but also for guiding me during the always difficult first stages of research life. My appreciation goes also to Prof. Bart Preneel for allowing me to join the COSIC research group in the first place, before the start of my PhD.

Last but definitely not least, I would like to thank my family, including in-laws, for their encouragement and support during my stay abroad. Also to my childhood friends back in Avinyó, for reminding me that no matter the distance, home will always be home. And finally to Sarah, for constantly being there in good and in bad times and for her unconditional support along the way.

To all of you, *moltíssimes gràcies*!

Josep M. Balasch Masoliver
March 2014

Abstract

Embedded devices are nowadays largely represented across the compute continuum. From mobile phones to smart cards and RFID tags, digital devices are becoming increasingly ubiquitous, mobile and integrated with their environment. This gradual shift towards pervasive computing envisions many benefits in sectors as diverse as financial, entertainment, health care, information access, or automotive. Along with these possibilities however, there are also inherent risks to be addressed. It is in this context that this dissertation is situated. It provides contributions to the security of embedded devices and the privacy of the humans interacting with them.

The first part of the thesis is devoted to physical security. Many existing and future applications have built-in security capabilities which rely on keeping cryptographic keys secret. Typical examples include payment tokens, digital identity documents, or access control cards. As these devices operate in hostile environments, they need protection against physical attacks. Among these, side channel attacks and fault attacks represent two of the major threats in the security of embedded devices.

Our contributions in this area encompass three different but related aspects. First, we provide an in-depth analysis of vulnerabilities that lead to physical attacks. In particular, we characterize the effects of fault injections based on setup-time violations on a low-end microcontroller. Second, we show how physical attacks are still a prominent threat for secure devices by successfully attacking a widely used family of secure memories. And third, we devise and thoroughly evaluate a high-level mitigation against side channel attacks. More specifically, we employ the inner product construction to design a masking-based countermeasure implementable at any order.

The second part of the thesis deals with privacy aspects. Systems such as location-based services, health-care monitoring, or smart homes rely on the collection and processing of fine-grained information about users. Hazards derived from mining, sharing or misusing collected data are numerous, e.g. from discrimination, persecution or reputation damage for end users to large scale surveillance of individuals. Privacy-preserving methods to minimize the processing and/or disclosure of personal data is paramount for the acceptance of these systems.

We select Electronic Toll Pricing (ETP) as case study, a major representative of location-based services. This envisioned system allows governments to levy taxes on the use of public roads by deploying in-vehicle On-Board Units (OBUs). Our main contribution is the design, analysis and implementation of PrETP, a privacy-preserving ETP solution. The privacy guarantees of our system are achieved by letting OBUs compute their road fees locally. At the same time, we provide means for service providers to ensure that OBUs carry out correct computations.

Samenvatting

Van mobiele telefoons tot smartcards en RFID-tags, digitale apparaten zijn alomtegenwoordig, mobiel en geïntegreerd met hun omgeving. Deze geleidelijke verschuiving in de richting van pervasive computing leidt tot vele voordelen in diverse sectoren zoals de financiële, entertainment, gezondheidszorg, informatietoegang, of de automobielsector. Naast deze mogelijkheden zijn er echter ook inherente risico's die gemitigeerd dienen te worden. Deze risico's en hoe ze aangepakt kunnen worden vormen het onderwerp van dit proefschrift. Het behandelt de veiligheid van ingebedde apparaten en de privacy van de gebruikers van zulke apparaten.

Het eerste deel van deze thesis handelt over fysieke beveiliging. Vele bestaande en toekomstige toepassingen hebben ingebouwde beveiligingscapaciteiten die steunen op de geheimhouding van cryptografische sleutels. Typische voorbeelden hiervan zijn betalingstokens, digitale identiteitsdocumenten of kaarten voor toegangscontrole. Aangezien deze apparaten werken in een vijandige omgeving, hebben ze bescherming nodig tegen fysieke aanvallen. Nevenkanaalaanvallen en foutaanvallen vormen de twee grootste bedreigingen in de beveiliging van ingebedde apparaten.

De bijdragen in dit proefschrift omvatten drie verschillende, maar verwante, aspecten. Ten eerste analyseren we de kwetsbaarheden die leiden tot fysieke aanvallen. In het bijzonder karakteriseren we de effecten van foutinjecties gedurende de setup van een low-end microcontroller. Ten tweede tonen we aan hoe fysieke aanvallen nog steeds een belangrijke bedreiging vormen voor beveiligde apparaten door een succesvolle aanval uit te voeren op een veel gebruikte familie van beveiligde geheugens. Tenslotte ontwerpen en analyseren we een hoog-niveau beveiliging tegen nevenkanaalaanvallen. Meer specifiek gebruiken we een constructie gebaseerd op het inwendig product om een maskeringsgebaseerde tegenmaatregel te ontwerpen die implementeerbaar met een willekeurig aantal delen.

Het tweede deel van deze thesis handelt over privacy-aspecten. Systemen zoals locatie-gebaseerde diensten, monitoring voor gezondheidszorg, of slimme huizen steunen op de verzameling en verwerking van fijnkorrelige informatie over de gebruikers. De gevaren die voortkomen uit het analyseren, delen of misbruiken van zulke gegevens zijn bijvoorbeeld discriminatie, vervolging of reputatieschade voor eindgebruikers tot het observeren van individuen op grote schaal. Privacy-bewarende methoden die het bewerken en/of vrijgeven van persoonlijke gegevens minimaliseren zijn van groot belang om deze systemen in praktijk te aanvaarden.

Als voorbeeld van een locatie-gebaseerde dienst beschouwen we elektronische tolheffing. Dit systeem laat overheden toe om belastingen te heffen op het gebruik van openbare wegen door middel van een ingebedde eenheid in het voertuig. Onze belangrijkste bijdrage is het ontwerp, de analyse en de implementatie van PrETP, een privacy-bewarende oplossing voor elektronische tolheffing. De garanties die ons systeem biedt op vlak van privacy worden bereikt door de ingebedde eenheden in staat te stellen om de correcte wegentol lokaal te berekenen. Tegelijkertijd zijn de dienstverleners in staat om te controleren dat de ingebedde eenheden de juiste berekeningen uitvoeren.

Abbreviations

AES	Advanced Encryption Standard
ALU	Arithmetic Logic Unit
APDU	Application Protocol Data Unit
CMOS	Complementary Metal-Oxide-Semiconductor
CFA	Collision Fault Analysis
CPA	Correlation Power Analysis
CPU	Central Processing Unit
CRT	Chinese Remainder Theorem
DEMA	Differential Electromagnetic Analysis
DES	Data Encryption Standard
DFA	Differential Fault Analysis
DoM	Difference of Means
DPA	Differential Power Analysis
DRAM	Dynamic Random-Access Memory
DRP	Dual-Rail Precharge
DSA	Digital Signature Algorithm
ECC	Elliptic Curve Cryptography
EEPROM	Electronically Erasable Programmable ROM
EM	Electromagnetic
ETP	Electronic Toll Pricing

FIB	Focused Ion Beam
FF	Flip-Flop
FPGA	Field-Programmable Gate Array
GPS	Global Positioning System
GSM	Global System for Mobile Communications
HD	Hamming Distance
HW	Hamming Weight
HW/SW	Hardware/Software
ICT	Information and Communication Technologies
IFA	Ineffective Fault Analysis
IO	Input/Output
IoT	Internet of Things
ITS	Intelligent Transport Systems
MCU	Microcontroller Unit
MI5	Military Intelligence, Section 5
MIA	Mutual Information Analysis
NSA	National Security Agency
OBU	On-Board Unit
OP	Optimistic Payment
PAYD	Pay As You Drive
PETs	Privacy-Enhancing Technologies
POI	Point of Interest
PUF	Physical Unclonable Function
RAM	Random-Access Memory
RFID	Radio-Frequency Identification
RISC	Reduced Instruction Set Computing
RNG	Random Number Generator
ROM	Read-Only Memory

RSA	Rivest-Shamir-Adleman Algorithm
SABL	Sense Amplifier Based Logic
SEA	Safe-Error Analysis
SEM	Scanning Electron Microscope
SEMA	Simple Electromagnetic Analysis
SHA	Secure Hash Algorithm
SNR	Signal to Noise Ratio
SPA	Simple Power Analysis
SPN	Substitution-Permutation Network
SSL	Secure Sockets Layer
TC	Toll Charger
TSP	Toll Service Provider
WDDL	Wave Dynamic Differential Logic

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Security and Privacy Aspects of Embedded Systems

With roots dating back to the ancient civilizations, *cryptography* - the art of concealing written information - is a well-studied area that has developed through the ages in an arms race with its counterpart *cryptanalysis* - the art of analyzing and breaking codes. Both disciplines, encompassed under the science of *cryptology*, have historically addressed military and diplomatic demands for secret communications. From the simple substitution ciphers of the Roman Empire to the complex rotor machines used in World War II, *cryptosystems* and *cryptodevices* have evolved through history in parallel with technology. In the last decades, the advent of Information and Communication Technologies (ICT) has triggered major advances in the field. The historical goal of ensuring secrecy over an insecure channel has been updated and extended to support secure communication protocols in the Internet era. Today, confidentiality, message integrity, and entity authentication provide security pillars that enable online applications such as e-mail, e-commerce, or e-banking.

The current paradigm of pervasive computing constitutes an ongoing shift towards a scenario in which billions of ubiquitous and inter-connected devices are being deployed worldwide. Mobile phones, radio-frequency identification tags (RFIDs), medical electronics, automotive components, home appliances, and a myriad of other *embedded devices* are representative of this trend. With physical constraints in resources, size, and computing power, these devices not only inherit the complex security requirements of networked devices, but face a whole range of new problems due to their embedded nature. Attack points are no longer restricted to insecure communication channels, as cryptodevices themselves operate in hostile environments and in physical possession of potential adversaries. Credit cards, electronic IDs and passports, or multimedia players are examples of ideal targets for malicious users or organizations aiming at financial fraud, identity theft, or product counterfeiting. Adding sound protection mechanisms to such already constrained devices goes beyond the design of secure cryptosystems, opening a challenging and active research area in the field of *physical security*.

The foreseen interconnection of billions of uniquely identifiable and addressable smart devices in the so-called Internet of Things (IoT) not only involves new security challenges. The deployment of *monitoring systems* with abilities to collect, process, and communicate data about themselves and their close environment is already raising *privacy* concerns, as this may reveal personal information about the person to whom the device is associated. While this information can be used to the user's benefit, for instance to improve and personalize services, it can also be shared or misused to build user profiles that can result in discrimination, persecution or reputation damage. Ignoring privacy concerns may result in undesired consequences for service providers, as illustrated by the recent revocation of the Dutch smart metering bill [77]. The integration of *Privacy-Enhancing Technologies* (PETs) into these envisioned services is a complex problem that requires balancing technical aspects with social, legal, and even ethical concerns.

The Black-Box Security Model

One of the first enabling foundations of modern cryptography was the work of Auguste Kerckhoffs in the second half of the 19th century [133]. In his article *La Cryptographie Militaire*, Kerckhoffs gave a series of recommendations for the design of military ciphers. Amongst them, the so-called *Kerckhoffs' principle* states that the security of a cryptosystem should not rely on the secrecy of the employed algorithms, but *only* on the secrecy of the key(s). This axiom has been long embraced by the cryptographic community to support the need for open cryptographic design, and has laid down the basis for the black-box security model depicted in Figure 1.1.

The cipher is an abstract mathematical object composed by two algorithms: encryption under key K, denoted as E_K, is the process of converting an input message P (the *plaintext*) to an unreadable string C (the *ciphertext*); decryption under key K', denoted as $D_{K'}$, is the complementary operation to revert an input ciphertext C to the original plaintext P. The keys need to be accordingly distributed between the communicating parties, i.e. an encryption key K must be in possession of the *sender*, whereas the corresponding decryption key K'

must be in possession of the *receiver*. When the key used for both encryption and decryption is the same we talk about *symmetric-key cryptography*, whereas when the keys are different - although somehow related - we talk about *public-key cryptography*. Block ciphers such as DES [5] or AES [6] and stream ciphers such as Trivium [82] are exemplary of the former category, whereas cryptosystems such as RSA [189] or ECC [163, 138] are representative of the latter.

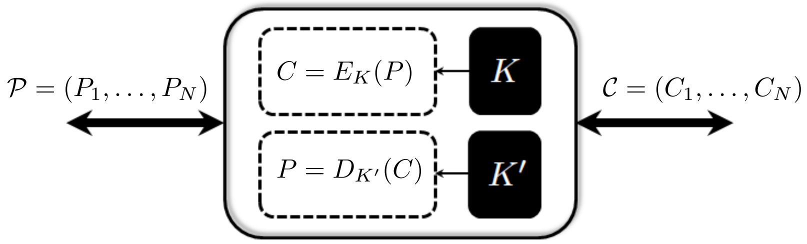

$\mathcal{P} = (P_1, \ldots, P_N)$
$C = E_K(P)$
K
$P = D_{K'}(C)$
K'
$\mathcal{C} = (C_1, \ldots, C_N)$

Figure 1.1: The black-box security model.

The black-box model provides a framework to assess the mathematical security of a cryptographic algorithm against an attacker whose goal is typically the extraction of either of its secret key(s) K or K'. The adversary is assumed to have knowledge of both encryption E and decryption D functions. Additionally, he is given the ability to query the black-box by providing a set of plaintexts $\mathcal{P}$ (resp. ciphertexts $\mathcal{C}$) and analyze the encryption (resp. decryption) results of the black-box under the key K (resp. K'). Depending on the message selection and the control the adversary has on setting them, the literature often distinguishes between different forms of black-box cryptanalysis. Ciphertext-only attacks, known-plaintext attacks, chosen-plaintext attacks, chosen-ciphertext attacks, adaptive chosen-plaintext attacks, and adaptive chosen-ciphertext attacks [158] are the most common ones.

The security assessment of cryptographic primitives depends also on their type. Symmetric-key algorithms rely on their resistance to probabilistic methods such as linear cryptanalysis [154] or differential cryptanalysis [41]. Public-key algorithms on the other hand are based on the computational intractability of certain mathematical problems such as integer factorization in RSA or the elliptic curve discrete logarithm problem in ECC. Independently of the approach followed, a cryptographic primitive is considered secure as long as the best known attack is the exhaustive search of the key, i.e. check plaintext/ciphertext pairs against all key space. If the search space is such that it exceeds the computational resources of an adversary, then the algorithm is assumed to be *computationally secure*.

The Grey-Box Security Model

In the late 90s, the diversification of cryptographic services and applications towards the world of low-end embedded devices triggered the appearance of a whole new range of threats often referred to as *physical attacks*. Contrary to black-box attackers, physical adversaries do not target the mathematical strength of cryptographic algorithms by solely analyzing input/output messages. Rather differently, they exploit additional leakage sources that stem from the particular *implementation* of the algorithms and/or from the embedded essence of the target device.

A cryptographic implementation on an integrated circuit is inherently subject to physical phenomena of various natures: temporal, electrical, electromagnetic, or optical, to name a few. Measurable quantities linked to internal states and operations performed by a circuit are naturally emanated through these channels. As such, they constitute a rich source of information that can be passively captured and analyzed by adversaries aiming to break the device's security properties. In addition to this, most of these physical channels can be actively altered and exploited by adversaries in possession of the circuits. By disrupting or manipulating the expected behavior of the device, it is possible to trigger errors that open the door to unexpected vulnerabilities. The grey-box model depicted in Figure 1.2 takes these considerations into account to provide an assessment framework for practical realizations of cryptography.

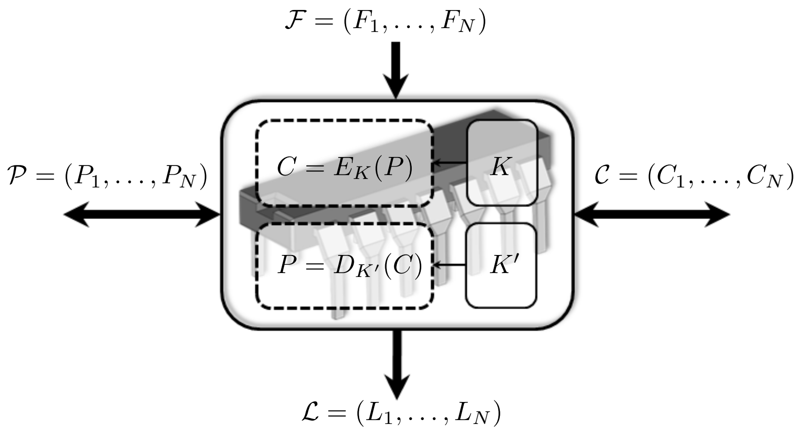

$\mathcal{F} = (F_1, \ldots, F_N)$
$\mathcal{P} = (P_1, \ldots, P_N)$
$C = E_K(P)$
K
$P = D_{K'}(C)$
K'
$\mathcal{C} = (C_1, \ldots, C_N)$
$\mathcal{L} = (L_1, \ldots, L_N)$

Figure 1.2: The grey-box security model.

The capabilities of a physical adversary are extended by two new interfaces.

A first channel denoted by $\mathcal{L}$ models any measurable leakage source that can be naturally captured from the device under attack. These leakage data, in what follows called *side channel information*, can be later exploited by *side channel analysis.* A second channel denoted by $\mathcal{F}$ models any physical alteration performed by an adversary on the cryptographic device, typically with the aim of disturbing its computation(s). The outcome of this disruption is often the reception of a set of corrupted outputs $\tilde{\mathcal{C}}$ (resp. $\tilde{\mathcal{P}}$) such that $\tilde{C} \neq C$ (resp. $\tilde{P} \neq P$). The *faulty information* collected as a result of this external manipulation may be subsequently exploited using *fault analysis* techniques.

Adversarial Models

Classifying the type of adversaries and their abilities is critical to determine the security requirements of a system. In the following we provide a widely accepted taxonomy originally introduced by IBM [8] and later adopted by Anderson and Kuhn [16] to the case of tamper-resistant embedded devices. It groups physical adversaries into three classes depending on their strength:

- Clever Outsider (*Class I*). An often very intelligent attacker, but constrained in terms of budget, time, and expertise. He possesses only limited information on the system under attack, e.g. publicly available documents. Adversaries in this category are likely to exploit known weaknesses rather than creating new ones.
- Knowledgeable Insider (*Class II*). A skilled specialist with vast experience and prior technical training. He possesses a wide knowledge of the targeted system, often a result of inside information. While his resources are not unlimited, he is likely to have access to advanced equipment.
- Funded Organisation (*Class III*). The most powerful adversarial model. It is often composed of a team of specialists backed by great funding resources, for instance, a country, a large corporation, or even a criminal organisation. Because of this, this adversary is capable of carrying out highly sophisticated attacks that require the possession - or even the design - of professional tools.

Taxonomy of Attacks

The wide range of existing physical attacks gives rise to multiple categorization options. Time, cost, or expertise, are just a few factors that can enable different partitions of the attack space. In the following, and as commonly accepted in the field, we categorize physical attacks according to two criteria [151].

The first criterion allows to divide physical attacks according to how an adversary manipulates the target device under the grey-box model. We denote *passive* attacks those in which the cryptographic device operates according to its functional specifications and the adversary simply collects data by monitoring the physical information channel $\mathcal{L}$. Side channel attacks fall in this category. In contrast, we denote *active* attacks those in which the expected normal functioning of the target device is intentionally manipulated through the channel $\mathcal{F}$ with the goal of inducing an erroneous or unexpected behavior. Fault attacks are part of this category. Note that these attack approaches are not mutually exclusive, and therefore can be carried out simultaneously by an adversary. This is commonly referred to as *combined* attacks.

Orthogonally to the previous classification, a second criterion allows to divide physical attacks according to the level of *intrusion* performed on the circuit. Non-invasive attacks are those in which no physical alterations are performed on the device. In other words, the adversary operates the circuit in its original manufacturing form. Semi-invasive attacks on the other hand involve a certain degree of physical intrusion on the device, typically conditioned to the passivation layer not being damaged. In practice, all approaches based on decapsulation, i.e. eroding the chip surface by mechanical or chemical means, fall in this category. Finally, invasive attacks are not subject to any limitations and typically involve access to the inner elements and circuitry of the target device.

Besides providing a more granular taxonomy, the latter classification gives an intuition of the cost of the attack, i.e. the higher the degree of intrusion, the higher the cost of the equipment required to mount the attack. Invasive attacks may involve expensive equipment such as Scanning Electron Microscopes (SEM) or Focused Ion Beams (FIB). Due to this, they are out of reach to most physical adversaries including those in Class I.

Embedded Design for Physical Security

The grey-box model underlines the importance of securing embedded systems. In fact, the selection of computationally secure cryptographic primitives and protocols is only one of many steps. Integrating countermeasures against physical attacks is critical to obtain robust systems. Along with throughput, area, or energy consumption, security must be considered as a dimension of the design space [191]. For designers, this implies to follow a systematic approach considering all abstraction layers of the embedded design space, as illustrated in Figure 1.3.

The *system/protocol* layer specifies the security properties of the system, e.g. confidentiality, integrity, entity/data authentication, or non-repudiation. These are derived from the functionality and/or goals of the end application, and they are achieved through secure cryptographic protocols. The *algorithm* layer contains the building blocks required to instantiate the secure cryptographic protocols. These may include cryptographic primitives such as public-key algorithms, symmetric-key ciphers, and hash functions. The *architecture* layer determines the platform in which cryptographic algorithms are implemented. Depending on the system requirements, these may be developed in *software* (executed on a microcontroller), in *hardware* (executed on a dedicated co-processor), or even in a partitioning of both approaches represented by *hardware/software* co-design space [190]. Finally, the *circuit* layer provides the mapping of the architecture level at transistor level.

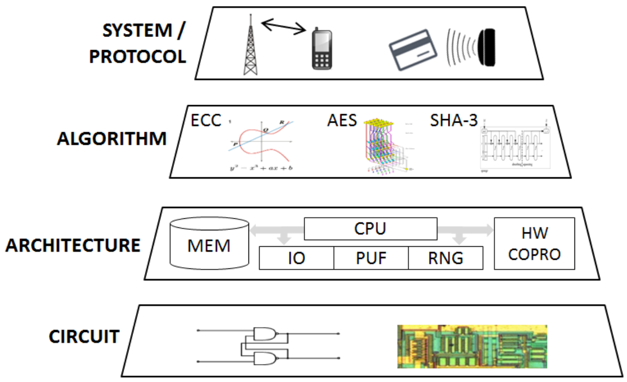


Figure 1.3: The embedded design space.

Mitigations against physical attacks can be instantiated at any of these levels of the design space. However, it is important to highlight at this point that no perfect solution exists to counter all physical attacks. While one mitigation strategy may be suitable against a certain class of attacks, it is often the case that its security may be compromised when considering other approaches. In practice, chip manufacturers and smart card developers often opt for a suitable combination of countermeasures against a well-defined adversary. A product is considered secure when the amount of resources needed for breaking it outweighs the capabilities of the attacker. In practice, the resistance of security-related products against physical attacks is evaluated by specialized labs prior to obtaining a security certificate.

Embedded Design for Privacy

Nowadays, most deployed monitoring systems are implemented following a centralized architecture as depicted in Figure 1.4. Users (*data subjects*) provide information about themselves or their direct environment to the service provider (*data controller*), who can later process, analyze, and store it in order to provide the service. This data can range from the vital signs registered by body sensors, to the electrical consumption measurements of a smart meter or the location data collected by smartphones and in-vehicle tolling devices. From a practical perspective this kind of centralized architectures are easy to design and deploy. The intelligence and core functionality of the system resides in the domain of the data controller, which is the central element in the system. This results in minimal computing requirements for the monitoring devices and, subsequently, cheaper development costs.

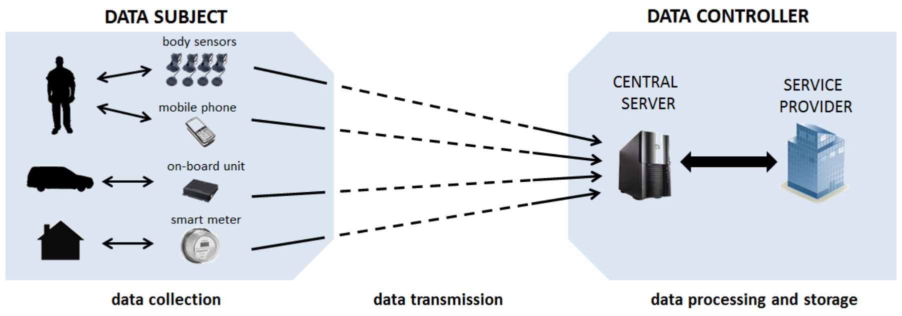


Figure 1.4: Centralized architecture for exemplary monitoring systems.

From a privacy perspective however, these type of architectures suffer from a major downside. The data controller is assumed to be a *trusted party* who acts as a well-behaved keeper of users' personal information. If that is not the case, then the information disclosed by users is not secret and privacy assurances do not hold. Malicious service providers have the motivation and ability for exploiting the collected data, for instance, in order to obtain advantageous business positions. Other attack scenarios include external adversaries - or even rogue employees - accessing or sharing the collected data driven by economical incentives. Overall, it is easy to see that a central entity storing all collected information represents a *single point of failure*. Furthermore, its existence opens the possibility of personal information being systematically accessed by national law enforcement agencies, see for instance the NSA Prism program [7]. This leads to a scenario in which customers' privacy is not only violated, but in fact individuals are subject to massive *surveillance*.

The development of architectures offering strong privacy guarantees is an active area of research. The goal of these approaches is to integrate Privacy-Enhancing Technologies (PETs) as part of the system. Following the lines in [86], we define PETs as a series of technological solutions aiming to the elimination of the single point of failure inherent to centralized architectures and to the application of data minimization principles in the system's flow of information. Such guarantees can be achieved by applying a data pre-processing in the monitoring device, in such a way that only the minimum amount of information required for the provision of the service is actually disclosed to the data controller. Intuitively, this implies to de-centralize (part of) the computing functionalities from central servers to the embedded devices, resulting in potentially higher deployment costs. In contrast, the reduction of personal data within their databases may benefit data controllers. First, by reducing maintenance costs. And second, by lowering the risk of financial impact and damage to their reputation due to potential privacy breaches.

Practical deployments of privacy-preserving architectures are nevertheless not straightforward, and multiple system aspects have to be considered. Data minimization techniques must be applied without being detrimental for the provision of the service. As there is no generic mechanism or set of rules to be applied, every system requires a careful balance between its functional requirements and the privacy of users. Moreover, the integration of PETs in the system should not open the door to security vulnerabilities. This requires to perform careful threat analyses considering, among others, the applicability of physical attacks on the monitoring devices. Finally, the proposed solutions should rely on realistic assumptions and its deployment cost should not be prohibitive for the service provider.

1.1 About this Thesis

This thesis deals with implementation aspects of security and privacy into embedded systems. Its contents can be mapped to the embedded design pyramid [191] as illustrated in Figure 1.5. The focus of the first part of the dissertation is devoted to physical attacks and covers three main aspects: analysis of low-level vulnerabilities, exploitation of physical attacks, and design of mitigation techniques. Because of the broad nature of the field, our latter studies are restricted to the exploitation of *non-invasive* techniques and to the design of *high-level* countermeasures.

On the exploitation side our choice originates from vulnerability assessment. As non-invasive attacks can be carried out with relatively inexpensive equipment and limited specialized training, weaknesses identified by attackers and brought to the public knowledge can consequently be reproduced at large scale by a myriad of potential adversaries. This poses a serious threat to the security of real world embedded systems. On the mitigation side our choice arises from the fact that security provided by high-level countermeasures can often be formally proven. In other words, mitigations for physical attacks can be designed while acknowledging the physical vulnerabilities of the low-level layers.

The focus of the second part of the dissertation is devoted to privacy. Here, we show how the integration of PETs in a system can lead to strong privacy guarantees for end users. As a use case for our study, we select a high impact representative application of monitoring systems: Electronic Toll Pricing (ETP). This envisioned system is expected to be deployed in the European Union to enable variable vehicle taxing. As this system requires to process fine-grained location data about citizens, ensuring privacy protection becomes a critical design condition.

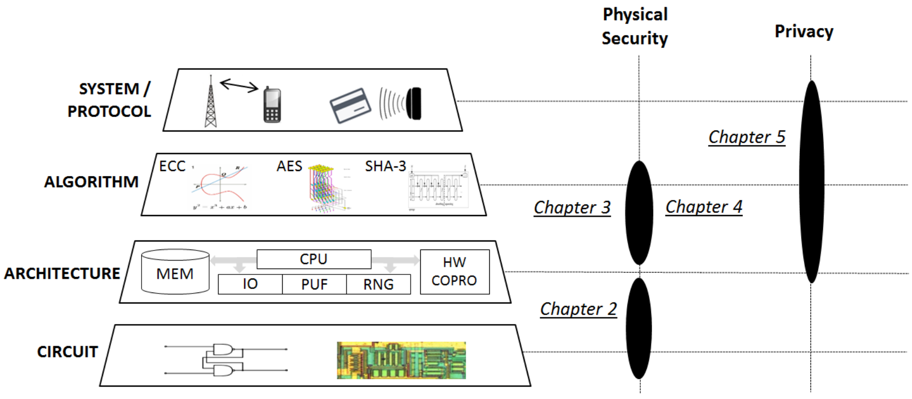


Figure 1.5: Thesis organization and link to embedded design space.

Outline and Summary of Contributions

This thesis is structured in six chapters. A description of their content and our personal contributions within each of them follows:

Chapter 1. The first chapter gives a brief introduction to the topics of embedded design for physical security and privacy. We first describe and compare the security assessment frameworks provided by the black-box and the grey-box models. Then, we introduce a taxonomy for physical adversaries and physical attacks, and we introduce the characteristics of the embedded design space. Next, we analyze how architectures for monitoring systems may lead to an invasion of users' privacy and how the integration of PETs can address this issue. We finish by defining the scope of this thesis and by summarizing the contents and contributions of each chapter.

Chapter 2. The second chapter entitled "An Insight into Physical Vulnerabilities" dives deeper into the topic of physical attacks. It explores which particularities of embedded devices and/or their implementations result in vulnerabilities. The goal of the chapter is to gain insight into the origin of physical security issues, particularly, what type of information leakage is exploited in side channel analysis and which type of induced errors enable fault analysis. A review of state-of-the-art countermeasures against physical attacks is also provided.

Our contribution within this chapter is a complete study and characterization of the effects of fault injections based on setup-time violations. These results are presented by Balasch, Gierlichs and Verbauwhede [23] at the workshop on Fault Diagnosis and Tolerance in Cryptography (FDTC 2011).

Chapter 3. The third chapter entitled "A Motivating Example" describes a combination of non-invasive attacks against a widely used family of secure EEPROM memories. The goal of this chapter is twofold. First, to illustrate the threat that even nowadays is posed by non-invasive physical attacks; and second, to clearly motivate the need for sound mitigation techniques.

Our contribution within this chapter is a security evaluation of the Atmel CryptoMemory family of secure EEPROMs against physical attacks. These results are presented by Balasch, Gierlichs, Verdult, Batina and Verbauwhede [24] at the Cryptographers' Track at the RSA Conference (CT-RSA 2012).

Chapter 4. The fourth chapter entitled "Masking at Algorithm Level" dives into the topic of high-level countermeasures, in particular, mitigations based on masking. As these techniques can be implemented at algorithm level and their security formally proven, they represent an active research area of countermeasures. We provide a comprehensive review of both state-of-the-art masking techniques and higher-order side channel attacks.

Our contribution within this chapter is the design and implementation of a novel masking scheme based on the *inner product* that can be implemented at any order. These results are presented by Balasch, Faust, Gierlichs and Verbauwhede [22] at the Annual International Conference on Theory and Application of Cryptology and Information Security (ASIACRYPT 2012).

Chapter 5. The fifth chapter entitled "Enabling Privacy in Embedded Design" focuses on the topic of privacy integration on monitoring systems. After reviewing common issues with currently deployed architectures, we discuss how the use of PETs can guarantee certain privacy properties for end users.

Our contribution within this chapter is the design, analysis, and implementation of PrETP, a privacy-preserving scheme for envisioned ETP applications. These results are presented by Balasch, Rial, Troncoso, Geuens, Preneel and Verbauwhede [26] at the USENIX Security Symposium (USENIX 2010).

Chapter 6: The last chapter concludes this thesis and discusses open security and privacy issues in the field of embedded systems.

Other Contributions

This dissertation includes only a selection of our published works in the fields of security and privacy. In the following we summarize our remaining contributions which are not included in the core of this thesis. The publications are grouped according to their main topic.

Attacks on Cryptographic Devices.

Gone in 360 Seconds: Hijacking with Hitag2, presented by Verdult, Garcia and Balasch [222] at the USENIX Security Symposium (USENIX 2012). The work introduces a series of vulnerabilities in the Hitag2 transponders, a family of vehicle immobilizers widely used in the automotive industry. Hitag2 allows replaying reader data to the transponder, provides an unlimited keystream oracle, and uses only one low-entropy nonce to randomize a session. When wireless access to the vehicle and key is available, these weaknesses can be exploited by adversaries to recover the secret key within seconds. If only a single communication with the vehicle is possible, the running time of the attack is still less than six minutes. All proposed attacks are demonstrated in practice by experimenting with more than 20 vehicles of various makes and models.

Cryptographic Implementations.

Teaching HW/SW Co-Design With a Public Key Cryptography Application, presented by Uhsadel, Ullrich, Das, Karaklajic, Balasch, Verbauwhede and Dehaene [218] in IEEE Transactions on Education. The work describes a lab session-based course on hardware/software co-design. The goal of the course is twofold. First, to illustrate the multiple alternative solutions available in the embedded design space. And second, to teach the fundamental concepts of hardware/software co-design. The sample application for the course project is a basic public-key application based on RSA. The project follows a step wise approach with assignments that build on each other. Students are required to make their own decisions as to the partitioning between hardware and software, the design of a communication interface, and the optimization goals. Besides imparting hard skills in embedded design, students gain several soft skills often overlooked in engineering, particularly, decision making, presentation skills, teamwork, and design creativity.

Compact Implementation and Performance Evaluation of Hash Functions in ATtiny Devices, presented by Balasch, Ege, Eisenbarth, Gérard, Gong, Güneysu, Heyse, Kerckhof, Koeune, Plos, Pöppelmann, Regazzoni, Standaert, Van Assche, Van Keer, van Oldeneel tot Oldenzeel and von Maurich [21] at the Smart Card Research and Advanced Application Conference (CARDIS 2012). The work provides a study on the software performance of multiple hash functions including the SHA-3 finalist candidates, SHA-256, recent lightweight proposals, and constructions based on block ciphers. All designs are implemented in software using an Atmel AVR ATtiny45 target platform, with the goal of minimizing code size and memory utilization. Using a common interface, the performance of all implementations is evaluated based on three main metrics: code size, RAM usage, and cycle counts for different message sizes.

Privacy-Preserving Systems.

A Privacy-Preserving Buyer–Seller Watermarking Protocol Based on Priced Oblivious Transfer presented by Rial, Balasch and Preneel [186] in IEEE Transactions on Information Forensics and Security. The work proposes a novel approach for the design of privacy-preserving buyer-seller watermarking protocols. In the proposed system, the seller authenticates buyers without learning which items are purchased. As buyers are not anonymous, customer management is eased and currently deployed methods of payment can be utilized. We define an ideal functionality for privacy-preserving copyright protection protocols. To realize our functionality, a protocol must ensure that buyers pay the right price without disclosing the purchased item, and that sellers are able to identify buyers that released pirated copies. We construct a protocol based on priced oblivious transfer and on existing techniques for asymmetric watermark embedding. Furthermore, we implement and evaluate the efficiency of our protocol, and we explain how to extend it in order to achieve optimistic fair exchange.

An Embedded Platform for Privacy-Friendly Road Charging Applications presented by Balasch, Verbauwhede and Preneel [27] in Design, Automation and Test in Europe (DATE 2010). The work presents a practical and functional ETP system based on PriPAYD [215]. We develop a functional on-board-unit capable of processing location data in real-time, while minimizing overheads required to ensure security and privacy. The performance of our software-based prototype is tested and proves that the deployment of a privacy-friendly solution can be achieved within a minimum cost increment compared to existing schemes.

PriPAYD: Privacy Friendly Pay-As-You-Drive Insurance (Journal version) presented by Troncoso, Danezis, Kosta, Balasch and Preneel [214] in IEEE Transactions on Dependable and Secure Computing. This works is an updated and extended publication resulting from merging the original PriPAYD proposal in [215] and our work in [27].

Chapter 2

An Insight into Physical Vulnerabilities

Understanding Leakage Sources on Embedded Devices

The aim of this chapter is to give an overview of the works in the research area of physical attacks. We divide the contents of the chapter in three main blocks: side channel attacks, fault attacks, and countermeasures. The two former focus on historical and state-of-the-art aspects related to attacks on cryptographic implementations, including descriptions of the origins of leakages exploited by physical adversaries. The latter block gives an overview of mitigation techniques currently employed to secure cryptographic devices.

2.1 Side Channel Attacks

A *side channel attack* is a particular type of physical attack carried out by a *passive* adversary. Its core idea is to exploit leakage sources present in cryptographic implementations of computationally secure mathematical algorithms. This leaked information - often referred to as *side channel* - stems from observable and measurable phenomenons caused by the physical essence of digital circuits. As such, its origin is often unintentional and in most cases difficult to prevent, resulting in a major threat for the security of practical realizations of cryptography.

One of the first successful applications of side channel attacks to break cryptographic devices is documented by former MI5 scientist Peter Wright in his memoirs [228]. Among others, the book relates the efforts of the British secret services during 1965 to intercept the communications of the Egyptian embassy, which were encrypted by a Hagelin rotor machine under a daily updated key. Wright suggested to place a microphone close to the cipher machine in order to record the sound of the rotors when new keys were being set. This additional information, together with the possession of an exact replica of the Hagelin machine, enabled MI5 to deduce the position of a few rotors, facilitating the extraction of the Egyptian secret keys.

In addition to its historical relevance, this story provides a perfect illustration of the powerful nature of side channel attacks. An adversary limited by his computational power in front of a computationally secure cipher, may use other sources of information to simplify cryptanalysis.

The rest of this section deals with the topic of side channel attacks. We start by reviewing the most common types of side channels reported in the open literature, distinguishing whether they can be accessed non-invasively or, in contrast, they require a certain degree of invasion on the device. Then, we dive deeper into the study of leakage in CMOS devices. We provide some insight on the source of these vulnerabilities and describe the most relevant attacks in the field.

2.1.1 An Overview of Side Channels

2.1.1.1 Non-Invasive Side Channels

Execution Time. The first side channel attack published in the open literature dates from 1996 and is due to Paul Kocher [140]. The leakage source targeted in this seminal work is the variable execution time of cryptographic implementations. Because of optimizations in the underlying software libraries, certain arithmetic operations require more or less time to execute given different input parameters. Kocher showed how timing variances in modular multiplications allow an adversary to recover private keys used in cryptosystems such as Diffie-Hellman [87] and RSA [189] after applying some statistical analysis.

Follow-up works such as Dhem *et al.* [85] demonstrated the applicability of timing attacks on portable cryptographic tokens, while Brumley and Boneh [56] successfully targeted RSA implementations running on a local OpenSSL-based web server.

Power Consumption. Three years after the introduction of timing attacks, Kocher, Jaffe and Jun [141] published a yet more threatening form of side channel: the power consumption. Kocher *et al.* showed that the instantaneous power consumption of a circuit over time is linked to the intermediate values and operations being processed, and presented two attacks capable of evaluating this dependence:

- *Simple Power Analysis* (SPA) exploits key-dependent patterns in the power consumption present in one (or very few) leakage measurements, often by simple visual inspection. Although the interpretation of power measurements requires some expertise and/or knowledge on the circuit and on the implementation, this form of attack is particularly devastating for algorithms where power patterns can be directly linked to key-bit dependent operations or branches.
- *Differential Power Analysis* (DPA) exploits the leakage present on a larger set of leakage measurements. As opposed to SPA, it does not require knowledge on implementation details and its basic principles are not algorithm dependent. The first step in a DPA attack consists in building a model to estimate the power consumption of the device given a set of possibilities on a computationally suitable (sub-)key space. The second step consists in evaluating the dependence between the power model and the leakage measurements using a statistical distinguisher that yields the strongest dependency for the correct key guess.

Electromagnetic Emanations. The electromagnetic (EM) side channel was independently proposed by Gandolfi *et al.* [101] and Quisquater and Samyde [182] in the early 2000s. Every changing current or voltage within a circuit generates an electromagnetic field. Measurements of this field over time inherently carry information about the circuit's internal behavior, which may in turn relate to the execution of a cryptographic algorithm.

Due to their similar leakage origin, attacks exploiting the EM side channel are analogous to the ones exploiting power consumption, namely *Simple EM Analysis* (SEMA) and *Differential EM Analysis* (DEMA). Note however that while power consumption is typically tied to *global* observations of a circuit, the electromagnetic field allows to focus on *local* elements over its surface.

Other Non-invasive Side Channels. While execution time, power consumption, and EM emanations are often acknowledged as the most relevant and threatening side channels, other leakage sources that can be potentially exploited in a non-invasive manner have been introduced in related works:

- *Temperature.* Brouchier *et al.* [55] show how it is possible for concurrent processes running on a computer to exploit the temperature side channel, i.e. heat dissipation of the CPU, in order to gain knowledge on internal computations. Access to this side channel can indirectly be done by software commands querying the speed of the CPU fan. A recent work by Hutter and Schmidt [123] characterizes the temperature side channel on embedded microcontrollers and identifies a (low-frequency) linear relationship between heat radiation and circuit activity.
- *Visual.* Kuhn [146] delineates a mechanism to eavesdrop contents displayed by a cathode-ray tube monitor at a distance, using off-the-shelf components such as a photomultiplier tube and a computer equipped with a fast analog-to-digital converter.
- *Acoustic.* Along with the previously mentioned attack of Wright [228], a study by Shamir and Tromer [203] demonstrates that some patterns of operations can be recognized by the sound emanated by a CPU. This low-frequency side channel stems from mechanical stress due to continuous heating and cooling effects.

2.1.1.2 (Semi-) and Invasive Side Channels

The previous side channels can be accessed in the vicinity of the target device, i.e. without need of tampering with its physical structure. However, it may be the case that adversaries can profit from a certain level of intrusion if it is within their capabilities. For instance, a semi-invasive approach based on chip decapsulation allows to bring the EM coil close to the passivation layer and therefore improve the signal to noise ratio of the leakage measurements [101].

Ferrigno and Hlavác [97] recently introduced the optical side channel, exploiting the number of photons that are emitted by transistors of an integrated circuit when changing their state. Access to such information not only requires techniques to thin down and polish the silicon layer on the backside of the die, but also specialized equipment to measure emitted photons. Recent works by Schlosser *et al.* [193] and Kramer *et al.* [144] have shown how this side channel can be exploited similar to SPA and DPA techniques, respectively, while proposing alternatives to lower the cost of the equipment.

A particular type of invasive attacks is based on (micro-)probing techniques [143]. These attacks target inner elements of a circuit that store or transport sensitive information, for instance, secret cryptographic key(s). By placing a thin needle on top of such elements adversaries can directly read out values processed by the circuit. Due to this, probing based attacks are often not considered as side channel attacks, but rather a particular case of passive invasive attacks. Needless to say, access to such fine grained information requires expensive equipment and thus its applicability is bounded to specialized labs or attackers.

2.1.2 Side Channel Leakage in Integrated Circuits

Complementary Metal-Oxide-Semiconductor (CMOS) is the predominant technology used nowadays in integrated circuits. One of the characteristics of CMOS, which justifies its dominance in the field, is its low energy usage. In fact most of the energy consumption of CMOS devices occurs dynamically as the circuit changes its state, while the static power consumption of the device is only significantly increasing in deep submicron technologies. Logic cells in a circuit are typically implemented using complementary transistors, i.e. there exist a pull-up network composed of P-transistors connected to V_{DD} and a pull-down network of N-transistors connected to GND.

For illustrative purposes let us consider the case of one of the simplest CMOS cells: an inverter, whose input/output transition table is depicted in Figure 2.1 (left). When there is no input transition the inverter circuit remains idle, and only the (rather low) static component of the current flows through the conducting transistor. When there is an input transition from high to low ($1 \rightarrow 0$), the pull-up network of the circuit (P-transistor) becomes active. A dynamic current flows through the circuit in order to charge the load capacitance, as shown in Figure 2.1 (middle), effectively generating an output transition from low to high ($0 \rightarrow 1$). Finally, when the input transition goes from low to high ($0 \rightarrow 1$) the pull-down network becomes active. In this case there appears a short-circuit current that causes the load capacitance to be discharged, producing an output transition from high to low ($1 \rightarrow 0$).

$\mathbf{V_{in}}$	$\mathbf{V_{out}}$	**Current**
$0 \rightarrow 0$	$1 \rightarrow 1$	static
$1 \rightarrow 1$	$0 \rightarrow 0$	static
$1 \rightarrow 0$	$0 \rightarrow 1$	charge
$0 \rightarrow 1$	$1 \rightarrow 0$	discharge

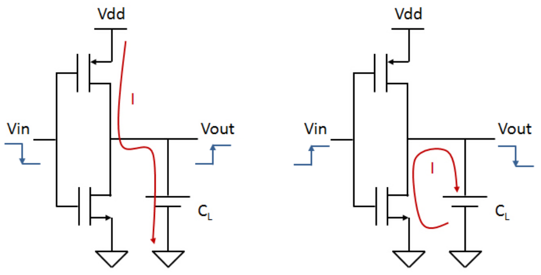


Figure 2.1: CMOS inverter. Transition table (left), charge circuit (center), discharge circuit (right).

The previous example highlights the inherent information leakage behavior of CMOS cells: dynamic current reveals information about inner transitions. As digital circuits are comprised of many logic cells, this effect can be observed at a global scale. Note however that several aspects influence and determine this leakage, for instance, the total number of logic cells, their interconnections, or the manufacturing process, among others.

2.1.2.1 Access to Side Channel Data

How to access and measure the leakage of CMOS devices is the first issue a side channel adversary must tackle. Figure 2.2 depicts the typical main elements of a measurement setup, namely a computer, an oscilloscope, and a target cryptographic device. The computer acts as the central element of the setup. It communicates with the cryptographic device, e.g. to provide commands/plaintexts and to collect ciphertexts, as well as with the oscilloscope, e.g. to retrieve digitized side channel measurements. The oscilloscope is required to acquire and record side channel signals from the target device *while* executing a cryptographic operation. Note however that oscilloscopes can only measure voltages, so the adversary needs to provide a means to “convert” the side channel of interest, i.e. power consumption or EM emanations, into a voltage signal.

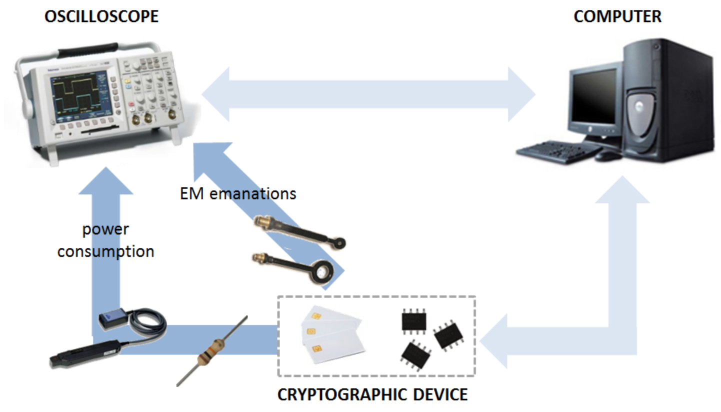


Figure 2.2: Typical non-invasive side channel measurement setup.

The instantaneous power consumption of the circuit over time can be measured by probing the voltage drop over a shunt resistor - typically of a few *Ohms* - placed into the V_{DD} or GND line of the target device. From Ohm’s Law it follows that the voltage drop over this resistor is proportional to the current that flows through it, which corresponds to the current drawn by the device.

This, in turn, is proportional to the device's instantaneous power consumption. Alternatively, one can directly employ current probes internally equipped with circuitry to convert the current in the V_{DD} or GND lines to voltage.

The electromagnetic field caused by the electrical activity of the circuit can be measured by means of H-field or E-field probes placed over the surface of the target device. These probes provide a voltage signal proportional to the field's amplitude. The size of the coil determines the granularity of the measurements; it can cover larger or smaller areas of the target device.

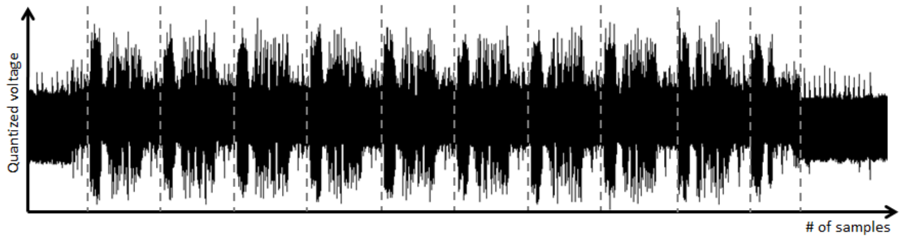
Independently of the leakage source selection, it is clear that the cost for building a non-invasive side channel measurement setup is rather low, particularly when compared to the equipment necessary to carry out invasive attacks. It is important to remark that timing side channels are also captured with such measurement setup, as oscilloscopes inherently measure over time.

2.1.2.2 Inspection of Traces

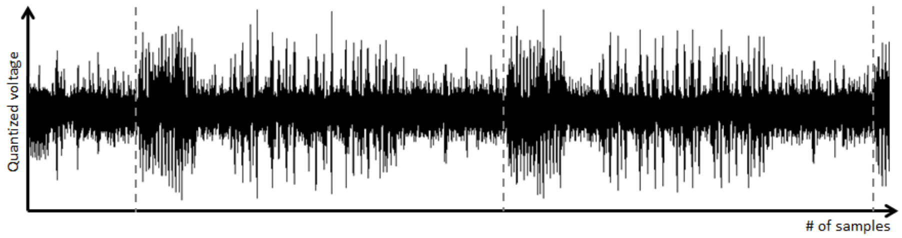
A trace captured by an oscilloscope during the execution of a cryptographic operation carries several types of information of interest to an adversary: the existence of *patterns* within the measurement is related to the structure of its implementation; the *amplitude* of the curve peaks carries information about the type of operation being performed and the data being processed; and the *timing* allows to distinguish variations in the program flow.

In order to illustrate these effects, let us consider an adversary in possession of a cryptographic device that performs bulk encryption. Let us further assume that specifications of the device, i.e. hardware or software implementation, processor type, etc., are not known to the attacker. Using a suitable measurement setup, the adversary captures a trace as shown in Figure 2.3. A quick observation of the global trace in Figure 2.3a reveals a series of patterns (highlighted by dotted lines) that stem from the structure of the algorithm's implementation. In particular, one can easily identify nine identical patterns followed by a shorter, yet rather similar one. Such a construction is consistent with the 128-bit version of the Advanced Encryption Standard (AES-128), composed of 10 encryption rounds with the particularity that the last round omits one operation.

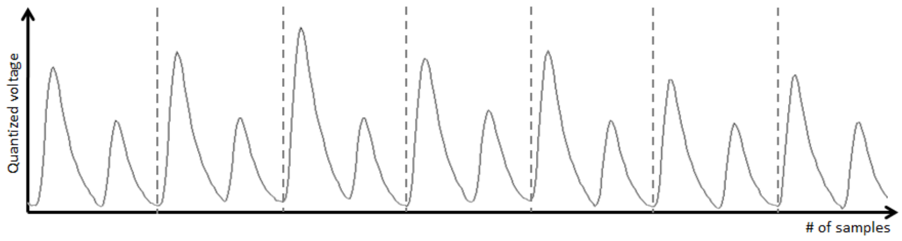
Zooming into the rounds as illustrated in Figure 2.3b allows to visualize their inner structure. The round operations of AES - `SubBytes`, `ShiftRows`, `MixColumns` and `AddRoundKey` - give rise to four distinguishable patterns. Notice that the amplitude of some peaks within the `SubBytes` and `MixColumns` computations is slightly higher than the rest. As these operations are often implemented by means of look-up tables in memory, one can infer that memory



(a) Full execution of AES-128.



(b) First two rounds of AES-128.



(c) A few cycles during the execution of AES-128.

Figure 2.3: A trace of an AES execution on an embedded device (a), zooming into first rounds (b), and zooming of a few cycles (c).

access operations consume more power than e.g. atomic operations in the Arithmetic Logic Unit (ALU).

Finally, a further zoom into a few clock cycles is depicted in Figure 2.3c. One can clearly notice here the charging/discharging effects of the CMOS cells within the circuit. The dotted lines denote the separation between consecutive clock periods; the two peaks within a single cycle indicate this particular circuit performs activities at both the rising and falling clock edges.

2.1.2.3 Modeling Traces

The information contained in a single point of a trace is often modeled as the sum of four main components [151]:

$$P_{total} = P_{op} + P_{data} + P_{noise} + P_{const}.$$

The two first components - P_{op} and P_{data} - denote the operation-dependent and data-dependent contributions, respectively; P_{noise} corresponds to electrical noise inherently caused by the measurement setup, and can be modeled as a normal distribution [151]; finally, P_{const} represents a constant independent of the operation or data processed.

Quantifying the data-dependent component P_{data} of a circuit is one of the first goals of an adversary, as this can help enhancing attacks at a later stage. One of the most employed models in the literature is the *Hamming Weight* (HW) model. It basically assumes that the data-dependent power consumption of a digital circuit is directly correlated to the number of bits equal to one in the data being processed. In other words, the power consumption of a device processing a variable Y on a certain clock cycle is proportional to $HW(Y)$.

Other models, such as the *Hamming Distance* (HD) model, take into account data transitions within consecutive clock cycles. In this case the power consumption of a device when switching from a value X to a value Y is given by $HD(X, Y) = HW(X \oplus Y)$. Figure 2.4 depicts a series of traces obtained from a CMOS device when performing data transitions on an 8-bit data bus. For each power trace, two random 8-bit values ranging from 0 to 255 are consecutively loaded from RAM to CPU registers. We average all traces with equal Hamming distance in order to remove the effects of the noise. By doing this, one can easily verify that the data-dependency of this device follows a Hamming distance model: the number of bit transitions between memory operations determines the amplitude of the measurement peak.

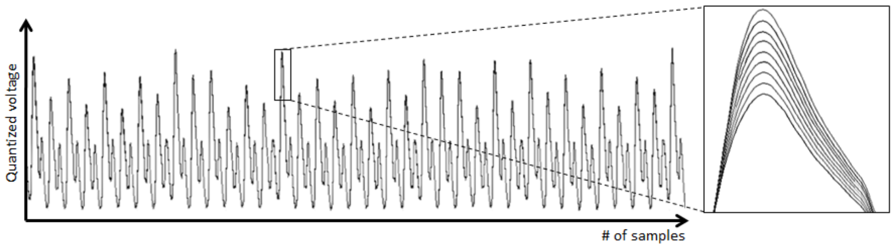


Figure 2.4: Transitions on the data bus of a CMOS device.

Further leakage models proposed in the literature aim to consider different bit transition leakages [171]. The research area of profiled attacks, which will be discussed later in the chapter, uses advanced statistical tools to fully characterize a device's leakage [63, 192].

2.1.3 Exploitation of Traces

So far we have introduced the type of information observed and/or expected in power and EM measurements. In the following we review the most common types of *power analysis* attacks proposed in the literature. Note that these attacks can be carried out with EM measurements instead of power traces.

2.1.3.1 Simple Power Analysis

Simple Power Analysis (SPA) was introduced in the seminal work by Kocher *et al.* [141]. The core idea of this attack is the direct exploitation of key-dependent leakages present in one or a small number of power traces. Its application is often based on heuristic analyses of visually identifiable leakages within traces. Because of this, and contrary to its name, it often requires experienced adversaries with detailed knowledge of the cryptographic implementation and the architecture of the target device.

The first class of SPA attacks, as well as the most exemplary, targets key-dependent *patterns* within power curves. This is particularly troubling in public-key cryptosystems with algorithms containing conditional statements directly dependent on secret key bits. A typical example of such constructions is given by binary modular exponentiations [158], one of the core algorithms in public-key cryptography. Algorithm 1 depicts the left-to-right binary algorithm for point multiplication, a basic operation in ECC based cryptosystems. This algorithm iterates over each bit of the exponent and always performs a point doubling operation. If the scanned bit equals one a point addition operation is as well executed.

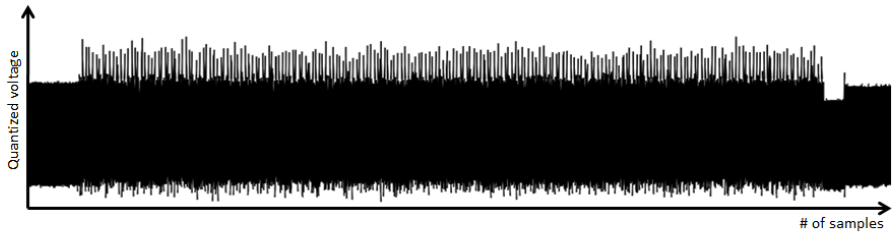
One can visually derive the leakage in this algorithm: the sequence of point doublings and point additions in a *single* power trace directly yields the value of the scalar k. If the point multiplication is carried out for instance in the context of ECDSA - the elliptic curve version of the Digital Signature Algorithm (DSA) - then knowledge of the ephemeral scalar k easily leads to the recovery of the secret key [155].

For illustrative purposes, we have performed an SPA attack against an implementation of Algorithm 1 on a 160-bit standard compliant elliptic

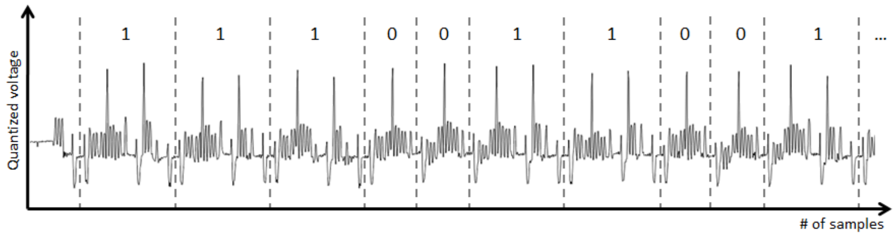
Algorithm 1 Left-to-right binary algorithm for ECC point multiplication.

```
Input: Base point $P \in E(F_q)$, scalar $k = (k_{t-1}, \ldots, k_0)_2$
Output: Point $Q = k \cdot P$
  $Q \leftarrow \infty$
  for $i = t - 1$ to 0 do
    $Q \leftarrow 2Q$
    if $k_i == 1$ then
      $Q \leftarrow Q + P$
    end if
  end for
```

curve (secp160r1) using affine coordinates. Our target device is an 8-bit platform extended with two cryptographic co-processors to speed up modular multiplications and modular inversions. A power trace collected when running this algorithm can be seen in Figure 2.5. Although a global inspection of the curve does not reveal much information, zooming into a few loop iterations allows to clearly distinguish the different patterns corresponding to point doubling and point addition. A mapping of these patterns to Algorithm 1 directly yields the value of the scalar k.



(a) Full execution of the left-to-right binary algorithm for ECC point multiplication.



(b) First iterations of the left-to-right binary algorithm for ECC point multiplication.

Figure 2.5: A power trace of an execution of Algorithm 1 on an 8-bit controller with co-processors (a) and zooming into its first iterations (b).

While the effects of the previous attack are devastating from a security point of view, it is nowadays rather uncommon to find such vulnerable cryptographic implementations. Algorithms such as the always-double-and-add [73] or the Montgomery Power Ladder [129] eliminate key-dependent operations in the point multiplication flow of operations. Alternatively, the use of unified formulae for point doubling and point addition [54] can also serve as a protection against an SPA side channel adversary.

A second class of SPA attacks targets information visible in the *amplitude* of certain peaks within a power trace. The adversarial model in this type of attacks is however quite powerful. Not only is he assumed to know which peaks correspond to which operations, but he has also the ability to deduce e.g. the Hamming weight of the processed value within that peak.

An exemplary SPA attack falling in this category is due to Mangard [150]. The attack targets the inputs and the outputs of S-Box lookups during the AES key expansion. If the adversary is able to determine through the peaks in the power traces the Hamming weight of the processed values, he can then use this information to considerably reduce the key search space. Similar attacks against the key schedule of AES [220], Serpent [72] or Camellia [229] have been proposed as well in the literature.

A third class of SPA attacks targets *cycle* de-synchronizations within a set of traces. These techniques can be seen as an advanced form of timing attacks, enabled by the timing granularity inherently provided in measurements. Rather than obtaining aggregated timing leakages at the end of cryptographic operation, an SPA adversary can detect time de-synchronizations within a set of intermediate operations. A potential SPA attack of this type is proposed by Koeune and Quisquater [142]. It targets non-constant time implementations of the `MixColumns` step of the AES.

2.1.3.2 Differential Power Analysis

In contrast to SPA, DPA attacks can be carried out with little knowledge on implementation details or target device characteristics. The core idea behind DPA is to perform a joint statistical processing of multiple power traces in order to retrieve the secret key of the cryptographic device.

A general overview of the steps followed in the execution of a DPA attack is depicted in Figure 2.6. The goal of the adversary is to extract the secret key K from the cryptographic device. It is assumed that the adversary has knowledge (or control) over input plaintexts P_i, which he can send to the device in order to obtain ciphertexts $C_i = E_K(P_i)$. The first step in a DPA attack is the collection of power traces L_i. Each trace results from a cryptographic execution in the target device when given an input P_i, and therefore can be represented as $L_i = f(P_i, K)$, where f represents the leakage function of the device, P_i the plaintext corresponding to that execution, and K the constant secret key of the device. Each power traces is composed of T samples such that $L_i = [l_{i_1}, \ldots, l_{i_T}]$.

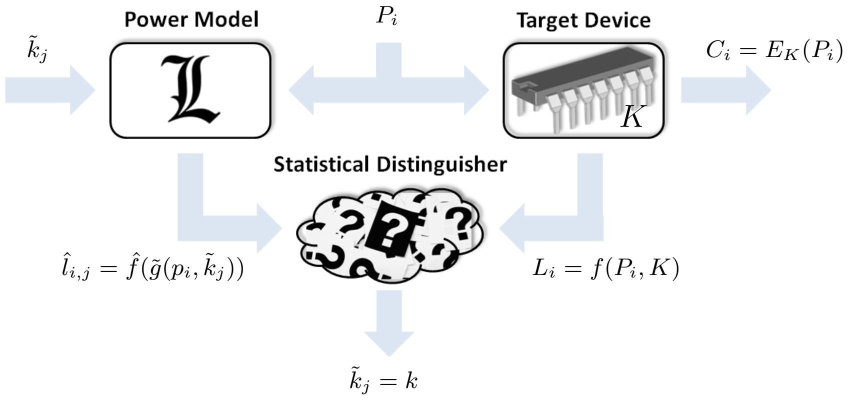


Figure 2.6: General overview of DPA.

After trace collection, the adversary must select an intermediate operation in the cryptographic algorithm. We denote this *target selection* as $\tilde{g}(p_i, k)$, where $\tilde{g}$ is an intermediate computation in the cryptographic algorithm that depends on a part of the plaintext p_i (and potentially other algorithm constants) and a part of the secret key k. In the literature, these intermediate values are often referred to as *sensitive variables*. For the DPA attack to be feasible, it is required that k is enumerable by an adversary, i.e. in order to compute statistics over all possible sub-keys.

One of the critical issues faced by a DPA adversary is the selection of a suitable *power model* to map intermediate values to hypothetical power consumption values. We denote such power model by $\hat{f}$. The adversary applies the power model to the target selection parametrized by the sub-key space. That is, he computes a matrix of hypothesized leakage points $\hat{l}_{i,j} = \hat{f}(\tilde{g}(p_i, \tilde{k}_j))$, where i and j represent the trace number and sub-key hypothesis, respectively. Each column of the matrix is a vector of hypothesized leakage values resulting from applying the target selection to a known plaintext p_i and a potential sub-key $\tilde{k}_j$.

As a final step, the DPA adversary employs a *statistical test* to quantify the dependence between the hypothesized leakage vector of each sub-key $\tilde{k}_j$ and the observed leakages from the target device. Basically, this implies to test each column j of the matrix $\hat{l}_{i,j}$ with the leakage samples of power curves $l_{i,t}$ at each point in time t. Iterating over all T samples is necessary to find the point in time when the device computes the target selection. The hypothesis yielding a stronger dependency is assumed to be the correct sub-key of the target device.

2.1.3.2.1 An Illustrative Attack: Single-bit DPA

The term *single-bit DPA* is commonly used to refer to the DPA attack introduced by Kocher *et al.* [141]. Although the target algorithm in [141] is an implementation of DES, in the following we apply the same attack to break a straightforward implementation of the AES running on an 8-bit microprocessor.

The initial flow of operations in AES is depicted in Figure 2.7. The 128-bit input plaintext P is viewed as an array of 16-bytes, ranging from p_1 to p_{16}. The first operation of the cipher consists in XORing each plaintext byte with one byte of the secret key, yielding a new state array where each byte is of the form $p_i \oplus k_i$. After this, each byte goes independently through a non-linear transformation corresponding to the `SubBytes` operation, which can be seen as a look-up table S. Each byte of the resulting state Y is thus computed as $y_i = S(p_i \oplus k_i)$.

Let us assume the adversary chooses the calculation of the first state byte y_1 as sensitive variable. In other words, he selects $\tilde{g}(p_1, \tilde{k}_1) = S(p_1 \oplus k_1)$ as target selection. This operation fulfills two of the common requirements of suitable DPA targets. First, the output of this operation strongly combines all input bits in a non-linear manner. And second, the sub-key space of this operation is only 8 bits.

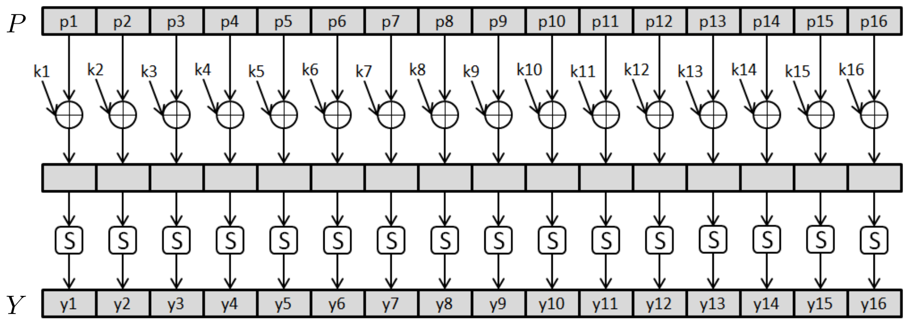


Figure 2.7: Initial flow of operations in AES.

The adversary collects a set of N power measurements $(L_1, \ldots, L_N)$ corresponding to multiple AES encryptions under the same secret key K. Note that these traces contain multiple samples as previously illustrated in Figure 2.3a, and contain one (or multiple) points in time where the target operation $\tilde{g}(p_1, \tilde{k}_1)$ is performed. Iterating over all 2^8 possible values of k_1, allows an adversary to construct a matrix of hypothesized values for the target variable. Each column of the matrix contains the expected values of $\tilde{y}_1$ under a key hypothesis $\tilde{k}_1$.

The next step consists in mapping the hypothesized values to a power consumption model, i.e. to compute the matrix $\hat{l}_{i,j}$ in Figure 2.6. The original DPA attack in [141] is based on the assumption that power samples at the point when an operation is performed depend on the values of the bits being processed. Let us focus on the least significant bit of the hypothesized values of $\tilde{y}_1$, namely b_0. It follows that the power samples due to a variable with $b_0 = 1$ being processed are higher than due to a variable with $b_0 = 0$.

For each key hypothesis $\tilde{k}_1$, the adversary partitions the N power traces in two sets S_0 and S_1 according to whether the least significant bit b_0 of $\tilde{y}_1$ is equal to zero or to one. Once these sets are created for each key hypothesis the adversary proceeds to the evaluation phase. The statistical test employed in single-bit DPA is the Difference of Means (DoM). That is, an adversary *averages* the traces in each set S_0 and S_1 and then computes a *differential trace* by subtracting them. This yields a total of 2^8 differential traces, one for each key hypothesis $\tilde{k}_1$.

The results of the attack are illustrated in Figure 2.8. For the correct key hypothesis, the *average* power consumption of the traces in the set S_1 is expected to be slightly larger that in the set S_0 at the points in time where the bit b_0 is processed. Consequently, the resulting differential trace obtained contains noticeable differences at the points in time where the bit b_0 of $\tilde{y}_1$ is manipulated. In contrast, the partitions of the adversary into the sets S_0 and S_1 for the rest of (incorrect) key hypothesis do not correspond to reality. Averaging each set S_0 and S_1 effectively removes the effect of the bit b_0. As a result, the differential trace does not yield any specific differences.

2.1.3.2.2 On Power Models and Statistical Distinguishers

The choice of power model and statistical distinguisher in DPA attacks is not arbitrary. Quite differently, it represents one of the key factors determining the success probability of an attack. The study and improvement of power models to capture the leakage present in power traces as well as statistical tests to optimally exploit the relationship between model and reality, have been the

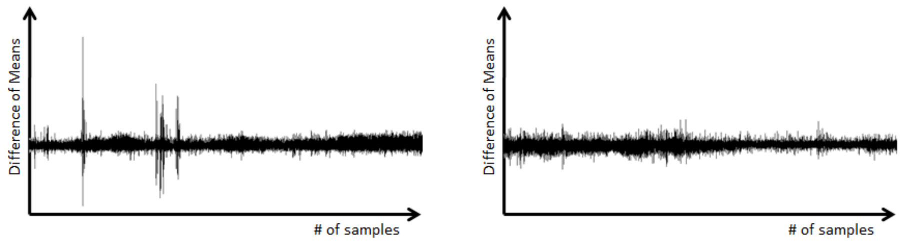


Figure 2.8: Results of single-bit DPA attack on AES for the correct key hypothesis (left) and a wrong key hypothesis (right).

main focus of many works in the field of side channel attacks.

Messerges *et al.* [161] proposed the application of *multi-bit DPA*, in which traces are separated in two sets according to whether all bits in a word are set to zero or to one. While this leads to a better signal-to-noise ratio in the differential traces, i.e. the leakage of *all* bits is exploited as part of the signal, it requires to discard all traces that do not fall into either of these two sets. More efficient attacks, in terms of required power traces, can be devised by selecting a subset of bits rather than the whole word processed.

Because the leakage of digital circuits can be nicely characterized using Hamming weight and Hamming distance models, most follow-up works have focused on such approaches. Assuming a target device with an 8-bit architecture, intermediate variables can be classified in 9 sets ranging from 0 to 8. The generalized multi-bit DPA by Messerges *et al.* [162] proposes to define a certain threshold d allowing to split traces in two groups depending on whether their HW or HD is higher or lower than d.

Brier *et al.* [53] propose Correlation Power Analysis (CPA). This attack builds on the Hamming distance model assuming the power consumption is directly dependent on $HD(X,Y)$, where X and Y correspond to the values of variables manipulated in two consecutive cycles. This leads to a power consumption model of the form $a \cdot HD(X,Y) + b$, where a is a gain scalar for bit transitions and b comprises effects unrelated to the transition. By using Pearson's correlation coefficient as statistical distinguisher, the estimation of the linear relation between model and actual traces is optimally exploited.

Gierlichs *et al.* [108] propose Mutual Information Analysis (MIA). This attack applies an information-theoretic method as distinguisher, namely the mutual information. Because of its ability to detect arbitrary dependencies, this technique allows the application of the most generic power model available: the identity function. Using this power model, one reduces the risk that the attack fails because of incorrect assumptions about the power model, at the cost of a limited increase in computation and data complexity.

2.1.3.3 Collision Attacks

Collision attacks were introduced by Schramm *et al.* [199] in 2003. They exploit the fact that two runs of a cryptographic algorithm with different inputs can yield the same intermediate result at a certain point in time. More formally, two executions with different inputs p_1 and p_2 and same secret key k, can produce an intermediate result $z = f(p_1, k) = f(p_2, k)$, where f is an intermediate operation. Collisions are inherent to cryptographic algorithms with an iterative structure in which the same transformations are repeatedly executed. In this type of algorithms an intermediate value can even collide at multiple points in time during a single encryption process.

In terms of power analysis, a collision can be found when parts of power traces have a high similarity during a time span. The length (and shape) of such interval depends on multiple aspects, e.g. algorithm, implementation, target device, etc., ranging from single cycle operations to a sequence covering multiple cycles. An adversary capable of detecting such patterns can exploit the side channel information provided by the collision.

While the original collision attacks by Schramm *et al.* [199] were targeted to DES implementations, follow-up works by Schramm *et al.* [197] and Bogdanov [49] have studied their application to AES. More recently, an improved collision attack by Moradi *et al.* [164] has been shown to break protected implementations of AES. Witteman *et al.* [227] have applied collision attacks to break protected RSA implementations.

2.1.3.4 Profiled Power Analysis

The techniques described so far attempt key extraction without any (or much) prior knowledge about the side channel leakage of an implementation. Profiled attacks on the other hand make use of a learning step to obtain 'a priori' information about the device's leakage. The better an adversary is able to characterize the leakage of a device, the higher his success probability will be. The profiling stage relies on the assumption that an adversary has access to a device identical (or at least very similar) to the one under attack. In some cases, a further requirement is that the adversary can change secret keys on the training device. While these conditions might be difficult to meet in real scenarios, profiled attacks are among the most powerful side channel techniques.

Template attacks, introduced by Chari et al. [63], have been shown to be the strongest side channel attack possible from an information theoretical point of view, under the assumption that the noise follows a multivariate Gaussian distribution. During profiling, an adversary produces a multivariate characterization of the noise in a measurement consisting of T points. This characterization, referred to as *template*, consists of a mean vector and a covariance matrix (m, C). During the profiling phase, the adversary uses the training device to collect different power traces for different input data P_i and (possibly) keys K_i. Then, he groups the traces corresponding to a certain pair (p_i, k_i) and calculates all corresponding templates $(m, C)_{(p_i,k_i)}$. During the attack stage, the adversary obtains a trace L from the device under attack. He then evaluates the probability density function of the multivariate normal distribution using each template $(m, C)_{(p_i,k_i)}$ and the measurement L. More formally, he computes the following probability:

$$p(L|(m, C) = \frac{1}{\sqrt{(2\pi)^T \cdot det(C)}} \cdot \exp(-\frac{1}{2} \cdot (L - m)' \cdot C^{-1} \cdot (L - m)). \quad (2.1)$$

This probability indicates how much each measurement fits a template, i.e. the higher the probability, the better the match. Once all probabilities are computed, the adversary applies a maximum-likelihood decision rule to find the correct template, which at the same time yields the value of the correct key.

In practice there are several issues that need to be dealt with when using templates. For instance, the size of the covariance matrix C grows quadratically with the number of points in the power trace. For the attack to be computationally feasible it is thus important to reduce the number of points, and only consider those which carry useful information. Finding points of interest (POI) inside power traces is key to successfully carry out a template attack, but unfortunately no optimal mechanism exists that allows to do this. In [63] the sum of pairwise differences of the average measurements is used to highlight and select interesting points, but other alternatives such as the sum of squared pairwise t-differences have been proposed [109]. Rechberger and Oswald [184] use DPA as means to identify suitable points for building templates, while Reparaz *et al.* [185] use the mutual information as criteria for point selection. Other issues often found when dealing with template attacks include ways to efficiently compute Equation 2.1. For instance, it is typical to apply logarithms to avoid the exponentiation, or to use the identity matrix instead of the covariance matrix to avoid computing the matrix inversion.

Stochastic attacks, proposed by Schindler *et al.* [192], model each point in a power curve $L_t(p, k)$ as a stochastic variable composed of two parts such that $L_t(p,k) = h_t(p,k) + R_t$. The data-dependent part $h_t(p,k)$ is a function depending on the input, known data p and subkey k, while R_t is assumed to be a stochastic variable representing zero mean noise. The profiling step is divided in two parts. The first part consists in finding an approximation $\hat{h}_t(p,k)$ of the data-dependent component in a suitable u-dimensional chosen vector subspace $F_{u;t}$, for each instant t. The second part consists in calculating a multivariate density of the noise at relevant time instants. In the attack phase, a maximum likelihood principle is used for key recovery. Stochastic attacks are more efficient than template attacks during the profiling step, but they achieve less precision in the classification step.

2.2 Fault Attacks

In the context of embedded systems a *fault attack* is performed by an *active* adversary in physical control of a cryptographic device. It consists of two closely related phases. First, the deliberate induction of a malfunction during the computation of a cryptographic algorithm; and second, the exploitation of the faulty outcome (or a set thereof) to gain information about secret cryptographic keys. The former phase deals mostly with practice-oriented aspects on *fault injection* mechanisms, whereas the latter follows a more theoretical approach by devising specific cryptanalytic *fault analysis* techniques.

Securing cryptographic devices and implementations against fault attacks is an active research area in the field of embedded cryptography, with multiple overview publications and dissertations [131, 221, 130], one extremely focused and dedicated yearly workshop (FDTC), and a first book completely devoted to the topic due to Joye and Tunstall [128]. Considered one of the major threats by the smart card industry, fault attacks are nowadays an essential step in security evaluation and testing of security-related products.

This section introduces the most relevant works in the area of fault attacks. We start by enumerating existing fault analysis techniques that have been proposed in the literature to attack implementations of several cryptosystems. Then, we review techniques for fault injection and their observed effects on integrated circuits. Here we present our contributions to the study of setup time violations on pipelined microcontrollers. Finally, we briefly describe some of the most common countermeasures against fault attacks at different design abstraction layers.

2.2.1 Fault Analysis

The public research on fault analysis was started by the seminal work of Boneh, DeMillo and Lipton [50] in 1996. Their proposal, often referred to as Bellcore attack, targets implementations of RSA that use the Chinese Remainder Theorem (RSA-CRT) [181], e.g. for signature generation. A straightforward RSA implementation computes an RSA signature as a single group exponentiation $s = m^d \mod n$, where m is the input message, d the private key exponent, and $n = pq$ the RSA modulus resulting from the multiplication of two prime numbers p and q. In contrast, RSA-CRT takes advantage of p and q being known to the device and breaks the computational effort as follows:

$$\begin{aligned} m_p &= m \bmod p, \\ m_q &= m \bmod q, \\ s_p &= m_p^{d_P} \bmod p, \\ s_q &= m_q^{d_Q} \bmod q, \\ s &= (((s_q - s_p) \cdot p_{inv}) \bmod q) \cdot p + s_p, \end{aligned}$$

where d_P, d_Q and p_{inv} are precomputed parameters.

The Bellcore attack assumes an active adversary able to corrupt the intermediate computation on either of the two CRT-branches, i.e. during the computation of s_p or s_q. The fault model considered is rather generic, in the sense that the only constraint on the attacker's corruption is to inject *any* fault in s_p or s_q leading to $\tilde{s} \neq s$. Boneh *et al.* showed that knowledge of both s and $\tilde{s}$ allows to directly factorize the RSA modulus by computing $\gcd(s - \tilde{s}, n)$.

Although this seminal work did not report practical experiments, the devastating effects of the attack quickly caught the interest of the cryptographic community and triggered the appearance of a new field of research. Comparison between fault analysis attacks is often difficult, not only due to differences in the necessary amount of faults and/or off-line computation requirements, but also on the assumptions that are placed on the fault effects in the circuit, i.e. the so-called *fault model.*

2.2.1.1 Fault Models

Roughly speaking, a fault model can be defined as the mathematical representation of a fault injection's outcome. It comprises a series of characteristics that determine which variables can be corrupted and the effects of the fault on the processed data [131]

- The **number** of affected bits determines the granularity of the fault. The most fine-grained attack considers *single bit* manipulations, while more coarse ones assume the disturbance of a *byte* or a *word.* Depending on the assumed implementation platform, word can refer to 8, 16, 32, or any number of bits.
- The **modification** of affected bits specifies the observable outcome of the fault. A *stuck-at* fault implies all targeted bits can be set to a fixed value. The typical assumption is that bits can be set to 0 or 1, giving rise to the so-called *stuck-at-zero* and *stuck-at-one* models. A *flip* fault results in the targeted bits being complemented. Finally, a *random* fault over n bits results in a uniformly distributed value between 0 and $2^n - 1$.
- The **control** of the fault determines the capabilities in inducing an error at a certain location and at a certain time. An adversary with *precise* control is able to affect specific bits within variables at accurate points in time. An adversary with *loose* control on the other hand can target specific variables as a whole, but over a set of operations or clock cycles. Finally, an adversary with *no control* will affect an arbitrary variable at any random point in time.
- The **duration** of the fault refers to the life-span of the induced error. Most faults are considered to be *transient*, meaning that their effect is only temporary. In contrast, *permanent* faults have a fixed effect and cannot be reversed.

Some fault analysis techniques assume a rather relaxed fault model. This is the case for instance of the Bellcore attack, which simply requires a transient corruption within a large interval of time. Other attacks however require extreme levels of precision that are sometimes difficult to argue in practical settings.

2.2.1.2 Fault Analysis of Public-Key Cryptosystems

The security of public-key cryptosystems is based on the intractability of certain mathematical problems. RSA for instance is based on the hardness of factoring very large integers. This is illustrated by the fact that adversaries knowing an RSA public modulus n are unable to find its two prime factors p and q.

Most fault analysis techniques on public-key cryptosystems are based on injecting a fault that results in a “shortcut” to solve its underlying mathematical problem.

A perfect example of such approach is the Bellcore attack: corrupting one of the branches of an RSA-CRT implementation allows an adversary to factor n given a pair of correct and faulty signatures $(s, \tilde{s})$. The same outcome can be achieved by using only one faulty signature $\tilde{s}$ and computing $\gcd(\tilde{s}^e - m, n)$ [125], where e corresponds to the public RSA key.

The first fault analysis attack against straightforward implementations of RSA was already proposed by Boneh *et al.* [50] and it requires an adversary capable of flipping a bit of the RSA private key during binary exponentiation. Similar attacks targeting private exponents are proposed in [29, 127]. Injection of faults on the public modulus can lead to forged signatures being accepted by an RSA verification process [200], and also to the recovery of the private key [52, 39, 37].

Similarly to RSA, fault analysis attacks against ECC often seek to inject faults resulting in the exposure of mathematical vulnerabilities. As the security of ECC cryptosystems depends on the choice of elliptic curve E, a natural adversarial approach is to induce faults such that computations are moved to a different, often weaker, curve E'. This can be done by corrupting parameters such as the base point, intermediate points, or curve parameters [40, 67, 98]. An alternative consists in exploiting faulty points on the original curve E, as for instance the sign change fault attack in [46].

Safe-Error Analysis (SEA). Originally introduced by Yen and Joye [230], safe-error attacks represent a particular branch of fault analysis techniques that do not require the analysis of faulty ciphertexts. The idea behind this powerful class of attacks is to observe whether an injected fault results in an erroneous computation. Because the relevance of some operations in certain public-key algorithms depends on bit values of the key, knowing whether a fault is able to induce an error suffices to gain information about cryptographic keys.

Safe-error attacks are often classified in two categories: C-type attacks [233] and M-type attacks [230, 135]. The former relies on the ability of the adversary to inject *computational* faults into the ALU of a device, whereas the latter targets *memory* locations or registers used by the implementation. Attacks of C-type are particularly devastating in implementations that employ dummy operations, as for instance the always-double-and-add algorithm for ECC point multiplication. Originally introduced to thwart SPA attacks, this algorithm performs a dummy point addition when the scalar bit is equal to zero. A computational fault resulting in the corruption of this operation clearly has no effect on the overall result of the point multiplication. Yet from an attacker perspective, knowing this information reveals the actual value of the scalar bit that is processed in this iteration [233]. Attacks of M-type on the other hand target memory locations and/or registers used by an implementation. Their core idea consists in corrupting a memory element and observe whether the error is propagated to the output or cleared during the execution of the algorithm, e.g. due to the memory location being overwritten with a new value. Similar to C-type attacks, this information suffices to learn the value of the key bit in that particular iteration.

2.2.1.3 Fault Analysis of Symmetric-Key Cryptosystems

Differential Fault Analysis (DFA). Not long after the publication of the Bellcore attack, Biham and Shamir [42] introduced the first fault analysis attack against symmetric-key implementations. The principle behind this technique, named Differential Fault Analysis, is to exploit differences between faulty and correct ciphertexts in order to gain information about the device's secret key. The core of the attack is similar to classical differential cryptanalysis [41], in which an adversary constructs and solves a series of differential equations to discriminate key guesses. The main difference is that while differential cryptanalysis relies on finding appropriate combinations of plaintexts/ciphertexts, DFA employs faults to obtain combinations of correct and faulty ciphertexts.

The original attack in [42] was targeted at implementations of DES, and assumed an adversary capable of flipping one bit of the right half of the DES state during one of the last rounds of the algorithm. Subsequent proposals to attack middle rounds of DES - considering several fault models resulting in different attack complexities - were first proposed in [11] and generalized in [187].

Following the selection of Rijndael [78] as the AES, the research focus on DFA attacks naturally shifted from DES to AES. Piret and Quisquater [172] introduced a powerful attack based on the corruption of one byte in the last rounds. This attack, applicable to other block ciphers using Substitution-Permutation Networks (SPN) structures, required only a few correct and faulty ciphertexts while assuming a rather generic random byte fault model. Following this work, a myriad of DFA attacks on AES have been presented [91, 47, 111, 217].

Collision Fault Analysis (CFA). A variant of DFA suitable to attack early rounds of block ciphers is the so-called Collision Fault Analysis due to Hemme [121]. Originally targeted to DES and 3-DES implementations, CFA assumes a stuck-at fault model, i.e. the result of some computation is set to a certain value known to the attacker, typically zero. The first step in a CFA attack is to obtain a faulty ciphertext corresponding to some arbitrary input value. Assuming a chosen plaintext scenario, the adversary then iterates over a part of the input space, for instance a byte, until a (fault-free) output collision is found. The plaintexts of the collision pair can be then analyzed to infer information about the first round key.

Ineffective Fault Analysis (IFA). An alternative to the previous techniques that does not require analysis of the faulty outputs is given by Ineffective Fault Analysis [47, 69]. Its core idea resembles that of safe-error attacks, albeit with two main differences: it applies to symmetric-key cryptosystems and a stuck-at fault model is assumed. Given a correct plaintext/ciphertext, the goal of IFA is to inject a fault at some point of the execution that has no effect on the ciphertext. This implies that the stuck-at value caused by the fault corresponds to the actual value of the inner computation. Thanks to this, the adversary is able to learn intermediate values of the computation, which can yield information about the key.

2.2.2 Fault Injection

During the late 70s, researchers from the aerospace industry became aware that cosmic rays, i.e. high-energy α-particles particularly found in outer space, were able to inflict damage on semiconductor electronics [234]. The particular effect these particles had on RAM memories was simulated by characterizing them as randomly induced errors in memory cells. This allowed to determine their overall effect at system level, and subsequently look for ways to protect aerospace equipment against this phenomena. Even though this research was not carried out in the context of cryptographic implementations, it is often cited as one of the first observations that faults can be injected on integrated circuits.

Nowadays, several techniques exist to induce errors in electronic devices are known. Depending on their level of intrusion, they are often classified into non-invasive or (semi-) and invasive mechanisms. The former is based on tampering with the interfaces and/or close environment of the device, while the latter requires to expose internal parts of the chip before tampering. In the following we enumerate the most common techniques reported in the literature as well as their effects on cryptographic implementations.

2.2.2.1 Non-Invasive Mechanisms

Clock manipulations. The first successful application of faults to overcome the security provided by cryptographic devices predates the publication of the Bellcore attack. In fact, it was originally deployed by the US pay TV hacking community in the beginning of the nineties [15, 16]. The early generation of pay

TV systems based on smart cards had security flaws that were exploited by the hacking community to reprogram these tokens and obtain unlimited channel subscriptions. In an attempt to disable all hacked devices, the pay TV service provider broadcast a firmware update that resulted in tampered cards entering an infinite loop on power up, rendering them useless. The reply of the hacking community was the design of a piece of hardware - known as *unlooper* - that allowed the cards to regain their functionality [205].

The secret recipe behind *unloopers* was in fact a fault injection whose outcome was the skipping of certain firmware instructions, in this particular case, the branch command within the infinite loop construction. The fault mechanism employed was the insertion of *glitches* in the clock supplied to the smart card. A glitch is a temporary increase of the nominal frequency given to a device. Because the updating of the program counter in a branch instruction is often set late in the clock cycle, the insertion of clock glitches resulted in such operation being skipped.

Due to its role as pioneering attack, the potential effects of clock glitches on software implementations in embedded CPUs were rapidly explored by earlier works [143]. Potential attack scenarios include for instance its usage to extend the runtime of loops in communication routines to see more of the memory after the output buffer [15], to reduce the number of loop iterations in cryptographic contexts, e.g. to convert a secure iterated block cipher into weaker single-round variant [16], and to inject any other fault. In hardware implementations clock glitches have been successfully applied to break implementations of AES [10] as well as other standardized block ciphers [99].

Voltage manipulations. In parallel to these attacks, manipulations on the power line have also been extensively studied. The insertion of voltage *spikes*, i.e. short but high variations of voltage levels, has been reported to result in the skipping of the instructions or in the modification of the data manipulated by the processor [30]. This has been exploited to reduce the number of rounds of an AES implementation on a PIC16F877 smart card [66], and to skip subroutine calls in RSA implementations based on square and multiply [194] and on the Chinese Remainder Theorem [134] on AVR processors. Attacks based on constantly *under-powering* target devices have been studied on hardware implementations of AES [201] and on 32-bit ARM processors [31].

Temperature variations. Altering the environmental conditions around the target device is another possibility for non-invasive fault adversaries. Global increases of temperature have been used to reproduce the Bellcore attack on an 8-bit controller executing RSA-CRT [123] and to induce multiple bit errors in DRAM memory modules [116]. Low-temperature attacks on the other hand enable *cold boot* attacks [120] by freezing data stored in memory cells for later recovery.

Electromagnetic pulses. The effect of injecting faults via the EM channel has also been explored in the literature. Quisquater and Samyde [183] employ a camera flash-gun to inject a high voltage into the coil of an active probe. The resulting magnetic field generates an eddy current on the surface of the chip, which leads to memory errors. Schmidt and Hutter [195] make use of a spark-gap transmitter to generate a strong electromagnetic burst and radiation, and apply this technique to break an RSA-CRT implementation on an 8-bit microcontroller. More recently, Dehbaoui *et al.* [84] investigate the injection of localized electromagnetic pulses on hardware and software implementations of the AES.

2.2.2.2 Semi-Invasive and Invasive Mechanisms

Optical attacks. As electronic circuits are intrinsically sensitive to light, brief and intense optical pulses can result in transistors having a faulty behavior. The idea to exploit this characteristic for fault injection purposes is due to Skorobogatov and Anderson [204]. The *white light* emitted by a low-cost camera flash, focused through a precision microscope, was shown to cause switching in memory cells, even single bit flips. This attack required a semi-invasive intrusion on the device based on the removal of its package.

An enhancement of optical injection mechanisms consists in the usage of a *laser* beam. Its precise nature allows the injection of localized faults targeting particular areas of the exposed chip, e.g. SRAM cells, registers, buses, etc. Exemplary attacks against cryptographic implementations using this fault injection mechanism include for instance [9, 219].

Other attacks. The most accurate fault injection technique requires the use of a Focused Ion Beam (FIB). The precision of this equipment allows attackers to modify the structure of circuits or even cut existing wires, resulting in devastating effects for physical security mechanisms. Despite being a powerful attack tool, the high price and required specialized training makes FIBs a threat only when considering adversaries with large budget.

2.2.3 Fault Attacks based on Setup Time Violations

Most digital circuits today are characterized by having information storage and processing *synchronized* to one (or more) global signal: the clock. An abstraction of the mode of operation of these circuits is given in Figure 2.9. It consists of two D-type flip-flop registers (FF) controlled by a common clock signal (CLK). Data traveling between these registers is processed and modified by a combinational logic block. At the rising edge of the clock, each of the flip-flops captures the value at its input D and makes it available at its output Q.

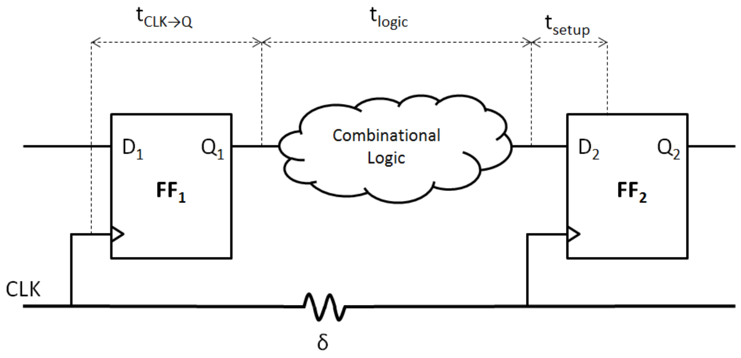


Figure 2.9: Timing Constraints in Synchronous Logic.

The nominal period T_n of this circuit corresponds to its maximal operating frequency, and it is determined by the time required to propagate data between the two flip-flops. A careful study of the delays in the circuit yields the following restriction:

$$T_n > t_{CLK\rightarrow Q} + t_{logic} + t_{setup} - \delta, \tag{2.2}$$

where $t_{CLK\rightarrow Q}$ is the propagation time of the flip-flop necessary to make the signal available at the output Q; t_{logic} is the delay to propagate the signal through all gates, multiplexers, latches, etc. conforming the combinatorial logic; and t_{setup} is the time margin within which data must be held steady at the input D. The parameter δ denotes the skew due to spatial variations in the CLK line.

Non-invasive fault injection techniques often seek to violate the *synchrony* condition of integrated circuits given in Equation 2.2. Glitches or bursts in the clock line result in temporary decreases of the clock period T_n, while low-voltage attacks and temperature manipulations result in an increase of the data propagation delays t_{logic}. More recently, the effects observed by injecting localized electromagnetic pulses have also been ascertained to setup time violations [84].

2.2.3.1 Characterization of Clock Glitches on 8-bit Pipelined Microcontrollers

Publication data

J. Balasch, B. Gierlichs, and I. Verbauwhede, "An in-Depth and Black-Box Characterization of the Effects of Clock Glitches on 8-bit MCUs", In *Fault Diagnosis and Tolerance in Cryptography - FDTC 2011*, pages 105-114. IEEE, 2011.

Contribution

Main author.

In the following we present our work on the study of the effects of setup time violations on smart card microcontrollers. While there exist a lot of works in the literature exploiting this type of faults, most of them target custom hardware implementations of cryptographic algorithms [201, 99, 10]. In the context of software-based implementations, the potential effects of setup time violations have been often enumerated in the literature [15, 16, 30], but without providing technical details on how to perform actual attacks or on the type of CPU architectures considered. Some follow-up works have successfully reproduced some of these attacks on cryptographic implementations running on PIC [66] and on AVR controllers [134, 194], but their focus is on fault exploitation rather than characterizing their effect.

Our original work aims to fill this gap in the literature by providing a complete study and characterization of setup time violations on a legacy smart card architecture. Rather than focusing on the exploitation of fault injection, our research is driven to answer questions such as how, when, and under which circumstances setup time violations result in computation faults. We select a commercial 8-bit AVR controller operating with a two-stage pipeline, and we provide a complete study of how *clock glitches* affect its normal functioning. Our experiments are carried out in a black-box setting, without advanced debugging tools or complete knowledge of the device's internal workings. Yet we are able to put forward a more concrete foundation for future work in setup time violation attacks by highlighting which type of faulty behaviour can be expected and/or exploited as a result of these techniques.

In the rest of this section we provide a description of our experimental setup and our target platform, and we introduce our testing framework. Then, we describe and characterize the effects of fault injection on the program flow and on the data flow of an illustrative software application. We finish by summarizing our findings and by enumerating potential adversarial exploitations of the observed faults.

2.2.3.1.1 Experimental Setup

The experimental setup used in this work is depicted in Figure 2.10. We have implemented a custom ISO/IEC 7816-3 compliant smart card reader with a fully controllable clock signal using a Virtex-II Pro XC2VP30 FPGA. The interface with the smart card at the link and physical layers follows the ISO/IEC 7816-3 standard, while the communication is performed via Application Protocol Data Units (APDUs) as specified in ISO/IEC 7816-4. The computer, acting as a user interface, communicates with the FPGA via an RS-232 interface.

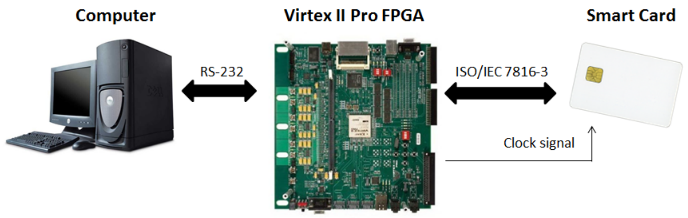


Figure 2.10: Experimental Setup

The FPGA behaves as an off-the-shelf smart card reader. The clock signal provided to the smart card has a fixed nominal frequency in accordance to the controller's specifications. The APDU commands exchanged between the computer and the smart card during communication are simply forwarded by the FPGA to the end receiver. Note that the T0 and T1 protocols specified in ISO/IEC 7816-3 are based on request/response commands, i.e. the computer is always the device that triggers an action by the smart card.

Glitch generation. The effect of injecting a glitch in the clock signal is depicted in Figure 2.11. We denote the nominal period of the clock signal as T_n and the period (or duration) of a glitch as T_g. The idea of injecting a glitch is to temporarily overclock the smart card, i.e. to insert a clock period such that $T_g << T_n$ that potentially causes a transient malfunction of the microcontroller. Notice that after injecting a glitch, the following clock period is reduced from T_n to $T_n - T_g$ in our setup. However, given that $T_g << T_n$, this "post-glitch" period does not affect the normal behavior of the microcontroller.

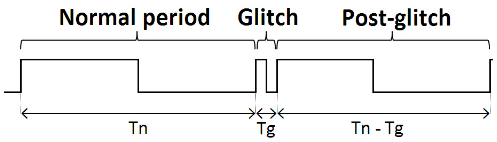


Figure 2.11: Injection of a glitch in the clock signal

We have developed two different mechanisms to introduce glitches in the clock signal. In the first mechanism, illustrated in Figure 2.12, the FPGA generates the *output CLK* fed to the smart card using a combination of two reference signals denoted as *nominal CLK* (with period T_n) and *high-freq. CLK* (with period T_g). Glitches in the frequency of *output CLK* are injected when indicated by the *selection* signal. For the sake of reproducibility, both *nominal CLK* and *high-freq. CLK* signals have to be perfectly phase-aligned. This can be easily achieved by generating the *nominal CLK* signal from the *high-freq. CLK* signal, which in turn determines the granularity of the glitch width. For instance, by fixing the frequency of the *nominal CLK* to 1 MHz (such that T_n is 1μs), the possible frequency values of the *high-freq. CLK* are tied to be multiples of 2 MHz. In other words, the set of possible glitch periods is given by T_g (in μs) $= 1/2i$, where $i = 1, 2, 3, ...$ The accuracy of the glitch period has a standard deviation of 60 ps.

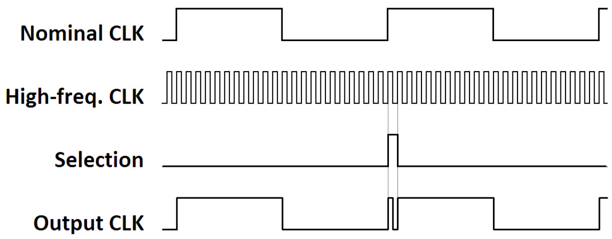


Figure 2.12: Glitch generation using high-frequency signal.

In the second mechanism, shown in Figure 2.13, we use a similar approach as in [10]. In this case, a glitch in *output CLK* is generated by switching between three signals with the same period T_n but with different phases. The advantage of this mechanism with respect to the first one is that it provides more granularity in the glitch period, in particular for low frequencies. We can increase the glitch period in steps of approximately 1 ns such that the set of possible glitch periods is given by T_g (in ns) $= i$, where $i = 1, 2, 3, ...$ The standard deviation of the glitch period for this mechanism is 70 ps.

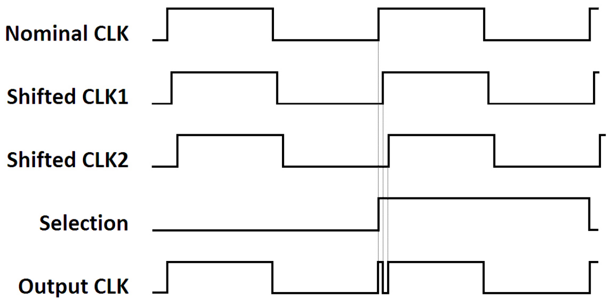


Figure 2.13: Glitch generation using phase-shifted signals.

As a final comment, note that the *selection* signal in these mechanisms allows a wide range of glitch injection patterns, which are by no means restricted to one glitch per trial execution. The selection of all parameters involved in the generation of the output clock signal (e.g. glitch mechanism, nominal period T_n, glitch period T_g, glitch position, etc.), is completely configurable by the user via commands send from the PC to the FPGA, thus obtaining a highly-flexible yet automatized experimental setup to carry out our study.

2.2.3.1.2 Target Platform.

We choose as target platform a microcontroller belonging to the 8-bit Atmel AVR family, namely an ATMega163 microcontroller in smart card packaging. There are several reasons for this choice. First, and most important, this device operates on an external clock signal, such that it is possible to inject faults to the device using this interface. Second, the characterization of the effects of fault injection on AVR microcontrollers is a challenging task: AVR controllers have a modified Harvard architecture, i.e. although access to program code (flash memory) and data (internal RAM) is physically separated in the chip (strict Hardvard architecture), the CPU can concurrently use both buses in a clock cycle. This characteristic, combined with a RISC architecture with most of the instructions executing in a single-cycle, allows to obtain a two-stage pipeline: while one instruction is being executed, the next one is pre-fetched from program memory as shown in Figure 2.14. Consequently, fault injections can have multiple and complex effects. Third, these devices are available in smart card form, making them a good representative of security devices. Finally, Atmel AVR controllers are known devices largely used in the related literature, not only in fault analysis but also in side channel attacks.

Prior to our work, no study has been made to fully characterize and understand

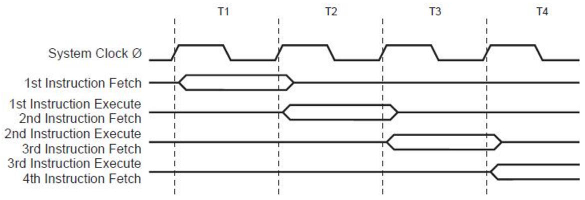


Figure 2.14: Pipeline in AVR controllers (source: ATMega163 datasheet).

the reaction of these devices to fault injection via clock glitches. The following sections fill this gap. Note that although Atmel offers a family of AVR microcontrollers specifically designed for security applications, the smart cards used in our tests have no security claims whatsoever. Our research motivation is not to evaluate the level of resistance of such microcontrollers to fault attacks, but rather to understand and characterize the effects of fault injection via clock glitches on one model of the low-cost family. Note that the analysis is done in a black-box setting, i.e. we only have access to the publicly available data sheets.

2.2.3.1.3 Testing Framework

The approach followed in our experiments consists in decreasing the glitch period T_g, starting with a value such as 125 ns (or 8 MHz) for which the microcontroller functions correctly, until 15 ns. This lower bound is determined by the switching speed of the FPGA board's I/O pins as well as some external analogue circuitry of our experimental setup. When faults start occurring, we analyze them in order to be able to characterize the chip's behaviour. Our experiments show that the critical path (i.e. maximum frequency tolerated by the microcontroller) is determined by the access to Program Memory. In other words, the first effect noticed when decreasing the glitch period is an erroneous behaviour of the pre-fetching stage.

In the following we will make a distinction between which pipeline stage is affected by the glitch. We will begin by focusing on the effects of clock glitches on the pre-fetching stage, analysing how it is possible to inject faults such that the *program flow* is altered. After this, we will focus on the effects of the glitches on the execution stage, studying how the expected *data flow* of a program is changed. In our experiments we have used five ATMega163 smart cards to verify that they all respond to fault injection in a very similar way.

We have implemented several test applications in assembly language and executed them a large number of times in order to obtain and analyze the effects of clock glitches. In order to make the interpretation of the results more clear, we provide some exemplary code fragments. Although such tests do not correspond to any particular cryptographic implementation, the results obtained can be easily extrapolated to the general case.

Finally, note that by targeting a device with a two-stage pipeline and without access to details of the inner workings of the microcontroller, the analysis of the faults' outcome becomes an arduous task. The only information available for the interpretation of the faults's effects consists of an array of output data. Before running a test, we bring the microcontroller to a state $\mathcal{A}$ such that all possible variables (RAM values, program memory, registers, flags, and even room temperature) are fixed and known. A normal execution of the test application brings the microcontroller to an "expected" state $\mathcal{B}$, whereas a faulty execution brings it to an "incorrect" state $\mathcal{B}'$. Manually reverse-engineering the chain of events that explains the transition from state $\mathcal{B}$ to state $\mathcal{B}'$ is far from trivial.

2.2.3.1.4 Effects of Clock Glitches on Program Flow

The AVR instruction set consists of 130 commands, most of them executing within a single clock cycle. Instruction opcodes are typically encoded and stored in Program Memory in 16-bit words. Although being an 8-bit device, the AVR microcontroller has a 16-bit Program Bus. This means that in the pre-fetching phase, the 16-bit opcode pointed to by the Program Counter (`PC`) is loaded at once. In turn, the `PC` is also incremented in this stage, such that the next opcode is correctly loaded in the following clock cycle. The behaviour of multi-cycle instructions differs from that of single-cycle instructions; these differences will be discussed later.

`NOP`: No Operation

We start our analysis by testing the effects on the most simple command available, namely `NOP`. As this instruction does not perform any operation in the execution phase, glitches will only affect the pre-fetching stage. Our first test is depicted in Figure 2.15, where `Inst` refers to *any* of the available AVR instructions. By injecting a glitch in clock cycle i (when `NOP` is being executed), one can possibly cause an erroneous behaviour in the pre-fetching phase.

Cycle	**Instruction**	**Opcode (bin)**
i	`NOP`	0000 0000 0000 0000
$i+1$	`Inst`	-

Figure 2.15: Code example for `NOP` (I).

For testing purposes, let us assume that `Inst` is the command `EOR R15,R5` (Exclusive OR) as illustrated in the top part of Figure 2.16. When injecting a glitch with period smaller than or equal to 59 ns in clock cycle *i*, we observe that `EOR R15,R5` is never executed. Intuitively, one can assume that the microcontroller does not have time to load the next command from Program Memory as consequence of the glitch. So a reasonable explanation is that the opcode being executed at the time of the glitch (e.g. `NOP` in cycle *i*) is executed again in cycle *i+1*, as shown in the lower part of Figure 2.16. Note however, that the `PC` is not affected by the glitch and is correctly incremented. Otherwise, the microcontroller would simply pre-fetch the command `EOR R15,R5` in cycle *i+1* and execute it in cycle *i+2*.

Glitch period	**Cycle**	**Instruction**	**Opcode (bin)**
-	*i*	`NOP`	0000 0000 0000 0000
-	*i+1*	`EOR R15,R5`	0010 0100 1111 0101
$\leq$ 59 ns	*i+1*	`NOP`	0000 0000 0000 0000

Figure 2.16: Code example for `NOP` (II).

Suppose now that `Inst` is the command `SER R18` (Set Bits in Register) as illustrated in Figure 2.17. In this case, we observe that depending on the glitch period the command `SER R18` is substituted by instructions other than `NOP`. In particular, for a glitch period equal to 61 ns the command `LDI R18,0xEF` (Load Immediate to Register) is executed. Decreasing the glitch period to 60 ns produces the appearance of the command `SBC R12,R15` (Subtract with Carry). Finally, for any glitch period smaller than or equal to 59 ns, we observe the same effect as shown in Figure 2.16, namely, `NOP` is executed.

Glitch period	**Cycle**	**Instruction**	**Opcode (bin)**
-	*i*	`NOP`	0000 0000 0000 0000
-	*i+1*	`SER R18`	1110 1111 0010 1111
$\leq$ 61 ns	*i+1*	`LDI R18,0xEF`	1110 1110 0010 1111
$\leq$ 60 ns	*i+1*	`SBC R12,R15`	0000 1000 0010 1111
$\leq$ 59 ns	*i+1*	`NOP`	0000 0000 0000 0000

Figure 2.17: Code example for `NOP` (III).

These results shown in Figure 2.17 illustrate the transition in which the microcontroller internally updates the opcode to be executed. As one can notice, there is a progression from the expected command (`SER R18`) to the previous command (`NOP`), in the sense that more bits of the erroneous opcodes are degraded to zero as the glitch period decreases. Strictly speaking, at this point it is not fully correct to describe the effect of the glitch as *skipping* an instruction; rather differently, as another command is executed instead of the expected instruction, a more accurate description of the glitch effect would be *replacing* an instruction.

Note that when `LDI` or `SBC` are executed instead of `SER`, some registers are overwritten with the values resulting from the execution of such commands; thus, a single fault injection disrupts at the same time both the program flow and the data flow.

The effects depicted in Figure 2.17 are observed in all five ATMega163 smart cards, although there are some slight differences. First, the glitch periods for which instructions are replaced can vary from card to card. And second, it is possible that instructions different than those in Figure 2.17 appear in cycle *i+1*. We have however verified that for glitch widths smaller than approximately 52 ns a `NOP` is always effectively executed in all cards.

A particularly interesting case in the test is observed when the command `Inst` has a 32-bit opcode. Consider, as shown in Figure 2.18, that this instruction is `LDS R22,0x0128` (Load Direct From Data Space). As the program bus of the AVR is 16-bit wide, `LDS` requires an extra cycle to fetch the second half of the opcode, i.e. the value 0x0128, from Program Memory. By injecting a clock glitch with period 59 ns in cycle *i*, `LDS` is replaced by `NOP` in cycle *i+1*. However, as the skipped command has a 32-bit opcode, the value 0x0128 is pre-fetched from Program Memory in cycle *i+1* and interpreted as a command in cycle *i+2*. As a result, a completely wrong instruction is inserted in the program flow. For this particular example, the instruction corresponding to opcode 0x0128 is `MOVW R4,R16` (Copy Register Word), which moves a 16-bit word from a pair of registers to another pair of registers in a single cycle.

Glitch period	**Cycle**	**Instruction**	**Opcode (bin)**
-	*i*	`NOP`	0000 0000 0000 0000
	i+1	`LDS R22,0x0128`	1001 0001 0110 0000
	i+2		0000 0001 0010 1000
$\leq$ 59 ns	*i+1*	`NOP`	0000 0000 0000 0000
	i+2	`MOVW R4,R16`	0000 0001 0010 1000

Figure 2.18: Code example for `NOP` (IV).

If the second half of the 32-bit opcode is not a valid command, for instance, `LDS R22,0x0060`, the microcontroller will execute the opcode 0x0060 in cycle *i+2* as a consequence of the fault. However, the execution of illegal opcodes in AVR microcontrollers is carried out without affecting the program flow; in fact, they have the same effect as `NOP`s. Note that it is possible that other microcontrollers behave differently when interpreting an invalid opcode, for example, resetting the chip. For the AVR controllers we have used this is however not the case observed.

Branching instructions

For the second set of experiments we target branching instructions. These commands do not have an execution phase that directly affects data; instead, they modify the value of the `PC` according to a tested condition. Consider the code example shown in the top part of Figure 2.19. In cycle *i*, the `TST` command checks whether register *R12* holds a value equal to zero. If so, it sets the Zero flag in the Status Register (`SREG`); otherwise, the flag is cleared. In cycle *i+1*, the `BREQ` (Branch if Equal) command checks the value of the Zero flag: if the flag is set, it modifies the value of the `PC` in order to branch to a different code segment; otherwise, `PC` is incremented such that the next instruction is `SER R26`. The former option requires two cycles to complete, while the latter executes in a single cycle.

If the Zero flag is cleared, `BREQ` simply increments the `PC` in a single cycle, thus behaving similarly to a `NOP`. By injecting a fault in cycle *i+1* one would expect a faulty behaviour such as in the previous experiments. However, as shown in Figure 2.19, the amount of faulty instructions executed instead of `SER R26` and the glitch periods for which errors appear are quite different.

Glitch period	**Cycle**	**Instruction**	**Opcode (bin)**
	i	`TST R12`	0010 0000 1100 1100
-	*i+1*	`BREQ PC+0x02`	1111 0000 0000 1001
	i+2	`SER R26`	1110 1111 1010 1111
$\leq$ 57 ns	*i+2*	`LDI R26,0xEF`	1110 1110 1010 1111
$\leq$ 56 ns	*i+2*	`LDI R26,0xCF`	1110 1100 1010 1111
$\leq$ 52 ns	*i+2*	`LDI R26,0x0F`	1110 0000 1010 1111
$\leq$ 45 ns	*i+2*	`LDI R16,0x09`	1110 0000 0000 1001
$\leq$ 32 ns	*i+2*	`LD R0,Y+0x01`	1000 0000 0000 1001
$\leq$ 28 ns	*i+2*	`LD R0,Y`	1000 0000 0000 1000
$\leq$ 27 ns	*i+2*	`LDI R16,0x09`	1110 0000 0000 1001
$\leq$ 15 ns	*i+2*	`BREQ PC+0x02`	1111 0000 0000 1001

Figure 2.19: Code example for `BREQ`.

In the range 57 ns $\leq T_g \leq$ 28 ns the opcode values describe a clear transition towards zero, i.e. stuck-at-zero pattern. The first erroneous command to be executed (`LDI R26,0xEF`) differs from the expected (`SER R26`) in that bit 8 is zero instead of one; for the second command (`LDI R26,0xCF`) bit 9 is also cleared. This progression is observed until for `LD R0,Y` a total of 11 bits are cleared compared to those of `SER R26`.

By decreasing the glitch period further than 27 ns, one would expect to obtain an erroneous opcode consisting of only zeroes. However, results show that after `LD R0,Y` (with only two bits set to one), there is a transition to `LDI R16,0x09` (with five bits set to one). Finally, for a glitch period equal to 15 ns, the command `BREQ PC+0x02` is executed again instead of `SER R26`.

At this point it is clear that the figure shows a transition from the expected opcode to the previous opcode, most probably going through an intermediate all-zero state. Depending on the period of the glitch injected, the instruction executed in cycle *i+2* is a degraded version of either the expected opcode (`SER R26`) or the previous opcode (`BREQ PC+0x02`). Once again, although the general behaviour observed in other cards is the same, the faulty instructions and the glitch periods for which they are observed might vary with respect to the ones given in Figure 2.19.

Single-cycle instructions

In order to prevent effects on the data flow, skipped instructions should ideally be replaced by either `NOP`s, illegal commands, or even testing commands (such as `TST`). However, the amount of variables that determine the final shape of the faulty opcode, as well as the differences observed for different ATMega163 cards, make it very difficult to characterize and determine the outcome of the fault.

One way to obtain a rough estimation of the probability that faulty commands affecting the data flow appear, consists in running a series of tests similar to the one depicted in Figure 2.20. This code fragment includes only single-cycle instructions, such that a pre-fetching is performed in each clock cycle. By injecting a fault in each cycle of these test programs, one can count how many times the data flow is affected, i.e. how many times the faulty instruction affects internal data variables. We have generated several of these tests, changing parameters such as the order of the instructions and the source/destination registers in order to maximize the variation of the opcodes. The results obtained indicate that around 50% of the time, the injection of a clock glitch with period 57 ns $\leq T_g \leq$ 28 ns results in the insertion of a faulty command whose execution alters the data flow.

Multi-cycle instructions

As mentioned earlier, multi-cycle instructions have a different behaviour than single-cycle commands in the sense that they do not follow exactly the two-stage pipeline as shown in Figure 2.14. Some of these instructions perform the pre-fetching in the last execution cycle. In such cases, it possible to inject faults with a similar effect as observed earlier, i.e. the following instruction is replaced by another instruction. This is the case for commands such as `RET` (Return

Glitch period	Cycle	Instruction	Opcode (bin)
	i	`SUB R7,R5`	0001 1000 0111 0101
	i+1	`ADD R8,R4`	0000 1100 1000 0100
-	*i+2*	`LSL R15`	0000 1100 1111 1111
		. . .	
	i+20	`MOV R27,R6`	0010 1101 1011 0110

Figure 2.20: Code example for single-cycle instructions.

from Subroutine) or branching instructions such as `BREQ` (Branch if Equal).

We have however noticed that not all multi-cycle instructions behave like this. For instance, we have not been able to replace instructions executed after commands such as `LD` (Load From Data Space), `LPM` (Load from Program Memory) or `RCALL` (Relative Call to Subroutine), independently of the glitch width and of the cycle in which the fault is injected.

2.2.3.1.5 Effects of Clock Glitches on Data Flow

Until now we have exclusively focused on how clock glitches affect the program flow of certain commands. However, given the two-stage pipeline of our target microcontroller, it might be possible to affect also the execution stage of certain instructions such that the resulting computation is corrupted. In this section we address the feasibility and the conditions under which these faults occur.

Single-cycle instructions

We have observed that it is possible to affect the execution stage of single-cycle instructions operating on data (e.g. arithmetic, logic, and bit operations). However, there are some issues that make it difficult to characterize the effect of such fault injections. First, the glitch period for which corrupted values start appearing depends on the specific instruction being executed; second, the source and destination registers used as operands also influence the outcome of the fault; and third, the fact that a single fault affects both pipeline stages at the same time makes the interpretation of the fault more complex.

In general, a minimum glitch period of approximately 27 ns is required to affect the execution stage of most instructions. The first effect observed by decreasing the glitch period is that one or more bits of the correct result are stuck-at-zero. For smaller glitches, the result of the operation becomes an erroneous value independent of the correct result or the previous value in the register. This behaviour should however not be taken as a generalization. As mentioned, there are too many parameters that influence the exact outcome of the fault.

An important observation is that the value of the corrupt result does not vary for an arbitrary number of executions using the exact same microcontroller and glitch parameters. In other words, given a state $\mathcal{A}$ and a fixed glitch sequence, the effect of the fault is constant. This observation implies that the result of the fault injection should not be referred to as *random*, but rather *deterministic* for a series of fixed executions.

Multi-cycle instructions

Given the complexity of the results obtained for single-cycle instructions, one might wonder whether it is easier to target the execution stage of multi-cycle instructions. In this section we focus on two of the more relevant such instructions, namely, `LD` and `LPM`.

The `LD` (Load from Data Space) instruction requires two clock cycles to execute. It loads the value pointed to by either register X, Y, or Z into any destination register in the microcontroller. In order to better understand the internal behaviour of this instruction, we have collected and analyzed different groups of power measurements while executing `LD` in the AVR controller. The result is illustrated in Figure 2.21, in which the execution of `LD` corresponds to time samples 80 to 180. The dashed curve indicates the difference of means of two sets of measurements for which a fixed value is loaded from two different memory positions. A peak is noticeable at the rising edge of the second cycle, thus verifying that the RAM address is updated at this time. The solid curve indicates the difference of means of two sets of measurements loading different values from a fixed RAM address. A peak is visible at the falling edge of the second cycle, indicating that the RAM value is loaded in the data bus at this time.

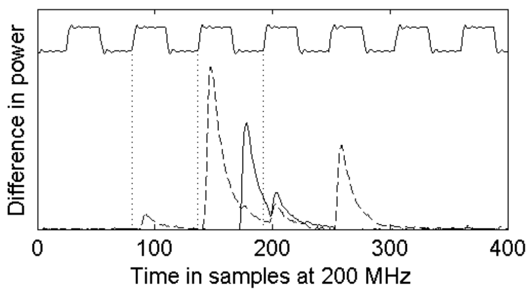


Figure 2.21: Difference of means for different executions of `LD`

In order to evaluate the effects of clock glitches on the `LD` instruction, we run the test in Figure 2.22 for a fixed value in the pointer Z. We ensure that potential effects of the destination register on the outcome of the glitch are also taken into account by considering all possible cases.

Glitch period	**Cycle**	**Instruction**	**Opcode (bin)**
	i	`LD R0, Z`	1000 0000 0000 0000
-	*i+2*	`LD R1, Z`	1000 0000 0001 0000
	...	...	...
	i+50	`LD R25, Z`	1000 0111 1001 0000

Figure 2.22: Code example for `LD`.

For the first cycle, faulty values start to appear for glitch periods smaller than 24 ns. A clock glitch in this position produces an error in the value of the RAM pointer; as a result, a value stored in another RAM address is loaded. We have verified this assumption by filling the RAM space with known values. By repeating the same test multiple times we verify that a large percentage of the erroneous values come from this pre-filled RAM space. We have not been able however to characterize how the pointer is affected by the glitch, i.e. the exact relation between the correct RAM address, the faulty RAM address, and the glitch period.

For the second clock cycle we begin to observe erroneous results when the glitch period is approximately 64 ns. In the low period of this cycle, the RAM value is transferred via the data bus. In AVR platforms the data bus is not pre-charged, meaning that the last value that uses the data bus is kept until overwritten. A glitch injection in the second cycle of `LD` prevents the value in the data bus from being updated. In particular, depending on the glitch period it is possible to prevent one or more bit transitions from happening.

Figure 2.23 shows the number of bit transitions that are avoided in function of the glitch width. The first effect produced by the fault is to prevent the transition of a single bit from 0 to 1. Our experiments show that bit number 4 is first affected. This implies that, considering that the previous value on the data bus is 0x00, if we try to load the value 0xFF while injecting a glitch, the outcome is the value 0xEF being loaded. Note that for the opposite case (previous value in the bus being 0xFF and trying to load 0x00), the glitch does not affect the result. This is because bit transitions are in this case from 1 to 0. As we decrease the width of the glitch more erroneous $0 \rightarrow 1$ transitions appear, until at around 55 ns no such transition is possible anymore.

Focusing on the other bit transition, namely $1 \rightarrow 0$, we observe that the faulty behaviour starts at around 54 ns. The first bit affected in our experiments is once again bit number 4. Similarly to the $0 \rightarrow 1$ transition, more erroneous bit

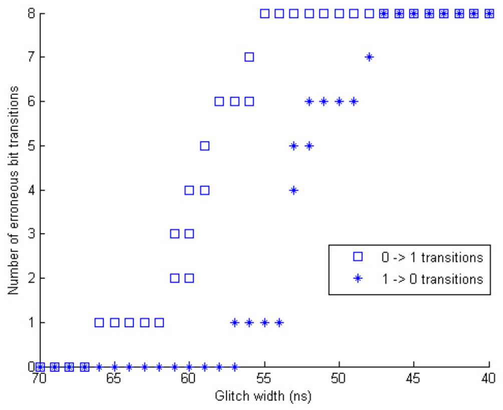


Figure 2.23: Number of erroneous transitions in the data bus when decreasing the glitch width.

transitions appear by decreasing the glitch period. Finally, when T_g is around 46 ns, no bit transitions are possible at all. At this point the effect of the glitch is that the erroneous value being effectively loaded corresponds to the previous value that was on the bus. In other words, if we inject a glitch with period equal to or smaller than 46 ns in cycle *i+3* in Fig. 2.22, the value that ends up in register R1 is the same as was previously loaded into R0.

Once again, although the transition in the data bus is clearly visible in all five ATMega163 cards tested, the results for a particular glitch period might vary. However, we have verified that for glitch widths smaller than or equal to 44 ns no bit transitions in the data bus are possible in any card; also important, the first transition affected corresponds to bit number 4 in all cards.

The `LPM` instruction (Load from Program Memory) requires three clock cycles to execute. It loads the value in Program Memory pointed to by register Z into any of the registers in the microcontroller. In order to evaluate the effect of the clock glitch on the `LPM` instruction, we run the test in Figure 2.24 such that the effects on each destination register are also taken into account.

Glitch period	**Cycle**	**Instruction**	**Opcode (bin)**
-	*i*	`LPM R0, Z`	1001 0000 0000 0100
	i+3	`LPM R1, Z`	1001 0000 0001 0100
	...	...	...
	i+78	`LPM R25, Z`	1001 0001 1001 0100

Figure 2.24: Code example for `LPM`.

By injecting a glitch with period smaller than 25 ns in the first cycle of `LPM` we observe that an erroneous value is loaded into the destination register. In this cycle, the `LPM` instruction sets the address to the program memory from which the value has to be loaded. The glitch alters the value of the address in a way that an erroneous code memory location is pointed to, thus resulting in a wrong value being stored into the register. We tested this hypothesis by filling the free space in Program Memory with known values (e.g. 0x4444, 0x5555, etc.). By repeating the tests multiple times, we observe that a large percentage of the values loaded corresponds to these pre-set constants, thus verifying our hypothesis.

In the second execution cycle of `LPM`, erroneous values start appearing for glitches smaller than or equal to 47 ns. We have observed that these faulty values correspond to one of the bytes of the next opcode in Program Memory. For instance, consider that we inject a glitch in cycle *i+2* in the code example shown in Figure 2.24. The following opcode corresponds to the instruction `LPM R1,Z`, and its value is 0x9014. If the least significant bit of pointer Z is set to one when executing `LPM R0,Z`, then the erroneous value that ends up loaded in *R0* is the upper byte of the next opcode, i.e. 0x90. If the least significant bit of the pointer Z is cleared, then the value that is loaded corresponds to 0x14.

This behaviour can be explained by the fact that, although the Program Bus is 16-bit wide, the value loaded by `LPM` is only 8-bits. Thus the least significant bit of the pointer `Z` is used to determine whether the upper byte or the lower byte is to be loaded into the destination register. By injecting a fault in the second cycle of `LPM`, the address to Program Memory is not updated with the value of pointer `Z`, such that it remains pointing at the following opcode to be loaded. That is the reason why the outcome of the fault consists in loading this particular erroneous value.

2.2.3.1.6 Summary and Applications of the results

Faults in program flow.

We have observed that we can affect the fetching of the next opcode such that it is replaced by another instruction. Accurate timing would ideally allow to fetch an all-zero opcode (`NOP`), i.e. to effectively “skip” a command. However, controlling the opcode transition turns out to be highly complex. So typically, a faulty version of the old or the new opcode with some 1 bits set to 0 will be fetched depending on the glitch timing. In this case it is difficult to control the fault’s effect, since the faulty opcode can represent one of many valid instructions that manipulate data or program flow. Moreover, the transition steps (faulty opcodes executed) vary depending on the smart card used and the specific microcontroller state. Using very short glitch periods, we can potentially prevent the fetching of a new opcode entirely such that the old opcode is executed twice. However, when using such short glitch periods the fault also affects the execution stage of the current instruction. This is in particular problematic if the current opcode is a single-cycle command, as the fault affects both program flow and data flow at the same time. This leads to a more complex fault, possibly affecting data in several registers resp. at several memory addresses. We have estimated that around 50% of the times, a faulty command that does not affect the data flow appears as a replacement for the skipped instruction. So variations on the glitch period might allow to inject faults with potential for exploitation. However, we find it easier to replace instructions without disturbing the data flow for some multi-cycle instructions such as branching or return commands, i.e. when no data is manipulated and the pre-fetching of the new opcode happens in the last execution cycle.

These observed faults in the program flow should allow the typical applications: to skip checks and to prevent counters from increasing or decreasing, e.g. round counters or counters in I/O routines. In addition, new (somewhat trivial yet powerful) attacks can be mounted to attack cryptographic implementations. For illustrative purposes only, we will provide an example using AES [6].

Suppose the four main operations of the cipher (`SubBytes`, `ShiftRows`, `MixColumns` and `AddRoundKey`), and possibly the key schedule, are implemented as subroutines that the main function simply calls as required using `RCALL`. Consider that the `SubBytes` subroutine has been called in the last round and is executing the final `RET` instruction to return to the main function. We can easily inject a fault into the last cycle of the four-cycle `RET` instruction that affects the fetching of the new opcode (it would be the next `RCALL` instruction, here calling ARK for the last time) and replaces it with an invalid opcode or with `NOP`, effectively skipping the subroutine call. An XOR of the obtained faulty ciphertext and the correct ciphertext yields the full last round key. Note that even if the implementation is organized differently, i.e. unrolled instead of using `RCALL`s, the fact that our setup supports the injection of faults into any arbitrary sequence of cycles still allows to perform the attack. For instance, we can then inject a series of consecutive faults into *all* cycles following the last SR operation to effectively skip each and every instruction performing the last ARK. Varying the glitch period might help the attacker in distinguishing whether faulty opcodes changing the data flow have been inserted. We have fully implemented and tested this trivial attack in all five smart cards in several repetitions, thus confirming its feasibility.

Faults in data flow.

We have further observed that it is possible to inject faults into the execution of instructions. For single-cycle instructions, the required glitch period affects also the fetching of the following opcode as explained above, thus leading to an even more complex fault. Such faults can probably be exploited only if not too many cryptographic transformations or non-critical instructions follow. Otherwise, the not well understood fault becomes too complicated to exploit. For multi-cycle instructions however, the glitch does not necessarily affect the fetching of the new opcode. We obtained the best and most stable results for multi-cycle instructions with memory access, e.g. `LD` and `LPM`. Depending on the glitch period, we can rather accurately prevent a given number of bits on the data bus from flipping, although the by far easiest option is to prevent any transition, i.e. to prevent the data bus from updating at all. The effect of such a glitch is that the instruction loads the last value that has been transferred to/from memory.

A typical and straightforward application of the fault results obtained in memory access instructions, again using AES encryption as an example, would be to glitch S-box lookups (typically implemented with `LD` and `LPM` commands) to apply some of the well known DFA, CFA or IFA fault analysis techniques in the literature.

2.3 Countermeasures

Mitigations against physical attacks can be implemented at multiple layers of the design space. In principle, low-level solutions can tackle the origin of the vulnerabilities at its root, and thus represent an ideal option to counter certain type of attacks. These solutions are also advantageous for cryptographic developers, as they allow to focus on further implementation constraints other than security. In practice however, low-level solutions tend to result in large area overheads. As such, their real-world deployment is not always feasible due to the budget limitations faced by hardware manufacturers.

High-level solutions on the other hand can take advantage of the mathematical structures of cryptographic algorithms to deploy effective and reliable countermeasures. An advantage of this approach is that countermeasures can benefit from formal security proofs. The protection level of high-level solutions depends however on a good modeling and understanding of the behaviour of the underlying layers. Consider the case for instance of countermeasures against timing leakages. Intuitively, timing leakages can simply be overcome by ensuring high-level cryptographic computations run in a constant amount of time. However, the application of this straightforward mitigation is rather complex to achieve in practice - particularly for software implementations - due to microarchitectural features present in digital processors. Cache memories in general purpose CPUs [34] and early-terminating multipliers on ARM processors [117], are examples of low-level features that can thwart the security of a supposedly time-constant implementation.

In the following we enumerate several countermeasures often mentioned in the literature to counteract physical attacks. Keeping track of state-of-the-art developments in the arms-race between attacks and countermeasures is often a challenging task. Moreover, a countermeasure crafted against a particular type of attack may unintentionally enable another attack. A survey of this effect for ECC countermeasures is given by Fan and Verbauwhede [95].

2.3.1 Against Side Channel Attacks

Countermeasures against side channel attacks are classified into two main groups: *hiding* and *masking*. Hiding techniques aim to break the direct link between the observations in a power trace and the data processed by the circuit. This can be achieved by manipulating either of the two dimensions of power traces, namely time or amplitude, in such a way that from an attacker's perspective the device seemingly consumes either random or equal amounts of power in each clock cycle. Masking techniques on the other hand aim to break the relation between algorithmic sensitive values and processed values. As opposed to hiding techniques, this is achieved by randomizing the actual processing of sensitive variables within the execution of a cryptographic algorithm.

Low-level protections can implement either of these two approaches. Hiding techniques in the temporal domain can be achieved for instance by adding random effects in the clock line. This includes the skipping of clock pulses, variations in the clock frequency, or the existence of multiple switching clock domains. In the amplitude domain, one often seeks to reduce the Signal to Noise Ratio (SNR) of the side channel leakage. The addition of parallel hardware blocks acting as noise generators or the insertion of filters/regulators in the power line can lead to an increase of the adversarial effort necessary to exploit side channels.

One of the well-studied family of low-level protections aims to tackle the CMOS leakage at its root by employing alternative logic styles with different power consumption behavior. Dual-rail precharge (DRP) for instance encodes each logic signal as two complementary wires, in such a way that circuit transitions are the same for each clock cycle independently of the values on the wires. Sense Amplifier Based Logic (SABL) [211] and Wave Dynamic

Differential Logic (WDDL) [212] are two exemplary proposals in this area of hiding techniques. Masking countermeasures can be also applied within logic styles. Random Switching Logic (RSL) [208], Dual-rail Random Switching Logic (DRSL) [65], and Masked Dual-rail Precharge Logic (MDPL) [175, 174] are the most representative techniques that have been proposed in the literature.

High-level mitigations based on hiding are implemented in the time domain. One possible technique consists in the addition of random delays within the execution of a cryptographic algorithm [70]. Alternatively, a similar effect can be achieved by shuffling the order of certain operations [223]. Techniques based on masking are are perhaps one of the most well-studied and interesting techniques to counter side channel attacks at algorithm level. The state-of-the-art review including our contributions to this area of countermeasures will be covered in Chapter 4.

Finally, a natural countermeasure that can be implemented at protocol/system level consists in frequently updating the session keys used by the cryptographic device. Because DPA attacks in particular require the processing of multiple power traces captured when operating with a fixed key, such a countermeasure inherently limits the applicability of these attacks. Techniques based on fresh re-keying [139, 157, 156] are a perfect example of such an approach.

2.3.2 Against Fault Attacks

The majority of low-level protections against fault attacks are based on the addition of physical countermeasures to prevent fault injection mechanisms [30]. The insertion of glitches or spikes on external interfaces can be mitigated by adding on-chip filters. Alternatively, the use of internally generated clock signals and voltage regulators may prevent these signals to be externally tampered with. Unstable clocks may lower the control an adversary has on injecting precise faults. The use of active sensors, e.g. to detect light or temperature variations, may similarly lessen the applicability of some techniques. Metal shield layers over the active circuitry are often employed to limit the applicability of (semi-) and invasive approaches.

Redundancy techniques can also be implemented at low-level layers to detect the insertion of faults. Dual-rail logic styles provide some robustness against certain fault attacks due to their structure, as changing one bit of a valid data encoding can result in entering one of the metadata states. Alternatively, one can consider the addition of redundancy in underlying hardware blocks. Parity checks in sensitive elements of the digital circuit, e.g. ALU, instruction decoders, memory data buses, etc. may be able to detect faults injected on the processed data.

High-level mitigations against fault attacks are often classified in two main groups: *error detection* and *infective computation*. The former is based on adding some sort of redundancy within the cryptographic algorithm in order to detect the injection of a fault and prevent the leakage of exploitable information. The latter on the other hand, adds a series of intermediate steps to make the effect of injected faults highly complex by generating a faulty output that does not reveal information about the cryptographic keys.

The most common technique for error detection consists in the addition of one or more re-computation steps. A straightforward deployment of this idea is given by the so-called *doubling method*, which is based on running two consecutive (or complementary) executions of the same algorithm and comparing the results. More efficient approaches based on the use of error correction codes [36, 132, 149] or digest values [104] have been explored in the context of block ciphers, particularly for AES.

Infective computation mechanisms were originally proposed by Yen *et al.* [232] to secure implementations of RSA-CRT. The core idea of this technique is to spread the effect of a fault into both CRT-branches, rendering the output value unexploitable. Although several infective computation techniques have been proposed in the last years to protect public-key algorithms [45, 68, 38, 196, 96], most of them have been broken [231, 226, 38, 96].

In the context of symmetric-key algorithms, infective computation has been applied to block ciphers by using multiplicative random masking structures [148] and dummy rounds [110]. However, both systems have been recently shown to contain exploitable weaknesses [32].

Because most fault analysis attacks require the analysis of one (or more) correct/faulty ciphertext pairs, a natural high-level countermeasure consists in preventing repetitive computations with the same key. Fresh re-keying mechanisms [139, 157, 156] can be for instance applied in such scenarios.

2.4 Conclusions

In this chapter we have given a broad introduction to the world of physical security. First, we have analyzed the vulnerabilities that enable the two most common physical threats: side channel attacks and fault attacks. For the former we have presented which particularities of CMOS devices result in exploitable leakages; for the latter we have enumerated the most common approaches for fault injection.

Our contribution to this area is a detailed study of the effects of clock glitches on legacy 8-bit microcontrollers. The existence of a two-stage pipeline, together with the lack of knowledge about the chip's internal working, limits the level of detail of the characterization of the observed faults. Yet, we have identified faults with a stable behaviour in our experiments. We have shown that instructions can be replaced rather than perfectly skipped, and that the effects of faults are deterministic and reproducible. The easiest targets, both for glitching the fetching and the execution stage, are multicycle instructions as it is possible to inject faults that do not affect the full pipeline at the same time. We have shown that such faults can be combined with known attacks, and we have illustrated a straightforward attack that is essentially based on injecting one or a sequence of faults in a single execution.

Chapter 3

A Motivating Example

Extracting Secret Keys from Secure EEPROMs

In this chapter we highlight the threat that even nowadays - more than 15 years after the seminal publications - is posed by implementation attacks. We show how a simple combination of low-cost and non-invasive techniques allows to extract the secret authentication keys from a widely used family of embedded secure memories. In the following we provide some background information about the device under attack - Atmel CryptoMemory - we briefly sketch its security functionalities, and we describe all steps that ultimately lead us to recover secret keys. We finish by elaborating on the implications of the attack and by enumerating potential countermeasures.

Publication data

J. Balasch, B. Gierlichs, R. Verdult, L. Batina, and I. Verbauwhede, "Power Analysis of Atmel CryptoMemory - Recovering Keys from Secure EEPROMs", In *Topics in Cryptology - CT-RSA 2012, The Cryptographers' Track at the RSA Conference*, Volume 7178 of Lecture Notes in Computer Science (LNCS), pages 19-34. Springer-Verlag, 2012.

Contribution

Main author.

3.1 Introduction

CryptoMemory, originally released in 2002 as part of the Atmel AT88SCxxxxC series [18], is a family of secure memories with authentication. As shown in Figure 3.1, they are commercially available either in plastic IC packages or as smart cards, and are essentially composed of three elements: a piece of non-volatile EEPROM memory, access control logic, and a cryptographic unit. Reading and writing permissions to EEPROM are only granted to a host after a valid authentication, ensuring that memory contents cannot be accessed by unauthorized parties. A series of Authentication Attempt Counters (AACs) monitors failed authentications, effectively locking the device after four consecutive failed attempts. At the core of the cryptographic unit lies a proprietary stream cipher with secret specification, that we refer to as Atmel cipher. This cipher is employed during the mutual authentication protocol, in which a host and a CryptoMemory device authenticate each other by proving knowledge of a 64-bit shared secret key. A session key resulting from the mutual authentication can be further used to provide authenticated encryption of the communication channel.

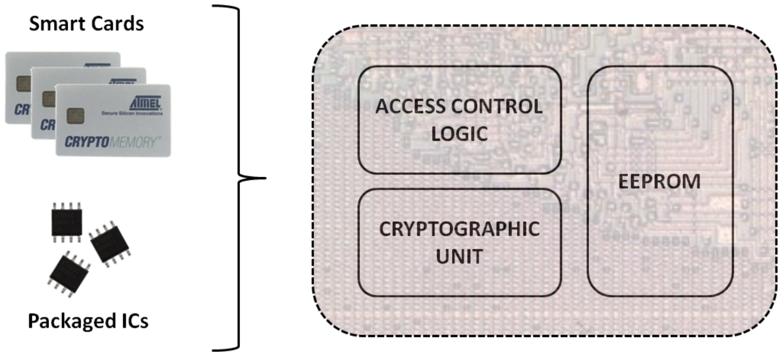


Figure 3.1: Atmel CryptoMemory Family.

CryptoMemory devices provide from 1 Kbit to 256 Kbits of memory divided into several user zones that are individually configurable with different access control policies. A separate configuration zone, customizable during the device personalization phase, is used to store such policies. A set of security fuses can be blown after the personalization phase in order to lock the configuration zone. CryptoMemory offers a total of three different security policies:

a) In password mode, a host simply needs to provide a 24-bit password to gain access to the zone. This mode offers a limited level of security, as all exchanged data (including passwords) is transmitted in the clear, i.e. it is vulnerable to eavesdropping and replay attacks.

b) In authentication mode, up to four 64-bit keys (in the following denoted by k) can be set during the personalization phase as shared secrets between host and device. An 8-bit AAC associated to each key controls the number of failed authentication attempts. The protected user zone(s) become inaccessible once AAC is set to x00[1]. Data transmitted to/from protected memory in this mode is in the clear but replay attacks do not apply.

c) In encryption mode, the communication channel between host and device is protected by authenticated encryption. A 64-bit shared session key K_s that is updated after each run of the mutual authentication protocol is used. Entering encryption mode requires the device to be already in authentication mode.

CryptoMemory's security features, low cost, and ease of deployment have allowed these devices to be widely used in plenty of commercial and military applications [19]. A typical example is the use of the AT88SCxxxxC series to store High Bandwidth Digital Content Protection (HDCP) keys in products such as NVIDIA graphics cards [168], Labgear digital satellite receivers [4], or the Microsoft Zune player [176]. CryptoMemory devices are also used to strengthen security in systems vulnerable to counterfeiting schemes. They are for instance used in printers and printer cartridges manufactured by Dell, Ricoh, Xerox, and Samsung [1]. Other potential applications include appliances with smart batteries, set top boxes, video game consoles, video game cartridges, PDAs, GPS, and any system with proprietary algorithms or secrets[20]. In smart card form, CryptoMemory devices are mainly used in vendor-specific electronic payments, e.g. in laundromats or parkings [2]. Further applications recommended by the manufacturer include ID and access cards, health care cards, loyalty cards, internet kiosks, energy meters, and e-government [33].

Security Claims. Atmel's marketing documentation states that CryptoMemory devices "*can secure data against all the most sophisticated attacks, including algorithmic attacks, systematic attacks, and physical attacks.*" [17]. In particular, physical attacks are counteracted by the use of tamper-proof features that includes metal shield layers over the active circuitry, encrypted internal buses, high-security test procedures, and defenses against timing and power supply attacks (to be understood as active tampering attacks, e.g. fault injection via glitching).

[1]In the most restrictive mode the maximum number of authentication attempts is set to four. The decreasing values of AAC are (xFF,xEE,xCC,x88,x00). A correct authentication automatically restores the value of AAC to xFF.

CryptoMemory also provides anti-tearing functionalities, i.e. in the event of a power loss during an EEPROM writing cycle, data can be recovered at the next power-up and written to the intended address. This feature is however optional, and it needs to be requested by the host prior to a write operation. A typical scenario in which this mechanism enhances security is payment systems, e.g. a malicious customer could try to remove a CryptoMemory-based card from the terminal slot before a decreased balance has been written to memory.

Surprisingly, we could not find a single reference to countermeasures against power analysis attacks in Atmel's marketing material and technical documentation. However, given the effort made to protect against invasive probing attacks and non-invasive tampering attacks, it is reasonable to assume that also power analysis attacks were considered. After all, it is claimed that "*CryptoMemory is designed to keep contents secure, whether operating in a system or removed from the board and sitting in the hacker's lab.*" [14]. In our view, it is likely that Atmel relies on the secrecy of the Atmel cipher and the AACs as countermeasures against basic power analysis attacks.

3.2 Related Work and Background

As it has previously occurred with other products using proprietary cryptographic algorithms [48, 102, 167], the security of CryptoMemory devices took a hit in 2010 when Garcia et al. [103] reverse-engineered the Atmel cipher and the authentication protocol used in the CryptoMemory family. The authors also cryptanalyzed these mechanisms and showed that an adversary could recover CryptoMemory authentication keys in 2^{52} cipher ticks using 2640 eavesdropped keystream frames. Biryukov et al. recently proposed an improved attack [43] that requires 30 eavesdropped keystream frames and 2^{50} cipher ticks to recover authentication keys. Other attacks against systems using CryptoMemory are known [147, 224], but they exploit weaknesses in poorly designed protocols and mistakes during deployment rather than vulnerabilities of CrypoMemory devices.

Atmel cipher. In the following, and for the sake of completeness, we briefly sketch the main functionality of the Atmel cipher. For a more formal and complete specification we refer the reader to [103].

Figure 3.2 depicts the inner structure of the Atmel stream cipher. The cipher state s is an element of $\mathbb{F}_2^{117}$ composed by a total of 4 shift registers. We denote these elements as left register l, middle register m, right register r, and feedback register f. In particular:

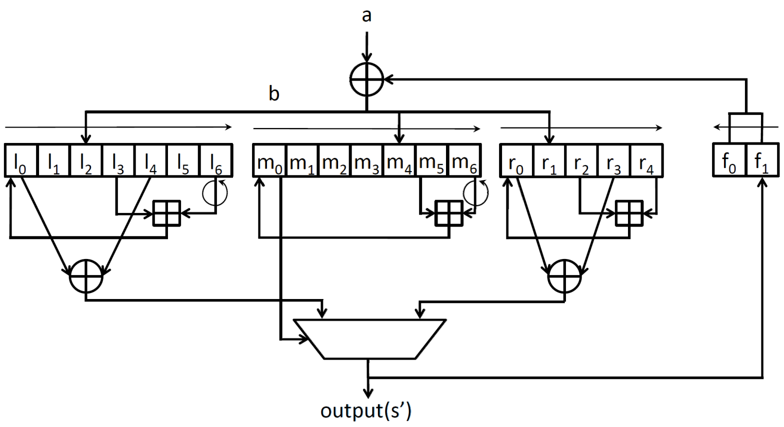


Figure 3.2: Atmel cipher.

1.	left register:	l	$=$	$(l_0, l_1, ..., l_6)$	$\in$	$(\mathbb{F}_2^5)^7$
2.	middle register:	m	$=$	$(m_0, m_1, ..., m_6)$	$\in$	$(\mathbb{F}_2^7)^7$
3.	right register:	r	$=$	$(r_0, r_1, ..., r_4)$	$\in$	$(\mathbb{F}_2^5)^5$
4.	feedback register:	f	$=$	(f_0, f_1)	$\in$	$(\mathbb{F}_2^4)^2$

At each tick, the cipher state $s = (l, m, r, f)$ is transformed into a successor state $s' = (l', m', r', f')$ going through an intermediate state $\hat{s} = (\hat{l}, \hat{m}, \hat{r}, \hat{f})$. During this whole process the cipher takes a single input $a \in \mathbb{F}_2^8$ and produces an output keystream nibble $output(s') \in \mathbb{F}_2^4$.

Mutual authentication protocol. The mutual authentication protocol between a CryptoMemory device and a host is illustrated in Figure 3.3. Let $n_r \in (\mathbb{F}_2^8)^8$ be a host nonce, $n_t \in (\mathbb{F}_2^8)^8$ be a CryptoMemory device nonce, and $k \in (\mathbb{F}_2^8)^8$ be a shared secret key. We denote a_r and $a'_r \in (\mathbb{F}_2^8)^8$ the host challenge authenticators, and a_t and $a'_t \in (\mathbb{F}_2^8)^7$ the device challenge authenticators. Both values are computed after feeding the values (n_r, n_t, k) into the Atmel cipher.

In the first phase of the protocol, the host reads the device randomness n_t and uses it, together with its own randomness n_r and the shared key k, to compute the authenticators (a_r, a_t). In the second phase, the host sends an authentication command to the device, namely $auth(n_r, a_r)$, including the value n_r and the first authenticator a_r. The device computes its own authenticators (a'_r, a'_t), and checks whether the provided a_r equals the calculated a'_r. If the check fails, the value of AAC is decreased; otherwise, it is set to xFF. The device

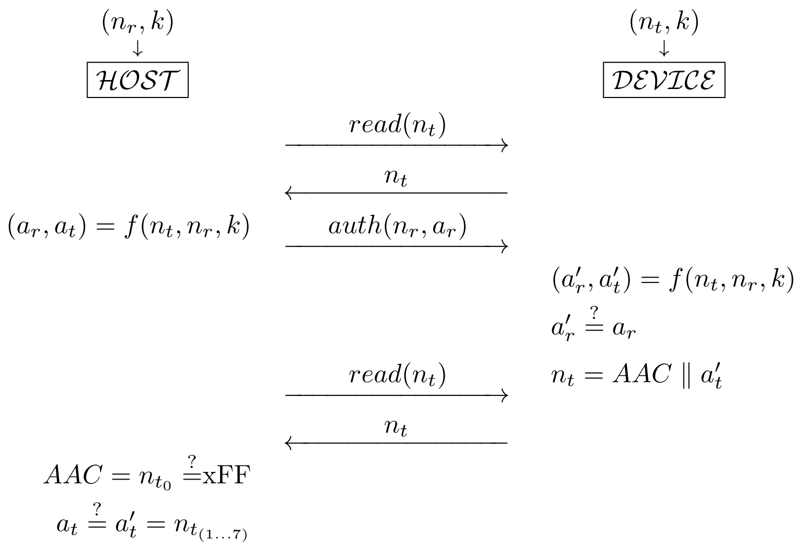

(n_r, k)
$\mathcal{HOST}$
(n_t, k)
$\mathcal{DEVICE}$
$read(n_t)$
n_t
$(a_r, a_t) = f(n_t, n_r, k)$
$auth(n_r, a_r)$
$(a'_r, a'_t) = f(n_t, n_r, k)$
$a'_r \overset{?}{=} a_r$
$read(n_t)$
$n_t = AAC \parallel a'_t$
n_t
$AAC = n_{t_0} \overset{?}{=} \text{xFF}$
$a_t \overset{?}{=} a'_t = n_{t_{(1\ldots7)}}$

Figure 3.3: CryptoMemory mutual authentication protocol.

also updates the value of n_t by concatenating the 8-bit AAC with the 56-bit authenticator a'_t. In the final phase, the host reads the recently updated value of n_t. It first checks whether the authentication was successful (i.e. whether AAC holds the value xFF), and later compares the authenticator a_t with the provided a'_t. If all checks are correct, then the mutual authentication protocol succeeds.

The procedure to compute the authenticators (a_r, a_t) resulting from feeding the values (n_t, n_r, k) into the Atmel cipher is intuitively depicted in Figure 3.4. Each of the states s_i indicates one tick of the cipher for which an input byte a is given and an output nibble $output(s')$ is obtained. At the start of the protocol all registers of the internal state are initialized to zero. In a first phase, ranging from states s_0 to s_{55}, the three parameters (n_t, n_r, k) are scrambled into the cipher state. The output keystream nibbles generated by the cipher are for the moment ignored. In a second phase, from states s_{56} to s_{125}, all input bytes are set to zero. The output nibbles obtained after some of the ticks are used to form the authenticators a_r and a_t. In the final phase, the session key K_s is computed during states s_{126} to s_{141}.

State	**Input bytes**							**Output nibbles**						
(s_{0-6})	n_{t_0}	n_{t_0}	n_{t_0}	n_{t_1}	n_{t_1}	n_{t_1}	n_{r_0}	-	-	-	-	-	-	-
(s_{7-13})	n_{t_2}	n_{t_2}	n_{t_2}	n_{t_3}	n_{t_3}	n_{t_3}	n_{r_1}	-	-	-	-	-	-	-
(s_{14-20})	n_{t_4}	n_{t_4}	n_{t_4}	n_{t_5}	n_{t_5}	n_{t_5}	n_{r_2}	-	-	-	-	-	-	-
(s_{21-27})	n_{t_6}	n_{t_6}	n_{t_6}	n_{t_7}	n_{t_7}	n_{t_7}	n_{r_3}	-	-	-	-	-	-	-
(s_{28-34})	k_0	k_0	k_0	k_1	k_1	k_1	n_{r_4}	-	-	-	-	-	-	-
(s_{35-41})	k_2	k_2	k_2	k_3	k_3	k_3	n_{r_5}	-	-	-	-	-	-	-
(s_{42-48})	k_4	k_4	k_4	k_5	k_5	k_5	n_{r_6}	-	-	-	-	-	-	-
(s_{49-55})	k_6	k_6	k_6	k_7	k_7	k_7	n_{r_7}	-	-	-	-	-	-	-
(s_{56-62})	0	0	0	0	0	0	0	-	-	-	-	a_{r_0}	a_{r_1}	-
(s_{63-69})	0	0	0	0	0	0	0	-	-	-	-	a_{r_2}	a_{r_3}	-
(s_{70-76})	0	0	0	0	0	0	0	-	-	-	-	a_{r_4}	a_{r_5}	-
(s_{77-83})	0	0	0	0	0	0	0	-	-	-	-	a_{r_6}	a_{r_7}	-
(s_{84-90})	0	0	0	0	0	0	0	-	-	-	-	a_{r_8}	a_{r_9}	-
(s_{91-97})	0	0	0	0	0	0	0	-	-	-	-	$a_{r_{10}}$	$a_{r_{11}}$	-
(s_{98-104})	0	0	0	0	0	0	0	-	-	-	-	$a_{r_{12}}$	$a_{r_{13}}$	-
$(s_{105-111})$	0	0	0	0	0	0	0	-	-	-	-	$a_{r_{14}}$	$a_{r_{15}}$	-
$(s_{112-118})$	0	0	0	0	0	0	0	a_{t_0}	a_{t_1}	a_{t_2}	a_{t_3}	a_{t_4}	a_{t_5}	a_{t_6}
$(s_{119-125})$	0	0	0	0	0	0	0	a_{t_7}	a_{t_8}	a_{t_9}	$a_{t_{10}}$	$a_{t_{11}}$	$a_{t_{12}}$	$a_{t_{13}}$
$(s_{126-132})$	0	0	0	0	0	0	0	K_{s_0}	K_{s_1}	K_{s_2}	K_{s_3}	K_{s_4}	K_{s_5}	K_{s_6}
$(s_{133-139})$	0	0	0	0	0	0	0	K_{s_7}	K_{s_8}	K_{s_9}	$K_{s_{10}}$	$K_{s_{11}}$	$K_{s_{12}}$	$K_{s_{13}}$
$(s_{140-141})$	0	0						$K_{s_{14}}$	$K_{s_{15}}$					

Figure 3.4: Generation of authenticators (a_r, a_t) and K_s given inputs (n_t, n_r, k).

3.3 Developing an Attack Path

Attack goal. The aim of an adversary targeting CryptoMemory devices may vary depending on the deployment setting, but typically the objective will be to either read protected information (e.g. to manufacture cloned chips) or to overwrite memory contents (e.g. to increase the balance in payment scenarios).

In the following we will assume an adversary who possesses a CryptoMemory device configured either in authentication mode or encryption mode. Note that the first security operation in either of these modes is always the mutual authentication protocol. In other words, their security relies on the secrecy of the authentication key (or keys) k. If an attacker knew k, he could easily compute session keys K_s and encrypt/decrypt valid communications. Therefore, we define the goal of the adversary as the recovery of k.

Attack approach. Previous work has already pointed out cryptographic weaknesses in the Atmel cipher, but the most efficient attack still requires substantial computational effort, e.g. two to six days computation on a cluster with 200 cores [43]. We focus on power analysis attacks as they are often more practical. Note that in the case of deployed CryptoMemory devices, eavesdropping the communication line for later cryptanalysis requires physical access to the target device. Thus, physical access for power analysis does *not* imply an additional requirement.

Target devices. We have tested three different models of the CryptoMemory family, namely the chips AT88SC0404C, AT88SC1616C, and the newer AT88SC0808CA. These devices differ in the amount of user zone slots and/or size available to the end application, but the protocol to carry out the mutual authentication is identical for all of them.

Given that CryptoMemory devices do not contain a microcontroller, we can assume that the cryptographic unit, including the Atmel cipher, is fully implemented as a hardware module. We further assume that the implementation computes one cipher tick in one or two clock cycles, which gives us an idea of the type of pattern to look for in the power traces.

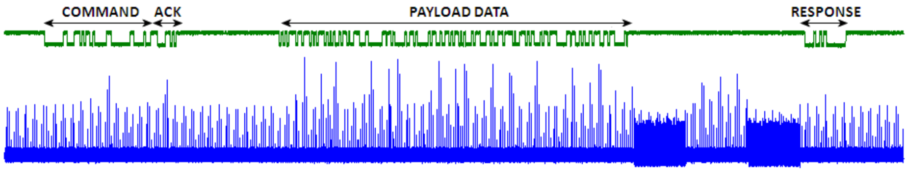
In the following we describe the experimental setup used in our analysis and the steps followed to evaluate the smart card versions of CryptoMemory. We summarize the slightly different approach required for CryptoMemory ICs, leading to the same results, at the end of the section.

3.3.1 Experimental Setup

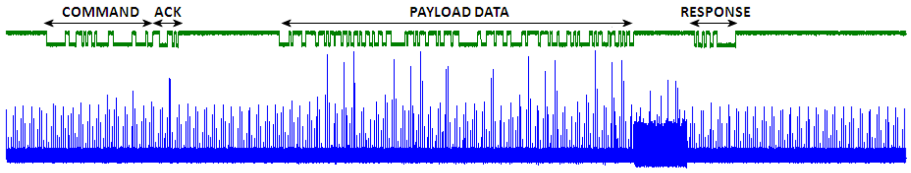
The main element of our experimental setup is a Virtex-II FPGA that communicates with a PC (user interface) and a target device (CryptoMemory). The FPGA can be configured to communicate with either of the CryptoMemory device forms, i.e. as a T=0 compatible smart card reader or as a TWI master device. Apart from the communication channel, the FPGA fully controls all external signals provided to the target device. For instance, the reset signal (RST), the external clock (CLK), and even the power supply (VCC) can be manipulated at will and at any point in time. A 50 Ohm resistor is inserted in series in the ground (GND) line of the target device. We use a Tektronix DPO7254 oscilloscope to measure the voltage drop over this resistor, thus obtaining a trace proportional to the device's power consumption. Note that we do not exhaust the capabilities of the oscilloscope and that a low-cost model could be used just as well. For all experiments we used a moderate sampling rate of 100 MS/s and 20 MHz bandwidth. In addition to the power consumption the oscilloscope also monitors the RST, CLK, and I/O lines.

3.3.2 Initial Investigation of Power Traces

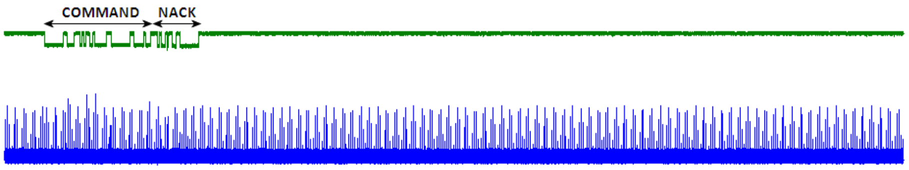
The first step in a power analysis attack is to determine in which part of the protocol the secret key is used by the target device, i.e. at which points in time are the key bytes fed as input to the Atmel cipher. An overview of the I/O and a power trace obtained during the processing of a mutual authentication command is shown in Figure 3.5. Figure 3.5a represents a successful authentication, Figure 3.5b represents a failed authentication, and Figure 3.5c represents an authentication attempt when the AAC is set to x00.



(a) Measurements for a successful authentication.



(b) Measurements for a failed authentication.



(c) Measurements for a not allowed authentication.

Figure 3.5: I/O and power trace for a successful authentication (a), a failed authentication (b), and a not allowed authentication (c).

The leftmost part of the figures starts with the host sending 5 bytes corresponding to an authentication command. After receiving these bytes, the card can reply either with an acknowledgement ACK, indicating that it accepts the command, or with a negative acknowledgement NACK, refusing to process the command. The latter is shown in Figure 3.5c, where the CryptoMemory device has locked access to the associated user zones. If the device acknowledges the command then the host sends the payload data, i.e. 16 bytes corresponding to the values n_r and a_r. Upon reception, the card performs a series of operations to determine whether the authenticator provided by the host is correct. Finally, the card sends a response to the host indicating the outcome of the authentication attempt.

The most interesting part of the figure corresponds to the card calculations upon reception of the payload data. One can clearly notice that the calculation interval in Figure 3.5b is significantly shorter than in Figure 3.5a. Further, a clearly distinguishable pattern with higher power consumption appears twice for the valid authentication and only once for the invalid authentication. This pattern corresponds to EEPROM writings in configuration memory: the first one, present for both valid and invalid attempts, corresponds to decreasing the value of AAC; the second one, only present for the valid authentication, corresponds to writing the new value of n_t and the session key K_s. Note that writing a new value n_t implies a valid authentication, and thus AAC is restored back to xFF.

Although not visible in these overview plots, we noticed that the cryptographic unit appears to run at a clock frequency slower than that we provided externally. Further investigations showed that the device internally derives this slower clock signal by dividing the external clock by approximately a factor 200.

The card's calculation of the values (a'_r, a'_t) must happen before the second EEPROM write operation, simply because the n_t cannot be updated without having computed a'_t beforehand. Our experiments have shown that the device computes the authenticators (a'_r, a'_t) on-the-fly, i.e. while the payload data (n_r, a_r) is being received from the host. Similarly, the calculation of the session key K_s is performed between the two EEPROM writings, only after the device has authenticated the host by verifying that a_r and a'_r are equal.

Figure 3.6 shows a zoomed version of the I/O and power traces during the transmission of the value $n_r = (n_{r_0}, \ldots, n_{r_7})$ to the card. A clearly distinguishable peak in the power trace is visible at the end of each byte transmission, while a total of six high peaks are also identifiable during the transmission of a byte. As explained in Section 3.2 the calculation of (a'_r, a'_t) requires 126 cipher ticks. The pattern in the power traces has a perfect mapping with the cipher behavior illustrated in Figure 3.4 considering a hardware implementation. In fact, the first peak in Figure 3.6 corresponds to the state s_6, i.e. when the value n_{r_0} is fed to the cipher. The following six peaks correspond to states s_7 to s_{12}, in which the card uses its own randomness (values n_{t_2} and n_{t_3}) as inputs. Once n_{r_1} has been received over the I/O line, a new peak corresponding to state s_{13} appears in the power trace, and the pattern is repeated for every transmitted byte. In summary, each of the 50 peaks highlighted in Figure 3.6 corresponds to a cipher state ranging from s_6 to s_{55} in Figure 3.4.

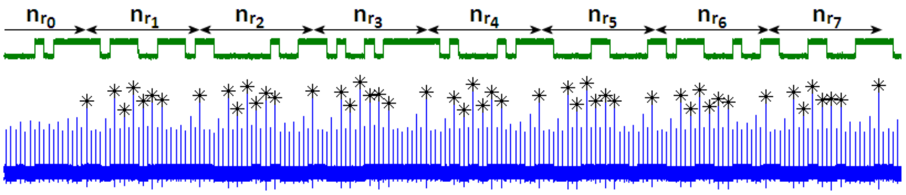


Figure 3.6: I/O and power traces during the transmission of n_r in authentication command. Interesting peaks are marked with *.

Since the key k is scrambled into the cipher states s_6 to s_{55}, the device might leak sensitive information through its power consumption. We could not immediately identify a visible pattern in the power consumption that could relate to k (SPA leak) but we also did not expect that from a hardware implementation of a stream cipher. Therefore, we focused our attention on attacks that use statistical post-processing of the collected power traces. Recall that DPA attacks require the processing of multiple power traces corresponding to multiple authentication attempts. For each attempt the device must use the same (unknown) k and varying input data.

3.3.3 Overcoming Authentication Attempt Counters

Even though we have identified the parts in the power traces that correspond to processing of the secret key k, an important practical issue still remains. In our adversarial model, the adversary possesses a CryptoMemory device that is *already* configured. Thus, he does not know the secret key k. In order to run the mutual authentication protocol, an adversary needs to provide an authenticator a_r to the device. However, as the attacker cannot compute this value correctly, the CryptoMemory device will not authenticate the host and, as a consequence, it will decrease the associated AAC. Given that the user zones become inaccessible to the host after four failed authentication attempts, an adversary can collect at most three power traces before permanently locking the device. The issue is that three traces are clearly not sufficient to carry out a successful DPA attack.

There are several ways to try to deal with this limitation. If the application under attack were to use the same key in all deployed CryptoMemory devices, then it would suffice to collect power measurements from several devices. One could also try to take many measurements from a single device, effectively sacrificing it. But as can be seen in Figure 3.5c, once the AAC value is set to x00 the device no longer computes the authenticators and measurements would thus be worthless. Therefore, an adversary could obtain at most four power traces per device tested. However, a scenario in which all devices share a single key k seems unlikely to be found in secure deployments[2].

An alternative could be to use template attacks as introduced by Chari et al. [63]. The devices provided by Atmel in evaluation kits, completely configurable by the user, could be used to build such templates. We expect template attacks to require very few power traces from the target device, but it is not clear if three traces would suffice, due to the large cipher state. We did not investigate this approach further as we were interested in more simple attack paths.

We followed a more intuitive approach to overcome the limitation imposed by the AACs. Recall from Figure 3.5 that *all* the information required to perform DPA, i.e. key-dependent information in power measurements, is obtained *while* the value n_r is being sent to the card. In other words, the information is available to the attacker *before* the card actually decreases the value of AAC. Our approach consists in injecting a negative pulse in the RST signal of the device before the new value of AAC is written into configuration memory. Doing so forces the device to reset its state before beginning the EEPROM write operation.

A successful implementation of this simple procedure is shown in Figure 3.7. Besides the I/O and power traces, the figure also shows the RST signal input to the device. Injecting a pulse on the RST line right after sending the payload data successfully resets the device. This can be observed on the I/O line, as the device sends its Answer-To-Reset (ATR) value to the host device right after the rising edge in RST. Note also that the first EEPROM write pattern indicating AAC being decreased does not appear in the power trace.

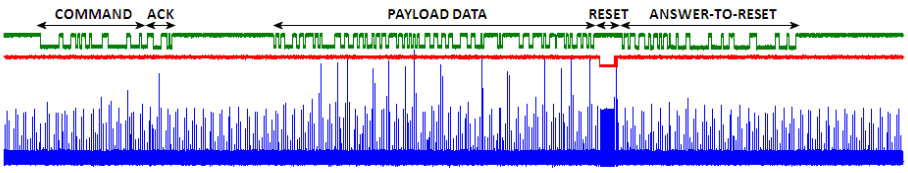


Figure 3.7: I/O, RST, and power traces during interrupted authentication.

The timing of the RST pulse is far from critical as the adversary can inject it at any point after the reception of n_r and before the first EEPROM writing, i.e. during the transmission of the value a_r. Recall that the transmission of a bit in the I/O line requires 372 clock cycles, and a byte transmission requires a total of 12 bits (1 start bit, 8 data bits, 1 parity bit, and 2 stop bits) [3]. Taking that

[2]Atmel actually recommends to diversify keys in all CryptoMemory deployments by combining the configurable device ID with a unique master key, e.g. using a hash function.

into account, the adversary has an interval of 35 712 clock cycles in which the RST pulse can be sent to the card.

Note that resetting the device before the EEPROM writings implies that the value n_t can never be updated. As a consequence, all power measurements collected using this approach will correspond to the same device randomness n_t. Although in other scenarios this characteristic could be problematic, in our case it does not limit the success or applicability of DPA attacks. This is because an attacker can still provide varying values of n_r for each authentication attempt, which is fed into the Atmel cipher some ticks before the key k.

Differences with CryptoMemory packaged ICs. Besides some particularities caused by the use of TWI instead of T=0, the overall behavior of CryptoMemory packaged ICs resembles what we have presented until now. Most important is the fact that the calculation of the parameters (a'_r, a'_t) is done in exactly the same way as in the smart card, i.e. on-the-fly while the host sends the values (n_r, a_r). It is thus possible to identify which zones of the power measurements correspond to which states of the Atmel cipher during the feeding of the values (n_t, n_r, k).

The main physical difference between CryptoMemory packaged ICs and smart cards is that the former do not have an external RST pin. This could have been a problem, but an “equivalent” mechanism to overcome the AACs consists in cutting the supply voltage (VCC) before the counters are decreased. Similarly to the RST mechanism the timing accuracy to cut the voltage is not critical, and the adversary has plenty of time to perform it[3].

Once the power traces are obtained, the attacks on CryptoMemory in smart card form and in IC form are identical and lead to very similar results.

3.4 Power Analysis Attack

We obtained a set of 1000 power traces sampled during executions of the mutual authentication protocol, for which we provided random nonces n_r to the device and, each time, reset it as described above.

We processed the traces with a simple routine that extracts the peaks highlighted in Figure 3.6, yielding a new set of highly compressed and well aligned traces. Contrary to typical attacks on block cipher implementations, the key bytes can

[3]CryptoMemory packaged ICs require the host to perform acknowledge polling after sending a mutual authentication command. If the host does not send any polling command the IC is effectively idle, which gives enough room for an adversary to switch off the supply voltage.

not be recovered independently but should be recovered in the order in which they are fed into the Atmel cipher implementation, i.e. first k_0, then k_1, etc.

We first investigated the feasibility of basic DPA attacks that recover k one byte at a time. We used a Hamming distance power model on the full cipher state, i.e. the total number of bit flips in the transition from state s to state s', and Pearson's correlation coefficient as distinguisher [53].

As explained in Section 3.2, each key byte is fed into the cipher three times in consecutive cipher ticks. Our basic attack worked best when we attacked the last cipher tick that fed in a given key byte, e.g. for byte k_0 the transition from state s_{29} to s_{30}.

Figure 3.8 exemplarily shows the results we obtained when attacking k_6 (the worst case). The left part of the figure shows the correlation coefficients for all 256 hypotheses, plotted over "time", computed using all 1000 measurements. The right part of the figure shows the evolution of the maximum and minimum correlation peaks per key hypothesis, plotted over the number of traces used. We verified that all key bytes can be recovered in this way using less than 500 traces.

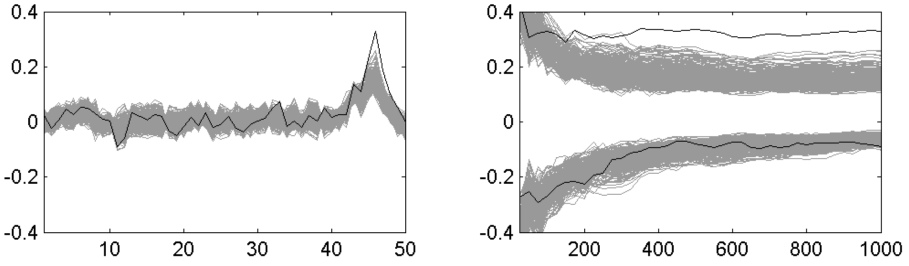


Figure 3.8: Results of basic DPA attack. Correlation coefficients per key hypothesis (left), and evolution of correlation peaks per key hypothesis (right).

Our slightly more elaborate attack additionally exploits two simple facts. First, we know exactly which sample in the power traces corresponds to which cipher tick. Thus, the attack focuses on the correct sample in the traces and ignores all other samples that appear as noise. Second, each key byte is fed into the cipher three times. Thus, the attack targets all three transitions and adds up the obtained correlation coefficients, per key hypothesis.

In addition, the attack further exploits that the dependence of the cipher state on k grows only slowly and byte per byte. Similar to the strategy described in [94], the attack does not aim to immediately identify the exact value of a key byte, but it maintains a list of the best candidate values and re-evaluates them when attacking the next key bytes.

We verified that this enhanced attack recovers the correct key k using less than 100 measurements. We note that both attacks, from measurements to full key recovery, can be carried out in less than 20 minutes on a standard laptop. Algebraic side channel analysis would perhaps allow to work with even fewer traces, but it requires to build templates.

3.5 Implications and Countermeasures

We have shown that an adversary can easily extract the secret authentication key(s) k from CryptoMemory devices using basic, well-understood power analysis techniques. As a consequence, the adversary can perform all actions that any authenticated host could perform. This includes reading the memory, which allows to clone the device, and manipulating its memory contents at will.

The success of our attack is due to two design flaws in CryptoMemory. First, the implementation of the Atmel cipher is not protected by countermeasures against power analysis attacks, except for the AACs that limit the number of traces that can be obtained from a device before it locks itself to at most three. And second, an inadequate handling of the AACs that allows an adversary to bypass this limitation and to obtain any number of measurements from a given device without locking it.

A simple way to prevent our attack would be to modify the handling of the AACs. As learned from our experiments, CryptoMemory performs the authentication procedure as follows: compute the authenticators, decrease the value of AAC, compare the authenticators, and update the value of AAC according to success/failure when writing n_t in memory. This sequence can also be extracted from Figures 3.5a and 3.5b. Atmel explicitely states that this procedure is used "*to prevent attacks*" [18]. In fact, the method protects only the comparison operation and is often used e.g. in SIM cards during PIN verification. Our attacks do, however, not target the comparison operation but the time instants when the secrets are manipulated. Since CryptoMemory manipulates the secret key(s) *before* it decreases AAC, the counters can be easily bypassed with a reset. A more secure handling of the AACs could be to decrease the counter right upon reception of a mutual authentication command, and *prior* to the reception of the payload data and computation of the authenticators.

Protecting against more sophisticated attacks than ours may require to implement some of the countermeasures against power analysis attacks enumerate in Chapter 2, e.g. noise generators and power filters at the hardware level, or masking, random delays, and shuffling at the circuit level.

3.6 Conclusions

In this chapter we have illustrated the powerful nature of physical attacks by attacking a real-world device with security functionalities. Moreover, we have carried out this attack with the capabilities of a Class I adversary.

After reviewing the features, primitives and protocols employed by CryptoMemory, we have developed an attack path aiming to the extraction of its secret key. Starting from a custom made and flexible experimental setup, we have first identified the time instants in power traces carrying key-related information. After this, we have devised an approach to overcome the limitation posed by the AACs. Ultimately, we have applied well-understood power analysis techniques to extract the secret cryptographic key of the device.

Our results in this chapter highlight the difficulty of preventing physical attacks at every layer of the embedded design space. CryptoMemory uses fuses to lock access control policies, access control to protect memory contents, and AACs to strengthen access control. Therefore, a large part of CryptoMemory's physical security relies on the AACs, that we have identified as a not sufficiently protected point of failure.

Chapter 4

Masking at Algorithm Level

Higher-Order Masking based on the Inner Product

This chapter focuses on data randomization techniques that can be applied at algorithm level to counteract power analysis attacks. After briefly introducing the approaches devoted to protect public-key primitives, the focus of the chapter turns to the application of masking techniques for protecting block ciphers. This topic is currently one of the most active within the research area of side channel countermeasures. We formalize the concept of masking and provide a chronological overview of developments on both attacks and countermeasures, ultimately leading to our contributions in the field.

4.1 Introduction

The goal of data randomization techniques, commonly known as *masking*, consists in randomizing any *sensitive variable* that appears during the processing of a cryptographic algorithm. Formally speaking, a variable is said to be sensitive if it can be expressed as a deterministic function of the plaintext (resp. ciphertext), the secret key, and possibly other parameters that are not constant with respect to the key [188]. Said variables are commonly the target in power analysis attacks.

Because of their mathematical structure, data randomization techniques are commonly integrated into public-key cryptosystems. Most proposals in the literature aim to protect modular exponentiation algorithms, and are thus generic for RSA and ECC. These techniques can either be applied by randomizing input messages, e.g. message blinding [140], or by randomizing exponents, e.g. exponent blinding [73] or exponent splitting [71]. In the context of ECC, one may resort as well to the randomization of projective coordinates [73].

Protecting symmetric key primitives on the other hand requires more effort. Block ciphers have iterative constructions combining a series of transformations within multiple rounds. Linear transformations such as bit permutations are in general well suited to combine with masking techniques. Non-linear transformations (commonly S-boxes) require more effort and can lead to large efficiency bottlenecks. Furthermore, applying data randomization to fully prevent side channel leakages has proven to be a non-trivial task, and it often depends on the nature of the final implementation, i.e. software-based or hardware-based. These issues are further discussed in the following sections.

4.2 Masking Block Ciphers

The generic principle of masking [62] consists in randomly splitting all intermediate variables X processed by a cipher into n shares such that $X = x_1 \odot x_2 \odot \ldots \odot x_{n-1} \odot x_n$. The operator $\odot$ corresponds to a group operation (or a combination thereof) and its selection determines the *type* of masking. Classical examples include *boolean* masking (based on the exclusive-or or XOR operand) and *arithmetic* masking (based on modular addition). When a type of masking is adapted to protect a particular cryptographic implementation or a set of operations, one talks about a *masking scheme*.

A masking scheme defines a construction that specifies how to propagate all shares through the cipher transformations while ensuring intermediate results remain independent of sensitive variables. Assuming the masks are uniformly distributed, a side channel adversary can in principle only observe leakage components $L(x_i)$. As these are statistically independent of the sensitive variable X, their leakage distribution cannot be effectively exploited by a side channel adversary.

The minimum number of shares $d+1$ required to reconstruct a sensitive variable determines the security level of the masking scheme. This is commonly referred to as dth-order masking. While it is often the case that $n = d+1$, i.e. *all* shares are required to fully reconstruct X, recent proposals are not subject to this constraint. In the following we employ a more generic notation introduced by Prouff and Roche [180] and refer to a masking construction as $(n, d) - sharing$.

4.2.1 1st-Order Masking

The $(n = 2, d = 1) - $ *sharing* of the form $X = x_1 \odot x_2$ is the most common and representative of *1st-order* masking constructions. It provides a sound countermeasure against classical *univariate* power analysis attacks, i.e. those targeting atomic leakages of sensitive variables at a single point in time. Examples of attacks falling into this category are the classical versions of DPA [141], CPA [53], and MIA [108].

The earlier application of 1st-order masking is due to Goubin and Patarin [115]. Originally named "duplication method", the construction devised in [115] uses *boolean* masking to protect implementations of DES. Each intermediate value X in the computation is split into two shares x_1 and x_2 such that $x = x_1 \oplus x_2$. Propagating and correcting these shares through most operations of the DES algorithm, i.e. bit permutations, expansions, rotations, and exclusive-or, is straightforward due to their linearity with respect to the masking construction. The main difficulty lies in dealing with the non-linear step of the cipher, in this case the DES S-boxes.

The technique proposed in [115] consists in generating two new S-boxes S_1' and S_2', such that the shares are propagated as $x_1' = S_1'(x_1, x_2)$ and $x_2' = S_2'(x_1, x_2)$. The first S-box S_1' is a randomly chosen transformation from a 12-bit input to a 4-bit output, while S_2' is computed as $S_2'(x_1, x_2) = S(x_1 \oplus x_2) \oplus S_1'(x_1, x_2)$. Here S refers to either of the original DES S-Boxes, mapping a 6-bit input to 4-bit output. This approach requires to store 32 KBytes of data corresponding to masking the 8 different DES S-boxes, although the paper elaborates on alternative, more memory-efficient techniques.

Masking non-linear functions at the cost of larger memory complexities is commonly known as *table re-computation*. This technique was originally hinted by Chari *et al.* [62] and further formalized by Messerges [159] as follows. Assume an S-Box T described by a non-linear function $f : \mathbb{F}_2{}^n \rightarrow \mathbb{F}_2{}^m$ such that $T[a]$ denotes the image of a by the corresponding function f. Masking such construction can be generically done by selecting two masks r_{in} and r_{out} and computing a *masked S-Box* such that $T'[a] = T[a \oplus r_{in}] \oplus r_{out}$ for all $a \in \mathbb{F}_2{}^n$.

Alternative techniques to propagate the mask through non-linear transformations have been mostly applied to AES. Because the non-linear step in AES consists in performing a multiplicative inversion in $GF(2^8)$, Akkar and Giraud [13] propose to protect it by means of a *multiplicative* mask such that $X = x_1 \otimes x_2$. This approach requires to switch between boolean and multiplicative masks at the input (resp. the output) of the AES S-Box lookup, and it is referred to as *adaptive masking.* As pointed out by Golic and Tymen [113] however, straightforward multiplicative masking only protects non-zero values, i.e. a byte equal to zero cannot get masked because of the nature of the multiplicative masking. This characteristic makes the method of Akkar and Giraud [13] vulnerable to classic DPA. Follow-up works on 1st-order adaptive masking of the AES have been introduced by Trichina *et al.* [213] and Genelle *et al.* [105], although the former has been broken by Akkar *et al.* [12].

Finally, several other methods to mask the AES S-Box without relying on table re-computation or adaptive masking techniques have as well been suggested. Among them, the use of homographic functions [76], decomposition into a square-and-multiply algorithm [44], or the use of tower field representations [170, 59] are the most relevant ones.

4.2.2 Higher-Order Attacks

The concept of *higher-order* attacks was first suggested by Kocher *et al.* [141], formalized by Chari *et al.* [62], and brought to practice by Messerges [160]. The main application of these attacks is to overcome the security provided by masking techniques. While univariate attacks focus on atomic leakages at single points in time, higher-order attacks - in the following referred to as *multivariate* attacks - exploit joint statistical properties of several time samples in power curves. As a general rule, a dth-order masking scheme can always be theoretically broken by a d-variate attack combining the leakage distributions of d shares.

In practice, most multivariate techniques in the literature perform a *pre-processing* step to adapt a multivariate statistical problem to a univariate one [225], i.e. atomic leakages at different points in time are first aggregated by using a *combining function* and subsequently exploited by performing a classical univariate attack such as DPA or CPA. In contrast to this approach, multivariate MIA [107, 177] is able to exploit leakages in multiple time samples without need of such a pre-processing step, which has been shown to inherently carry a loss of information [62].

Let us illustrate how an exemplary *bivariate* attack is able to break a 1st-order boolean masking scheme in which intermediate variables are processed as $X = X_m \oplus m$. Note that such construction is equivalent to our original definition of 1st-order boolean masking $X = x_1 \oplus x_2$, by denoting $X_m = x_1$ and $m = x_2$. Exploiting the leakage of either the masked variable $L(X_m)$ or the mask $L(m)$ independently using a univariate attack does not lead to a successful attack. This is because $L(X_m)$ depends on an unknown and uniformly distributed random mask, while the leakage of $L(m)$ is independent on the secret keys. The approach followed by a bivariate attack is to merge the leakages $L(X_m)$ and $L(m)$ into a single variable by means of a combination function

$C = f(L(X_m), L(m))$, and then exploit the combined leakage by means of a univariate attack.

The success rate of a bivariate (and in general multivariate) attack relies on a suitable selection of the combination function. In the literature two of such functions have been formally studied. The first one is the *absolute difference combining* method due to Messerges [160], and later formalized by Joye *et al.* [126]. It is based on aggregating the leakage components by computing

$$C = |L(X_m) - L(m)|.$$

The second one is the *product combining* method originally introduced by Chari *et al.* [62]. It is based on computing

$$C = L(X_m) \times L(m).$$

Variants of this function are proposed by Waddle and Wagner [225] in the particular case that both $L(X_m)$ and $L(m)$ leak at the same time instant. Under these conditions, squaring the power traces allows to break the implementation by carrying out a univariate attack. A further variant by Prouff *et al.* [178] adds a normalization step to the product combining method yielding:

$$C = (L(X_m) - E[L(X_m)]) \times (L(m) - E[L(m)]).$$

Bivariate attacks building upon these combination methods have been shown to be successful in defeating implementations protected by 1st-order masking. See for instance [160, 169, 210].

In addition to bivariate attacks, 1st-order masking techniques have been shown to be vulnerable to univariate attacks when implemented in hardware rather than in software. As shown by Mangard [152], the existence of *glitches* on masked gates leads to side channel leakage. Glitches are switching operations of logic gates caused by timing properties of gates and interconnection delays [153]. In CMOS technology, the power consumption of a circuit is highly correlated to the number of glitches that occur. Most masking schemes process masks and masked values within the same combinational circuitry. Glitches inherently combine information of both shares, and this leads to univariate leakage appearing in the power traces. Limiting this source of side channel leakage can be done by using synchronization elements [174], or tackling the problem at algorithm level. Threshold implementations due to Nikova *et al.* [166] represent a particular case of masking schemes that take into account the effect of glitches on hardware-based implementations, although so far their application is limited to counter first-order attacks.

4.2.3 Higher-Order Masking

The design of higher-order masking countermeasures is of practical interest due to two main reasons. First, it was shown by Chari *et al.* [62] that the number of power samples required to retrieve information about an unmasked variable is lower bounded by an exponential function of the masking order whose base is related to the noise standard deviation. In other words, the number of power traces required to break an $(n, d) - sharing$ grows exponentially on d given a sufficient amount of noise. In practice, noise can be added by physical noise generators or algorithmic countermeasures based on shuffling or time randomization. And second, multivariate attacks need to search over $d + 1$ tuples of time samples in order to identify and exploit the leakages corresponding to all shares, meaning that the complexity of attacks grows combinatorially in the attack order. These two characteristics impose a limitation to the practical application of a multivariate attack in terms of computation effort.

The first proposal for a higher-order masking scheme at any order d is due to Ishai *et al.* [124]. The technique, based on boolean masking, is tailored to secure hardware implementations of circuits where internal operations are carried out over $\mathbb{F}_2$. Basically, it implies to transform a circuit composed by NOT and AND gates to a secured version in which no subset of d wires gives information about sensitive variables. Despite its theoretical relevance, a practical downside of this technique is the large overhead it requires in terms of area.

A generalization of the method by Ishai *et al.* [124] to secure any finite field is given by Rivain and Prouff [188]. Using boolean masking, this work devises a series of masked operations particularly tailored to software implementations of the AES. The results of this work have been recently improved by Kim *et al.* [136] in the context of hardware implementations by combining boolean masked operations with tower field representations. Protecting S-boxes at any order by means of boolean masking techniques has been presented by Carlet *et al.* [60].

Alternatively to boolean masking, the use of multi-party computation techniques based on Shamir's secret-sharing [202] has been proposed by Goubin and Martinelli [114] and Prouff and Roche [180]. Both schemes, commonly referred to as *polynomial masking*, delineate a series of masked operations at any order d. However, the secure multiplication routine in the former scheme has recently shown to be vulnerable to univariate attacks by Coron *et al.* [75].

Along with this generic masking techniques that work at any order of d, other works have tackled the issue of higher-order masking. Schramm and Paar [198] devised a higher-order masking scheme to secure AES, but Coron *et al.* [74] showed that the scheme was only secure for $d \leq 2$. Techniques to switch between boolean and multiplicative masking at any order d have been introduced by Genelle *et al.* [106].

To conclude, the remaining efficient $(n, d)-sharing$ constructions at any order d in the literature are the boolean masking scheme by Rivain and Prouff [188] and the polynomial masking scheme by Prouff and Roche [180], which additionally provides security against glitches.

4.3 Inner Product (IP) Masking

Publication data

J. Balasch, S. Faust, B. Gierlichs, and I. Verbauwhede, "Theory and Practice of a Leakage Resilient Masking Scheme", In *Advances in Cryptology - ASIACRYPT 2012*, Volume 7658 of Lecture Notes in Computer Science (LNCS), pages 758-775. Springer-Verlag, 2012.

Contribution

Shared work between authors.

Our contribution to the research area of data randomization countermeasures is the design, evaluation, and implementation of a practical higher-order masking scheme based on *inner product* constructions. Originally proposed in the area of leakage resilient cryptography [93], the greater algebraic complexity of the inner product has been suggested to yield better resistance against side channel attacks than, for instance, boolean masking. However, the original theoretical security analysis requires strong security because it assumes any efficient adversary and against a broad class of leakages, thus preventing it to be efficiently implemented in practice.

The core of our work consists in bringing the inner product masking - in the following IP masking - to practical scenarios. To this end, we devise a complete set of algorithms for carrying out operations in the masked domain. Some of these operations result from modifying and simplifying the original scheme according to practice-oriented security models. An efficiency comparison with other state-of-the art higher-order masking schemes shows that our constructions for non-linear operations, e.g. multiplication, clearly outperform schemes based on polynomial-based masking solutions [180, 114] that enjoy similar algebraic complexity. At the same time, our practical security analysis reveals that the information leakage of IP masking is more than two orders of magnitude smaller than that of boolean masking [188], for low levels of noise and the same number of shares. As a final contribution, we provide a methodology to implement the AES block cipher using the IP masking scheme. A software-based implementation for $d = 3$ shares is provided together with performance results. This not only verifies the correctness of our proposal, but also confirm its feasibility in embedded contexts.

4.3.1 Construction of IP Masking.

Circuit model. Following the model of Dziembowski and Faust [93, 92], we consider that the target device running the masked computations contains two separate processors. Each of these processors, in the following referred to as *left processor* (P_{L}) and *right processor* (P_{R}), executes a part of the masked operations. Communication between processors is performed via a bidirectional data bus. Such a model is introduced in order to provide a framework to analyze the security of the masking scheme. As will be further explained in the following sections, its main purpose is to facilitate the assumption that P_{L} and P_{R} have completely independent side channel leakage, i.e. an adversary can only retrieve information specific to each physical processor. Notice that from a practical point of view, the required independent side channel leakage can be obtained by temporal (rather than physical) separation of the masked computations, e.g. in the context of sequential software implementations on a single processor.

Overview. The IP masking scheme can be instantiated to secure operations in any finite field $|\mathbb{F}| \geq 2$, such that all elements and operations in $\mathbb{F}$ can be mapped to and performed in the masked domain. This feature is extremely useful in the context of securing cryptographic applications, as the underlying field of the masking scheme can be adapted according to the characteristics of the cryptographic algorithm and/or the target platform. Without loss of generality, and driven by our goal to implement the AES, we provide in the following an efficient instantiation of the IP masking scheme for the field $\mathbb{F}_{2^8}$ of characteristic two.

Notation. We represent field elements with upper-case letters, e.g. $X \in \mathbb{F}_{2^8}$, and we use $\oplus$ to denote field addition and $\otimes$ to denote field multiplication. Vectors are represented with bold upper-case letters, e.g. $\mathbf{X} \in \mathbb{F}_{2^8}^n$ such that

$\mathbf{X} = (X_1, \ldots, X_n)$. For two vectors $\mathbf{X}, \mathbf{Y} \in \mathbb{F}_{2^8}^n$ we denote by $\mathbf{X} \oplus \mathbf{Y}$ the vector addition in $\mathbb{F}_{2^8}^n$ calculated as $(X_1 \oplus Y_1, \ldots, X_n \oplus Y_n)$. The inner product $\langle \mathbf{X}, \mathbf{Y} \rangle \in \mathbb{F}_{2^8}$ is calculated as $\bigoplus_{i=1}^n X_i \otimes Y_i$.

Construction. In the IP masking scheme each sensitive variable $X \in \mathbb{F}_{2^8}$ is split into an even number of $2n$ shares such that:

$$X = L_1 \otimes R_1 \oplus \ldots \oplus L_n \otimes R_n. \tag{4.1}$$

We denote $\mathbf{L} = (L_1, \ldots, L_n)$ as *left vector* and $\mathbf{R} = (R_1, \ldots, R_n)$ as *right vector*. A variable X is represented in the masked domain as $(\mathbf{L}, \mathbf{R})$, and can be recovered by calculating the inner product of these two vectors, e.g. $X = \langle \mathbf{L}, \mathbf{R} \rangle$. In order to prevent a practically exploitable bias between the shares and the masked value, it is required that elements of $\mathbf{L}$ belong to $\mathbb{F}_{2^8} \setminus \{0\}$. We define $n \geq 2$ as the security parameter of our masking scheme.

Note that IP masking is a generalization of previously published masking schemes. Indeed, one can trivially derive boolean masking [188] from Eq. (4.1) by setting all elements in $\mathbf{L}$ (resp. $\mathbf{R}$) to one. Multiplicative masking [13] can be achieved by setting $n = 2$ and either of the shares L_2 and/or R_2 (resp. L_1 and/or R_1) to zero. Affine masking, described in [100] as $V = (A \otimes X) \oplus B$, can be obtained by fixing $n = 2$, $L_1 = L_2 = A^{-1}$, $R_1 = V$, and $R_2 = B$. Finally, as a secret variable in polynomial masking [180, 114] is given by an interpolation polynomial in the Lagrange form, such masking scheme can be obtained by considering all elements in $\mathbf{L}$ to be public Lagrange coefficients.

Algorithm 2 depicts the procedure `IPMask`() to convert a variable X into the IP masked domain as two vectors $(\mathbf{L}, \mathbf{R})$ of size n. The function `rand`() returns a random element in $\mathbb{F}_{2^8}$, whereas the function `randNonZero`() returns a random element in $\mathbb{F}_{2^8} \setminus \{0\}$. The function `IPUnmask`() to convert a masked variable $(\mathbf{L}, \mathbf{R})$ of size n back to X consists in calculating the inner product $X = \langle \mathbf{L}, \mathbf{R} \rangle$.

Algorithm 2 Masking a variable: $(\mathbf{L}, \mathbf{R}) \leftarrow$ `IPMask`(X)

```
Input: variable $X \in \mathbb{F}_{2^8}$
Output: masked variable $(\mathbf{L}, \mathbf{R})$
Ensure: $X = \langle \mathbf{L}, \mathbf{R} \rangle$
  $L_1 \leftarrow$ randNonZero()
  for $i = 2$ to $n$ do
    $L_i \leftarrow$ randNonZero(); $R_i \leftarrow$ rand()
  end for
  $R_1 \leftarrow (X \oplus \bigoplus_{i=2}^n L_i \otimes R_i) \otimes L_1^{-1}$
```

4.3.1.1 Operations in the masked domain

After introducing how to convert variables between $\mathbb{F}_{2^8}$ and the IP masked domain, we need to provide a set of higher-level functions that allows us to operate directly on the masked variables. In order to fulfill our security requirements, computations regarding the left vector $\mathbf{L}$ of masked variables should be executed in the left processor P_{L}, whereas calculations regarding $\mathbf{R}$ should be carried out in the right processor P_{R}. Moreover, the condition that elements of the vector $\mathbf{L}$ are different from zero must be inherited by all operations in order to avoid output masked values from being biased.

In the following we make use of a special operation called `IPHalfMask`(), which on input a variable X and a vector $\mathbf{L}$ calculates the corresponding vector $\mathbf{R}$ such that $X = \langle \mathbf{L}, \mathbf{R} \rangle$. It is thus a simplified version of Algorithm 2 for which the left vector $\mathbf{L}$ is already given and thus elements L_i do not need to be sampled.

Another operation that will be often used is `IPRefresh`(). This operation, depicted in Algorithm 3, takes as input a masked variable $(\mathbf{L}, \mathbf{R})$ and returns a new one $(\mathbf{L}', \mathbf{R}')$ such that $\langle \mathbf{L}, \mathbf{R} \rangle = \langle \mathbf{L}', \mathbf{R}' \rangle$. The purpose of the refreshing is to pump new randomness into the masking scheme. Algorithm 3 is tailored particular to work for the field $\mathbb{F}_{2^8}$. For a generalization we refer the reader to [93].

Algorithm 3 Refresh vector: $(\mathbf{L}', \mathbf{R}') \leftarrow$ `IPRefresh`$(\mathbf{L}, \mathbf{R})$

Input: vector $\mathbf{L}$ in processor P_{L}, vector $\mathbf{R}$ in processor P_{R}
Output: vector $\mathbf{L}'$ in processor P_{L}, vector $\mathbf{R}'$ in processor P_{R}
Ensure: $\langle \mathbf{L}, \mathbf{R} \rangle = \langle \mathbf{L}', \mathbf{R}' \rangle$

P_{L}		P_{R}
$\mathbf{A} \in_R \mathbb{F}_{2^8}^n$		
$\mathbf{L}' = \mathbf{L} \oplus \mathbf{A}$	$\xrightarrow{\mathbf{A}}$	$X = \mathtt{IPUnmask}(\mathbf{A}, \mathbf{R})$
	$\xleftarrow{X}$	
$\mathbf{B} = \mathtt{IPHalfMask}(X, \mathbf{L}')$	$\xrightarrow{\mathbf{B}}$	$\mathbf{R}' = \mathbf{R} \oplus \mathbf{B}$

Although not clearly specified in Algorithm 3, it is necessary that the vector $\mathbf{A}$ sampled by P_{L} is such that the resulting elements of $\mathbf{L}'$ are non-zero. In other words, we need to ensure that $A_i \neq L_i$ for all $1 \leq i \leq n$. Details on how to implement this step efficiently, in constant time and flow are given in Section 4.3.3.

Addition. The procedure `IPAdd`() to calculate the addition of two masked variables is depicted in Algorithm 4. This algorithm requires a three vector additions, two joint executions of `IPRefresh`(), one of `IPUnmask`(), and one of `IPHalfMask`().

Algorithm 4 Masked addition: $(\mathbf{X}, \mathbf{Y}) \leftarrow \texttt{IPAdd}((\mathbf{L}, \mathbf{R}), (\mathbf{K}, \mathbf{Q}))$

```
Input: vectors $\mathbf{L}$ and $\mathbf{K}$ in processor $P_\mathsf{L}$, vectors $\mathbf{R}$ and $\mathbf{Q}$ in processor $P_\mathsf{R}$
Output: vector $\mathbf{X}$ in processor $P_\mathsf{L}$, vector $\mathbf{Y}$ in processor $P_\mathsf{R}$
Ensure: $\langle \mathbf{X}, \mathbf{Y} \rangle = \langle \mathbf{L}, \mathbf{R} \rangle \oplus \langle \mathbf{K}, \mathbf{Q} \rangle$
        $P_\mathsf{L}$                                              $P_\mathsf{R}$
              $\xleftrightarrow{(\mathbf{A},\mathbf{B}) \leftarrow \texttt{IPRefresh}(\mathbf{K},\mathbf{Q}\oplus\mathbf{R})}$
              $\xleftrightarrow{(\mathbf{C},\mathbf{D}) \leftarrow \texttt{IPRefresh}(\mathbf{L}\oplus\mathbf{K},\mathbf{R})}$
              $\xleftrightarrow{Z \leftarrow \texttt{IPUnmask}(\mathbf{C},\mathbf{D})}$
              $\xleftrightarrow{\mathbf{Y} \leftarrow \texttt{IPHalfMask}(Z,\mathbf{A})}$
        $\mathbf{X} = \mathbf{A}$                                   $\mathbf{Y} = \mathbf{Y} \oplus \mathbf{B}$
```

Notice that it might be the case that the component $\mathbf{L} \oplus \mathbf{K}$ in the second execution of `IPRefresh`() has elements equal to zero. While this is a source of first-order leakage in IP masking, i.e. the probability $\Pr(Z = 0|(L_i \oplus K_i) = 0)$ is twice than that for any other value of Z, it is in this particular case not exploitable by an attacker. This is because $\Pr(\langle \mathbf{X}, \mathbf{Y} \rangle | Z = 0)$ is uniformly distributed, i.e. knowing that the intermediate value Z is zero does not give any information about the sensitive output value $(\mathbf{X}, \mathbf{Y})$.

Addition of a constant. The procedure `IPAddConst`() to add a constant $Z \in \mathbb{F}_{2^8}$ to a masked variable $(\mathbf{L}, \mathbf{R})$ can be carried out more efficiently than Algorithm 4. Let $(\mathbf{L}, \mathbf{R})$ and Z be the input operands, and $(\mathbf{X}, \mathbf{Y})$ the output masked variable. Addition of a constant can be simply calculated by letting $\mathbf{X} = \mathbf{L}$ and $\mathbf{Y} = \mathbf{R}$, except for the first element $Y_1 = (R_1 \oplus Z) \otimes L_1^{-1}$.

Multiplication. The procedure `IPMult()` to calculate the multiplication of two masked variables is depicted in Algorithm 5. This algorithm requires $2n^2$ initial field multiplications, one execution of `IPRefresh`() with input/output vectors of size n^2, one execution of `IPUnmask`() with input vectors of size $n^2 - n$, one execution of `IPHalfMask`(), and one final vector addition.

Multiplication by a constant. The procedure `IPMulConst`() to multiply a masked variable $(\mathbf{L}, \mathbf{R})$ by a constant $Z \in \mathbb{F}_{2^8}$ is efficiently computed in IP masking. Let $(\mathbf{L}, \mathbf{R})$ and Z be the input operands, and $(\mathbf{X}, \mathbf{Y})$ be the output masked variable. Multiplication by a constant can be performed by letting $\mathbf{X} = \mathbf{L}$ and calculating $\mathbf{Y} = (R_0 \otimes Z, \ldots, R_n \otimes Z)$. For this operation it is not necessary to execute `IPRefresh`() after `IPMulConst`().

Algorithm 5 Masked multiplication: $(\mathbf{X},\mathbf{Y}) \leftarrow \texttt{IPMult}((\mathbf{L},\mathbf{R}),(\mathbf{K},\mathbf{Q}))$

```
Input: vectors $\mathbf{L}$ and $\mathbf{K}$ in processor $P_{\mathsf{L}}$, vectors $\mathbf{R}$ and $\mathbf{Q}$ in processor $P_{\mathsf{R}}$
Output: vector $\mathbf{X}$ in processor $P_{\mathsf{L}}$, vector $\mathbf{Y}$ in processor $P_{\mathsf{R}}$
Ensure: $\langle \mathbf{X},\mathbf{Y}\rangle = \langle \mathbf{L},\mathbf{R}\rangle \otimes \langle \mathbf{K},\mathbf{Q}\rangle$

  $P_{\mathsf{L}}$                                          $P_{\mathsf{R}}$
  for $i = 1$ to $n$ do                          for $i = 1$ to $n$ do
   for $j = 1$ to $n$ do                          for $j = 1$ to $n$ do
    $\tilde{U}_{(i-1)*n+j} \leftarrow L_i \otimes K_j$          $\tilde{V}_{(i-1)*n+j} \leftarrow R_i \otimes Q_j$
              $\xleftrightarrow{(\mathbf{U},\mathbf{V})\leftarrow\texttt{IPRefresh}(\tilde{\mathbf{U}},\tilde{\mathbf{V}})}$
  $\mathbf{A} = (U_1,\dots,U_n)$                         $\mathbf{B} = (V_1,\dots,V_n)$
  $\mathbf{C} = (U_{n+1},\dots,U_{n^2})$                  $\mathbf{D} = (V_{n+1},\dots,V_{n^2})$
              $\xleftrightarrow{Z\leftarrow\texttt{IPUnmask}(\mathbf{C},\mathbf{D})}$
              $\xleftrightarrow{\mathbf{Y}\leftarrow\texttt{IPHalfMask}(Z,\mathbf{A})}$
  $\mathbf{X} = \mathbf{A}$                                $\mathbf{Y} = \mathbf{Y} \oplus \mathbf{B}$
```

Squaring. The procedure `IPSquare`() can be carried out quite efficiently in the masked domain given that we work over a field of characteristic 2. Let the input masked variable be $(\mathbf{L},\mathbf{R})$. The output masked variable $(\mathbf{X},\mathbf{Y})$ can be calculated by squaring all elements of each vector independently, i.e. $X_i = (L_i)^2$ and $Y_i = (R_i)^2$. The masked squaring operation does not require refreshing the masks, and can be thus carried out with only $2n$ field squarings.

4.3.1.2 Complexity of operations

The complexity of the main operations in the IP masked domain, namely addition and multiplication, is given in Table 4.1. We also provide a comparison with some masked operations that can be implemented at any order d, recently published in the literature for boolean and polynomial masking schemes, namely [188, 180, 114]. The complexity numbers are given in terms of d for all the schemes, where d indicates the number of random values in each masked variable. Recall that in IP masking, this number of random values is given by $d = 2n-1$, with $n \geq 2$. Therefore, note that the following comparison is only valid for odd numbers of d, i.e. any value $d \in \{2,4,\dots\}$ does not allow to generate the necessary vectors $(\mathbf{L},\mathbf{R})$ of size n for the case of IP masking.

As shown in Table 4.1, the complexity of the addition operation in IP masking is slightly larger than in the other proposed methods. This is mainly due to the internal use of the `IPRefresh`() operation which, as opposed to the other masking schemes, involves several field multiplications. However, the

Table 4.1: Complexity of IP masked operations and comparison to d^{th} order boolean masked operations and polynomial masked operations in the literature.

Masked Operation	**Scheme**	**Operations in $\mathbb{F}_{2^8}$** $\oplus$	$\otimes$	x^{-1}	**Rand**
ADDITION	Boolean [188]	$d+1$	-	-	-
	Polynomial [114]	$d+1$	-	-	-
	Polynomial [180]	$d+1$	-	-	-
	Inner Product	$(13d+1)/2$	$3d+3$	3	$(7d+3)/2$
MULTIPLICATION	Boolean [188]	d^2+d+1	$2d^2+2d$	-	$(d^2+d)/2$
	Polynomial [114]	$2d^3+7d^2+d$	$2d^3+5d^2+5d$	-	$2d^2+d$
	Polynomial [180]	$4d^3+8d^2+7d+2$	$4d^3+8d^2+3d$	-	$2d^2+d$
	Inner Product	$(5d^2+12d-9)/4$	$(5d^2+10d+5)/4$	2	$(3d^2+8d-3)/4$

results obtained for the multiplication operation are favourable for IP masking. In particular, both polynomial masked multiplications have complexity $O(d^3)$ while IP masked multiplications have complexity $O(d^2)$. The boolean masked multiplication has a similar complexity but, as we will show in the next sections, the masking construction itself leaks considerably more information that the inner product construction.

4.3.2 Security Evaluation

In this section we evaluate the SCA resistance of IP masking and compare it to that of other masking schemes that can be implemented at any order, e.g. boolean masking and polynomial masking. We focus the analysis on the masking schemes themselves, i.e. we analyze the leakage of the shares of one masked value. We will show in the next section that the security relevant properties of IP masking carry over to the basic operations in the masked domain.

Attack order. We begin the evaluation by deriving the minimum order for an attack against IP masking. For this we need the following definitions:

Definition 1: We say that a variable is sensitive, if it is an intermediate result in an implementation that leaks through side channels, and if it is a function of the input (resp. output), the key and possibly other parameters that are not constant with respect to the key [188].

Definition 2: We say that a masking scheme is d^{th} order SCA secure, if every tuple of d or less shares is independent of the variable that is masked. Accordingly, a masked implementation of an algorithm is d^{th} order SCA secure, if every tuple of d or less intermediate variables is independent of any sensitive variable.

1^{st} *order SCA resistance.* Clearly, IP masking with $n \geq 2$ is 1^{st} order SCA secure. This is a simple consequence of the fact that, even if the value of one of the shares in $\mathbf{L}$ or $\mathbf{R}$ is known (in the worst case one R_i is known to be zero such that $L_i \otimes R_i = 0$), the value of the variable that is masked is still information theoretically hidden by the $\oplus$ with $n-1$ terms that are all uniformly distributed over $\mathbb{F}_{2^8}$.

2^{nd} *order SCA resistance.* IP masking with $n = 2$ is not 2^{nd} order SCA secure. This is because the product of two values is determined to be zero if one of the values is zero. Multiplicative masking [13] suffers from the same problem [113]. Suppose that the values of R_1 and R_2 are known to be zero. Then, $L_1 \otimes 0 \oplus L_2 \otimes 0 = s = 0$. This leads to a bias in the distribution $p(S = s|R_1 = r_1, R_2 = r_2)$, and the mutual information $I(s;(R_1, R_2))$ is non-zero.

d^{th} *order SCA resistance.* IP masking with $2n = d+1$ is SCA secure up to $n-1^{th}$ (or $\frac{d+1}{2} - 1^{th}$) order, but not secure against n^{th} (or $\frac{d+1}{2}^{th}$) order SCA. Following the above examples, as long as the product of one pair $(L_i, R_i), i \in \{1, \ldots, n\}$ is unknown, the value of the variable s that is masked is still information theoretically hidden. On the other hand, if $\forall i \in \{1, \ldots, n\}$ the value of R_i is known to be zero, then the value of s is known to be zero. However, the probability that this case occurs is small and decreases rapidly with increasing n. More precisely, it is (2^{-8n}).

In summary, IP masking with $2n = d + 1$ can, in theory, be broken by a n^{th} order SCA. On the other hand, similar to polynomial masking, it creates a much more complex relation between the shares than boolean masking, which is known to be more difficult to exploit. Hence, we expect IP masking with $2n = d + 1$ to provide much higher security in practice than boolean masking of order $d + 1$, i.e. with the same number of random masks. Following this line, we opt to consider the leakage of all $2n$ or $d + 1$ shares in the following analysis, since an attack exploiting all shares is more powerful in an information theoretic sense, unless the noise levels are extremely high.

In polynomial masking half of the shares are non-zero public constants and the other half are random and secret masks. In particular, there is no direct correspondence to the notion of a masked variable. In what follows we refer only to the random and secret shares, and their number determines the masking order. For example, polynomial masking of order $d - 1$ uses d random and secret shares, and can theoretically be broken by a d^{th} order SCA. We will compare polynomial masking of order $d - 1$ with boolean masking of order d ($d + 1$ shares, d masks) and with IP masking of order $2n = d + 1$ ($d + 1$ shares, d masks). One could expect IP masking with $2n = d + 1$ to provide a similar level of security as polynomial masking of order $d - 1$, i.e. both schemes should provide similar security when they use the same number of random and secret masks.

Information Leakage. As motivated and done in previous works [206, 100, 207, 180], we use the mutual information between a variable and the leakage of all shares of its masked representation as a figure of merit. We estimate it using simulations. For IP masking, we set $n = 2$ and let $R_2 \in_R \mathbb{F}_{2^8}$ and $L_1, L_2 \in_R \mathbb{F}_{2^8} \setminus \{0\}$ such that $S = L_1 \otimes R_1 \oplus L_2 \otimes R_2$. Boolean masking uses $d+1$ shares $(M_1, \ldots, M_d, V)$ where the $M_i \in_R \mathbb{F}_{2^8}$ and V is computed such that $S = M_1 \oplus \ldots \oplus M_d \oplus V$ holds. We evaluate boolean masking for $d \in \{1, 2, 3\}$. Polynomial masking uses d shares $(Y_1, \ldots, Y_d)$ with $Y_i \in_R \mathbb{F}_{2^8}$ and d public Lagrange coefficients $(\beta_1, \ldots, \beta_d)$ with $\beta_i \in_R \mathbb{F}_{2^8} \setminus \{0\}$ and pairwise distinct [114]. We evaluate polynomial masking for $d \in \{2, 3\}$.

To quantify the amount of information leaked, we need to model the relation between the value of a variable and its physical leakage. We follow the approach that is usual in the literature [100, 180, 207]: we model that a variable leaks its Hamming weight, that each share leaks independently of all other shares, and that the leakage of each share is affected by independent Gaussian noise. The latter serves to mimic the noise effects that affect physical measurements. Putting this together, we model the leakage of IP masking as

$$\mathrm{Leak}(\mathbf{L}, \mathbf{R}) = (\mathrm{HW}(L_1) + n_1, \mathrm{HW}(R_1) + n_2, \mathrm{HW}(L_2) + n_3, \mathrm{HW}(R_2) + n_4),$$

the leakage of boolean masking as

$$\mathrm{Leak}(M_1, \ldots, M_d, V) = (\mathrm{HW}(M_1) + n_1, \ldots, \mathrm{HW}(M_d) + n_d, \mathrm{HW}(V) + n_{d+1})$$

and the leakage of polynomial masking as

$$\mathrm{Leak}(Y_1, \ldots, Y_d) = (\mathrm{HW}(Y_1) + n_1, \ldots, \mathrm{HW}(Y_d) + n_d)$$

where the n_i are independent Gaussian variables with mean zero and standard deviation σ. The mutual information is then $I(S; \mathrm{Leak}(\mathbf{L}, \mathbf{R}))$, $I(S; \mathrm{Leak}(M_1, \ldots, M_d, V)$ resp. $I(S; \mathrm{Leak}(Y_1, \ldots, Y_d))$. The number of measurements that a Template Attack [63], i.e. the worst case scenario of a profiled attack, requires to achieve a given success probability is directly related to this mutual information via $c \cdot I(\cdot;\cdot)^{-1}$, where the constant c is related to the success probability [207].

Figure 4.1 shows plots of the mutual information ($\log_{10}$) between S and the information leaked by all shares of its masked representation, over increasing noise levels σ, for all masking schemes considered[1] [2].

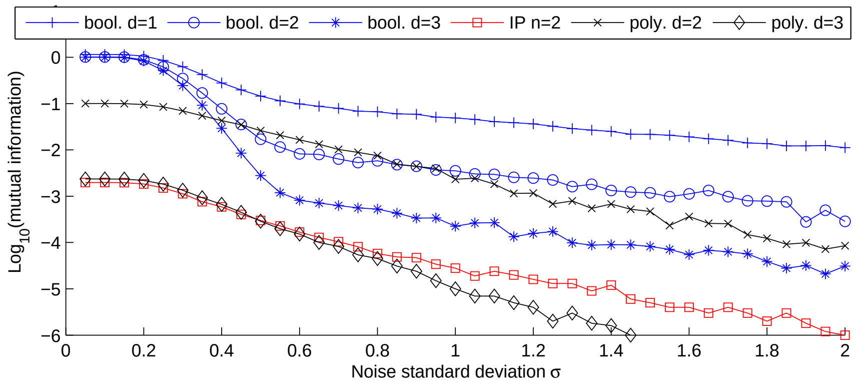


Figure 4.1: Mutual information ($\log_{10}$) over increasing noise standard deviation σ for different masking schemes.

The figure shows that IP masking with $n = 2$ leaks consistently less than boolean masking with $d \in \{1, 2, 3\}$ across the range of tested noise levels, which confirms our expectation. The advantage is more pronounced for low noise levels, where e.g. for $\sigma = 0.2$ the information leakage of IP masking is about 2.5 orders of magnitude(!) smaller than that of boolean masking. As expected, polynomial masking with $d = 2$ leaks consistently more than IP masking with $n = 2$. Polynomial masking with $d = 3$ provides a level of security very similar to IP masking with $n = 2$ for low noise levels. However, contrary to what one could expect, for high noise levels, polynomial masking with $d = 3$ leaks less than IP masking with $n = 2$. There are several possible explanations for this observation. For instance, IP masking with $n = 2$ involves two field multiplications while polynomial masking with $d = 3$ involves three field multiplications, i.e. the algebraic complexity of the masking is greater. Furthermore, IP masking with $n = 2$ is 1^{st} order SCA secure while polynomial masking with $d = 3$ is 2^{nd} order SCA secure. It is known that leakage of lower order is easier to exploit, in particular with increasing noise [207]. We leave the careful analysis of the observed difference in information leakage as an open question for future research.

[1] Note that the mutual information values we computed for boolean masking are consistent with Figure 1 in [100] and Figure 3 in [180]. One has to take into account that the Y-axis in those figures is erroneously labeled $\log_{10}$ while it should be $\log_n$.

[2] For polynomial masking with $d = 3$, reasonably accurate estimation of the mutual information values for high noise levels is beyond our computational budget.

Discussion. Our evaluation shows that IP masking with $n = 2$ provides high security even if there is little noise. However, although the simulated scenario (Hamming weight leakage, independent leakage of each share, Gaussian noise) is standard in the practice-oriented literature, it is synthetic and in particular meets the requirement of the masking schemes for independent leakage perfectly. It can be hard to achieve this for real-world implementations that are affected by effects such as coupling (we show in the next section that glitches do not affect the security of IP masking). Clearly, our evaluation does not allow to blindly assume that an implementation of IP masking is secure. What it shows is the level of security that a secure implementation of IP masking can provide. An interesting topic for future research is to analyze the security provided by a real-world implementation, and to analyze how violating a requirement, e.g. independent leakage, affects practical security.

4.3.3 Performance Evaluation

In this section we evaluate the performance and correctness of IP masking. We provide a general overview on how to implement the IP masking building blocks on an 8-bit embedded platform, and describe how to use them to protect an implementation of the AES.

4.3.3.1 Implementation of masked operations

The 8-bit Atmel AVR ATMega128 is chosen as target platform. This device provides an advanced RISC architecture with 133 low-level instructions and it offers 128 kBytes of flash program memory and 4 kBytes of internal SRAM. The independent side channel leakage required by our model is in our implementation achieved by temporal separation, i.e. instead of using two physically separated processors P_{L} and P_{R}, we use a single 8-bit processor and we ensure independent leakage by not overlapping their respective operations.

For the sake of optimization, we have implemented all operations in assembly language. The ATMega128 does not provide an internal random number generator to implement the `rand`() and `randNonZero`() functionalities. Therefore, and only for the purposes of evaluating the implementation, the required random bytes are provided to the microcontroller externally previous to the encryption process. We note that a real-world implementation of IP masking would require a platform capable of generating such randomness internally.

Addition in $\mathbb{F}_{2^8}$ is carried out in a single clock cycle via the available XOR instruction, whereas the rest of field operations (multiplication, inversion, raisings to the power of 2) are implemented via lookup tables, requiring a total of 1,536 bytes in program memory. Besides the squaring, we have also implemented as lookup tables the rising to the powers of 4 and 16 required in the power function of the AES `SubBytes` step. On devices with limited program memory these raisings can be alternatively carried out by consecutive squarings, effectively saving 512 bytes of program memory.

Special care has been taken in order to make the implementation not only time-constant, but flow-constant i.e. conditional execution paths, which can be a potential source of side channel leakage, have been avoided. A typical example of a function with conditional execution is the multiplication in $\mathbb{F}_{2^8}$ using log/alog tables. This method only works when both input operands are different than zero; otherwise, the result of the multiplication must be equal to zero. Implementing this routine in constant flow requires to calculate the potential outputs of *all* conditional paths, and thus it ends up requiring 22 clock cycles.

Worth mentioning is the implementation of the first part of Algorithm 3 for mask refreshing, namely sampling a vector $\mathbf{A}$ such that $A_i \neq L_i$ for $1 \leq i \leq n$. This step is carried out as follows for each element A_i. First, we sample two elements $A'_i \in \mathbb{F}_{2^8}$ and $A''_i \in \mathbb{F}_{2^8} \setminus \{0\}$. If $A'_i \neq L_i$ we simply set $A_i = A'_i$; otherwise, we assign $A_i = A'_i \oplus A''_i$. Independently of the sampled values A'_i and A''_i, this conditional statement ensures that i) the final value A_i is different than L_i, and ii) the final value of A_i is uniformly distributed over $\mathbb{F}_{2^8}$. Needless to say, such implementation is also performed in constant flow execution to prevent conditional execution branches.

4.3.3.2 Applying IP masking to the AES

The AES operates on a 128-bit state, arranged as a 4 x 4 array of bytes in $\mathbb{F}_{2^8}$, and denoted by $(S_{i,j})_{0 \leq i,j \leq 3}$. During encryption the AES state goes through a series of rounds, each performing a total of four individual transformations: `AddRoundKey`, `SubBytes`, `ShiftRows`, and `MixColumns`. In order to secure AES with IP masking we must ensure that we are working on the same field $\mathbb{F}_{2^8} \equiv \mathbb{F}_2[x]/(x^8 + x^4 + x^3 + x + 1)$, i.e. multiplication in $\mathbb{F}_{2^8}$ must be modulo the irreducible polynomial $x^8 + x^4 + x^3 + x + 1$ as defined in [6]. After this, each byte element of the state $S_{i,j}$ needs to be transformed and operated in the IP masked domain, correctly adjusting the round transformations to the masking scheme.

The `SubBytes` step is computed by applying two consecutive transformations to each state byte: i) the non-linear calculation of the inverse in $\mathbb{F}_{2^8}$, and ii) an affine transformation over $\mathbb{F}_2$. The calculation of the inverse has a multiplicative structure, making it difficult to combine with the additive component of IP masking, while the bit manipulations of the affine transformation cannot be easily adapted for the field $\mathbb{F}_{2^8}$. The full `SubBytes` step can be calculated using the following equation defined over $\mathbb{F}_{2^8}$, for a given input state byte X:

$$\begin{aligned}\texttt{SubBytes}[X] = {} & \{05\} \otimes X^{254} \oplus \{09\} \otimes X^{253} \oplus \{f9\} \otimes X^{251} \oplus \{25\} \otimes X^{247} \oplus \\ & \{f4\} \otimes X^{239} \oplus X^{223} \oplus \{b5\} \otimes X^{191} \oplus \{8f\} \otimes X^{127} \oplus \{63\}.\end{aligned}$$

Note that this equation requires a lot of operations (specially multiplications) to calculate the different powers of X. As masked multiplications are much more inefficient than masked squarings or masked additions, we need to find an alternative way to compute this transformation.

Our proposed solution is to perform the inner `SubBytes` transformations (inverse and affine transformation) separately. Specifically, we first compute the inverse using the following power function from [188]:

$$\text{Inverse}[X] = X^{254} = \left((X^2 \otimes X)^4 \otimes (X^2 \otimes X)\right)^{16} \otimes (X^2 \otimes X)^4 \otimes X^2.$$

As shown in [188], this equation computes the inverse using a lower bound of 4 multiplications, plus 7 squarings. The affine transformation, linear in $\mathbb{F}_2$, can be defined as the following polynomial over $\mathbb{F}_{2^8}$ [165]:

$$\begin{aligned}\text{AffTrans}[X] = {} & \{05\} \otimes X^{128} \oplus \{09\} \otimes X^{64} \oplus \{f9\} \otimes X^{32} \oplus \{25\} \otimes X^{16} \oplus \\ & \{f4\} \otimes X^{8} \oplus \{01\} \otimes X^{4} \oplus \{b5\} \otimes X^{2} \oplus \{8f\} \otimes X \oplus \{63\},\end{aligned}$$

requiring 7 squarings, 8 additions, and 7 multiplications with a constant.

The `MixColumns` transformation operates on the AES state column-by-column. In particular, each of the bytes in the $0 \leq j \leq 3$ columns is replaced as:

$$\begin{aligned}s'_{0,j} &= \{02\} \otimes s_{0,j} \oplus \{03\} \otimes s_{1,j} \oplus s_{2,j} \oplus s_{3,j} \\ s'_{1,j} &= s_{0,j} \oplus \{02\} \otimes s_{1,j} \oplus \{03\} \otimes s_{2,j} \oplus s_{3,j} \\ s'_{2,j} &= s_{0,j} \oplus s_{1,j} \otimes \{02\} \oplus s_{2,j} \oplus \{03\} \otimes s_{3,j} \\ s'_{3,j} &= \{03\} \otimes s_{0,j} \oplus s_{1,j} \oplus s_{2,j} \oplus \{02\} \otimes s_{3,j}.\end{aligned}$$

From these equations it follows that this step can be implemented using a total of 12 masked additions and 8 masked multiplications by a constant, for each column. In [79], the authors of AES suggest a more efficient way

Table 4.2: Performance evaluation (in clock cycles) of AES round transformations and AES encryption with IP masking scheme with $n = 2$.

`AddRoundKey`	`SubBytes` (Inverse)	`SubBytes` (Aff.Transf.)	`ShiftRows`	`MixColumns`	**Full AES**
8,796	45,632	72,128	200	27,468	1,912,000

to compute the `MixColumns` step by using the so-called `xtime` tables. Such technique takes advantage of the fact that field addition is more efficient than field multiplication in general purpose processors. Due to this, they suggest an alternative approach that requires 15 additions and 4 multiplications by 02, which can be simply performed as table lookups. This technique is however not optimal for IPmasking since masked multiplications by a constant are way more efficient than masked additions. In other words, the efficiency in our scheme for the `MixColumns` transformation is achieved by minimizing the number of masked additions.

As shown in Table 4.2, our implementation requires around $1.9 \cdot 10^6$ clock cycles to perform a protected AES encryption (including on-the-fly key schedule calculation).

We stress that these results should not be simply taken as an indicator to judge the practicality of IP masking, as they are obtained using a legacy general-purpose device without any type of hardware enhancements. If multiplication in $\mathbb{F}_{2^8}$ was available in the instruction set of the controller our timing for AES encryption would be instantly reduced to less than a million cycles. This could be achieved e.g. by providing instruction set extensions to the target device.

4.4 A 1st-order Attack Against (IP) Masking

A recent work by Prouff, Rivain, and Roche [179] has identified 1st-order leakage in the `IPRefresh`() and `IPAdd`() operations of the IP Masking scheme [22]. The result of this evaluation implies that the original claim of generic $(n-1)$th-order SCA security for any order $d = 2n - 1$ is effectively reduced to 1st-order security due to a flaw in these operations.

The 1st-order leakage comes from the computation of intermediate variables using the inner product and can be exhibited as follows. Let $\mathbf{A} = (A_1, \ldots, A_n)$ and $\mathbf{B} = (B_1, \ldots, B_n)$ be random vectors with elements in $\mathbb{F}_{2^8}$, and let $\mathbf{R} = (R_1, \ldots, R_n)$ be a random vector with elements in $\mathbb{F}_{2^8} \setminus \{0\}$. We denote a and

b the variable $a = \langle \mathbf{A}, \mathbf{R} \rangle$ and $b = \langle \mathbf{B}, \mathbf{R} \rangle$ resulting from the inner product calculations.

A study of the amount of information that a leaks on b yields the following:

$$Pr[A = a|B = b] = \begin{cases} \frac{1}{256} + \frac{1}{256(255)^{n-2}} & \text{if } a \text{ is zero} \\ \frac{1}{256} - \frac{1}{256(255)^{n-1}} & \text{if } a \text{ is non-zero} \end{cases}$$

This bias identified shows that when b is zero, the variable a is more likely to be zero than any other value in $\mathbb{F}_{2^8}$. Note however that this bias is increasingly small as the size n of the vectors in IP masking increases.

Such a bias can be found in our proposed `IPRefresh`() and `IPAdd`() algorithms. Focusing on the former, there exists an intermediate calculation of the form $x = \langle \mathbf{A}, \mathbf{R} \rangle$ given an input value $b = \langle \mathbf{B}, \mathbf{R} \rangle$. Therefore, knowing that the intermediate value x is zero, gives information about the sensitive masked variable b that is input to the algorithm. A similar construction is also found in `IPAdd`().

This bias is in contradiction to our claim in *Definition 2* that every tuple of d or less shares is independent of the variable that is masked (see Sect. 4.3.2). However, it does not invalidate the security analysis of the information leakage of IP masking, as this depends exclusively on the masking construction. Further research is required to determine up to which extent the `IPRefresh`() and `IPAdd`() algorithms can be re-designed to avoid such intermediate leakages, as well as how complex is for a side channel adversary to practically exploit this leakage.

4.5 Conclusions

In this chapter we have surveyed the landscape of masking based techniques to protect implementations of block ciphers. Starting from the earlier proposals to counter classic univariate attacks, the research scope has gradually moved towards the design of masking schemes at any order. The review of works in this research area poses an evident example of how the state-of-the-art advances driven by an active arms-race between attacks and countermeasures.

As a contribution, we have presented an efficient higher-order masking scheme based on the inner product construction. While non-linear masking constructions have the potential to yield better resistance against power analysis attacks than e.g. boolean masking, their complexity has so far limited their feasibility in practical settings. Despite the weaknesses identified in two of our constructions, our work represents an important step towards achieving a practical masking scheme at any order using non-linear constructions.

Chapter 5

Enabling Privacy in Embedded Design

A Use Case for Electronic Toll Pricing

In this chapter we focus on aspects related to the integration of privacy in embedded services and applications. We briefly enumerate some characteristics and practical drawbacks commonly found in current systems relying on the collection of personal data. After this, we motivate and illustrate how the use of PETs can successfully lead to the design of alternative systems offering stronger privacy guarantees. The core of the chapter is devoted to our contributions in the design of a privacy-preserving system for Electronic Toll Pricing.

5.1 Introduction

Integrating privacy protection into a system is a complex task. From a legal perspective, service providers have to follow a series of rules described in regulations, policies, and codes of conduct. As legislation on data protection varies from country to country, this results in different views on the actual definitions of privacy protection. Nevertheless, one can often identify a series of common drawbacks. Even though the concept of data minimization is typically mentioned, there are no effective mechanisms to prioritize privacy-preserving technologies over more invasive ones. In other words, it is rather challenging to ensure the amount of collected data is kept to a minimum: providers can simply justify that they only collect whatever is strictly necessary for the provisioning of the service. While in practice clients agree to data collection by accepting the terms of service, these are often not trivial to understand and basically offer no opt-out alternatives to concerned users. Finally, most of the enforcement mechanisms in privacy protection legislation are reactive, i.e. based on sanctioning data controllers once an infraction is found.

In such scenarios, the privacy guarantees of the system exclusively rely on the data controller playing the role of a *trusted* entity. As all collected personal data is assumed to be fairly processed and safely stored by the service provider, the privacy of users is safeguarded. However, one can argue whether this assumption is valid in the real world. After all, service providers have incentives to exploit collected data, e.g. to obtain advantageous business positions. Even if the data is not misused, its centralized storage represents a single point of failure in the system. Rogue employees, external adversaries or even law enforcement agencies can try to access these databases. Damage to users cannot be reverted when a data breach occurs, as their data has already been leaked at that point.

5.2 Privacy-Preserving Systems

The development of systems integrating both privacy and data protection starting from their *design* stage is sometimes referred to as *Privacy-by-Design*. This term, originally coined by Ann Cavoukian [61], has been proposed by policy makers to denote a series of principles to achieve privacy-preserving services. While these principles provide a comprehensive view of the challenges in privacy protection, they are still too vague when actual practical designs have to be implemented [119].

The interpretation of privacy can give rise to multiple meanings [81]. In this dissertation we define privacy protection as the prevention of personal data from being disclosed to external parties. This interpretation is suitable when considering information based services such as monitoring systems. Users should not be forced to trust their personal data to service providers. Rather differently, service providers (or other external parties) should be perceived as *potential adversaries* within the system. Departing from this view, the privacy properties of the system can be laid out at design time. The use of PETs can help in achieving the fulfillment of these requirements.

Privacy-Enhancing Technologies encompass a set of technologies to enforce the protection of personal information. In particular, they provide a series of properties that can lead to the protection of a user's identity or actions when using resources or services from one (or multiple) parties. *Anonymity* allows users to communicate with a recipient without disclosing their identity;

unlinkability allows users to communicate with a recipient multiple times while ensuring these interactions cannot be linked together, i.e. traced back to the same user; and *pseudonimity* allows users to hide their identity within a subset of interactions under a certain pseudonym.

An exemplary application of how PETs guarantee privacy properties for the provision of a service is given by anonymous E-Cash [64]. Proposed by David Chaum in 1982, the goal of this system is to provide electronic means to mimic the functionalities of traditional paper cash. To this end, the considered model has three interacting parties: a customer, a merchant, and a bank. A customer can withdraw a wallet of electronic coins from a bank, and he can later spend these coins by purchasing goods from a merchant. At the end of the purchase, the merchant can deposit all collected coins into his bank account.

From a privacy perspective, anonymous E-Cash satisfies similar properties found in “off-line” real-world payments, namely anonymity and unlinkability. The former is achieved by preventing a merchant from learning the identity of a user during or after a purchase transaction. The later impedes a merchant from determining whether two or more purchase transactions correspond to the same or different user(s), as well as impeding the bank from concluding whether coins withdrawn by a user correspond to coins deposited by a merchant.

From a security perspective, these privacy properties should obviously not conflict with other generic security requirements of “off-line” real-world payments, for instance prevention of coin double-spending and coin double-depositing. Rather differently, anonymous E-Cash systems are designed such that system misuse invalidates privacy assurances. For instance, users that double-spend a coin are automatically detected and their anonymity is revoked. Vendors that attempt to double-deposit a coin can be similarly detected and identified.

Despite its interesting properties, anonymous E-Cash relies on rather complex building blocks that have limited its feasibility in practical settings. User payment tokens are portable hardware devices, resource constrained due to their size, and therefore not suited to support such complex protocols. However, the technological advances in embedded computing platforms and their current widespread deployment is changing this landscape.

Among our contributions to the area of privacy-preserving systems, we have proposed an efficient and practical anonymous E-Cash suited for mobile phone applications [25]. Our system is a variant of the Compact E-Cash scheme due to Camenisch *et al.* [57] that borrows some ideas from the Direct Anonymous Attestation protocol due to Brickell *et al.* [51]. In particular, we consider a model in which payment tokens combine untrusted but powerful execution platforms (e.g. a mobile phone) with trusted but constrained secure elements (e.g. a SIM card).

As exemplified by anonymous E-Cash, the integration of privacy guarantees within a system is a complex task that requires to balance the use of privacy technologies with multiple aspects. It requires a concise study of the functional requirements of the system, the combination of data minimization techniques with privacy-enhancing technologies, the fulfillment of other, commonly required security properties and assurances, and the implementation, testing, and validation of the design's correctness and feasibility.

In the following we apply all these observations to design a privacy-preserving solution for Electronic Toll Pricing (ETP) applications.

5.3 Use Case: Electronic Toll Pricing

In the field of car telematics, a new series of applications and services known as Intelligent Transport Systems (ITS) are currently being developed and studied. Among these, Electronic Toll Pricing solutions based on satellite localization are of great interest. Two applications based on this system are either commercially available or in the process of being deployed: Pay As You Drive insurances (PAYD) and Public Electronic Toll Pricing. The former is currently being offered by insurance companies all around the world, while the deployment of the latter will soon be a reality in the European Union with the adoption of the European Directive 2004/52/EC by Member States. Even though these systems are managed by different entities - private insurance companies in PAYD, central governments in Public Electronic Toll Pricing - they are both based on the same idea: drivers should pay according to their road usage, as opposed to the current flat yearly fees.

This approach benefits all actors involved. First, users pay an amount of money proportional to the usage of their vehicle. Second, insurance companies are able to offer fairer and targeted tariffs by charging an amount of money proportional to the statistical risk of accidents in roads. Third, governments can achieve mobility reduction in congested roads or cities by charging more money in peak hours. And finally, the environment can benefit from a decrease of the pollution levels due to traffic reduction.

A common requirement for these satellite-based road charging systems consists in the installation of a monitoring device inside the vehicles. The minimum functionalities of this device are to determine its own location and to communicate with entities outside the vehicle. The former requirement can be achieved by using a Global Navigation Satellite System (GNSS) such as GPS, while for the latter a mobile communications network such as GSM can be used.

The most typical and straightforward architecture for Electronic Toll Pricing is depicted in Figure 5.1. It comprises three entities: an On-Board Unit (OBU), a Toll Service Provider (TSP), and a Toll Charger (TC). The OBU is an electronic device installed in vehicles subscribed to an ETP service. It is in charge of collecting GPS data and sending it to the TSP, which is the entity offering the ETP service. At the end of the tax period the TSP processes all location data and derives the fee to be paid by the user. The TSP is responsible for providing vehicles with OBUs and monitoring their performance and integrity. Finally, the TC is the organization (either public or private) that levies tolls for the use of roads and defines the correct use of the system. In agreement with the TC, the TSP establishes prices for driving on each of the roads. Such pricing policy can depend on the type of road (e.g., highways vs. secondary roads), its traffic density, or the time of the day (e.g., rush hours vs. the middle of the night). Additionally, prices can also depend on attributes of the vehicle or of the driver (e.g., low-pollution vehicles, or discounts for retired people).

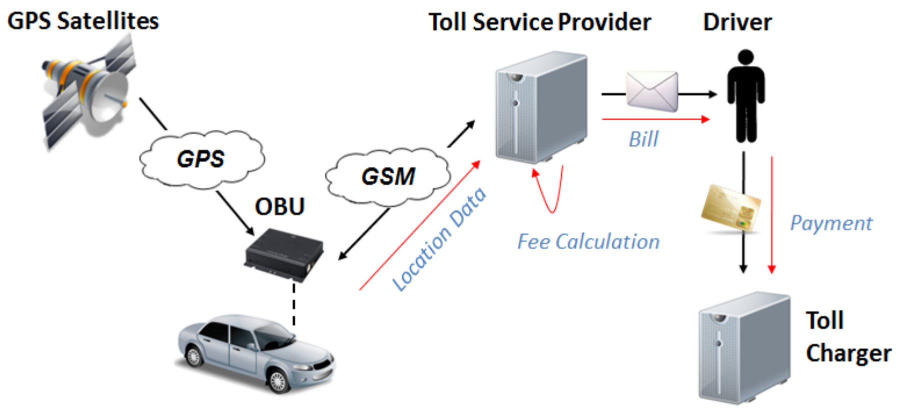


Figure 5.1: Straightforward model for Electronic Toll Pricing.

The downside in terms of privacy of this architecture is obvious: an external server is able to collect precise location data of all vehicles in the system. Note that a straightforward approach such as anonymizing the location data is in this case fruitless, as it has been shown that analysis of location data allows to re-identify drivers [122, 118, 145]. Therefore this application provides an excellent use case to study and develop the integration of privacy-preserving techniques based on data minimization.

5.3.1 PrETP: Privacy-Preserving Electronic Toll Pricing

Publication data

J. Balasch, A. Rial, C. Troncoso, C. Geuens, B. Preneel, and I. Verbauwhede, "PrETP: Privacy-Preserving Electronic Toll Pricing", In *Proceedings of the 19th Usenix Security Symposium - USENIX Security 2010*, pages 63-78. USENIX Association, 2010.

Contribution

Shared work between authors.

Our contribution is the design of PrETP, a privacy-preserving ETP system in which, without making impractical assumptions, OBUs i) compute their road fees locally, and ii) prove to the service provider that they carry out correct computations while revealing the minimum amount of location data. PrETP employs a cryptographic protocol, Optimistic Payment (OP), in which OBUs send along with the final fee commitments to the locations and prices used in the fee computation. These commitments do not reveal information on the locations or prices to the service provider. Moreover, they ensure that drivers cannot claim that they were at any other position, nor used different prices, from the ones used to create the commitments.

In order to check the correctness of the committed values, we rely on the TSP having access to a proof (e.g., a photograph taken by a road-side radar or a toll gate) that a car was at a specific point at a particular time, as previously suggested in [83, 173]. Upon being challenged with this proof, the OBU must respond with some information proving that the location point where it was spotted was correctly used in the calculation of the final fee. To this end, it opens the commitment containing this location, thus revealing *only* the location data and the price at the instant specified in the proof. This information suffices for the provider to verify that correct input data (location and price) was used to calculate the fee.

We perform a holistic analysis of PrETP. Along with the security, privacy, and legal analyses, we build an OBU prototype on an embedded platform, as well as a TSP prototype on a commodity computer, and we thoroughly test the performance of both using real world collected data. The result of our experiments confirms that our protocol can be executed in real time in an OBU assembled with generic, off-the-shelf components.

5.3.1.1 System Overview and Related Work

The design of PrETP builds on the PriPAYD solution originally devised by Troncoso *et al.* [215] in the context of car insurance. This system is depicted in Figure 5.2. The main difference of PriPAYD with respect to straightforward ETP models is that all processing and storage of location data is done exclusively in the OBU, without transmitting this information to any external entity. The OBU is responsible for computing the fees corresponding to each trip. The results are then aggregated in order to obtain the final fee, which is later sent to the insurance company. Note that in this model the "intelligence" of the system is put on the OBU rather than in the TSP. This means that both digital road maps and a valid taxing policy need to be locally stored for computations, while giving the insurance company the option to send secure updates.

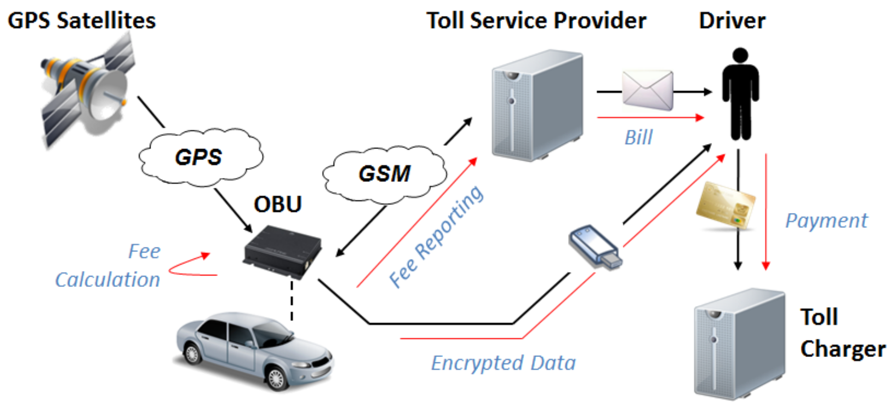


Figure 5.2: PriPAYD-based architecture for Electronic Toll Pricing.

PriPAYD is designed in such a way that the driver is the only party that has access to his own location data. For this purpose, the authors of [215] propose the use of a dedicated interface from where location data stored in the OBU can be securely retrieved by a portable memory device (e.g., a USB stick). Three conditions need to be met: i) the location data records must be signed by the OBU, ii) an authenticated encryption mechanism must be applied over the location data in order to ensure confidentiality and integrity, and iii) the encryption key used by this mechanism must be known exclusively by the OBU and the driver.

One of our contributions not covered in the core of this dissertation is the demonstration of a practical and functional ETP system based on PriPAYD [27]. In particular, we build an OBU that guarantees real-time processing of location data while minimizing the overheads required to ensure security and privacy. The performance of our software-based prototype is tested and proves that the deployment of a privacy-friendly solution can be achieved within a minimum cost increment compared to existing ETP solutions offering only soft privacy guarantees. An extended and updated version of the original PriPAYD article, including our demonstrator, is further given in [214].

Despite its attractive privacy properties, a limitation of PriPAYD is that it does not provide means for the TSP and the TC to check the correctness of the operations carried out in the OBU. In other words, there is no effective, privacy-preserving solution for the TSP or the TC in detecting tampering attacks, either on the OBU itself or on its interfaces (GPS or GSM). Note that while these tampering attacks are carried out by adversaries in possession of the OBU, they are different than the physical attacks as described in earlier chapters. Their goal is not necessarily the extraction of secret cryptographic keys. Instead, they may target non-cryptographic functionalities of the OBU, e.g. by turning it off, preventing it from communicating, or manipulating the expected processing of location data or fee computations.

In order to overcome these tampering attacks, a line of research has focused on the design of secure multi-party protocols between the TSP and the OBUs that allow TSPs to compute the total fee and detect malicious OBUs while protecting location privacy. An efficient instantiation of this idea is given by VPriv [173]. Its basic idea consists in sending the location data generated by a driver sliced into segments to the TSP, in such a way that it remains hidden among segments from multiple drivers. Then the TSP calculates the subfees (fees of small time periods that add to the final fee) of all segments and returns them to all OBUs. Each OBU uses this information to compute its total fee and, without disclosing any location data, proves to the TSP that the total fee is computed correctly, i.e., by only using the subfees that correspond to the location data input by this particular OBU. Moreover, in order to prevent malicious users from spoofing the GPS signal to simulate cheaper trips, VPriv has an out-of-band enforcement mechanism. This mechanism is based on the use of random spot checks that demonstrate that a vehicle has been at a location at a time (e.g., a photograph taken by a road-side radar). Given this proof, the TSP challenges the OBU to prove that its fee calculation includes the location where the vehicle was spotted.

The protocol proposed in [173] has several practical drawbacks. First, it requires vehicles to send anonymous messages to the server (e.g., by using Tor [89]) imposing high additional costs to the system. Second, the protocol only avoids leaking any additional information beyond what can be deduced from the anonymized database. As the database contains path segments, the TSP can use tracking algorithms to recover paths followed by the drivers [122, 118, 145] and infer further information about them. Third, the scalability of the system is limited by the complexity of the protocol on the client side, as it depends on the number of drivers in the system. Practical implementations require simplifications such as partitioning the set of vehicles into smaller groups, thus reducing the anonymity set of the drivers. Fourth, VPriv only uses spot checks to verify correctness of the location, and thus needs an extra protocol to verify the correct pricing of segments. This extra protocol produces an overhead both in terms of computation and communication complexity.

Our solution, similar to PriPAYD [27], does not require messages between the OBU and the TSP to be anonymous as the computation of the fee is made locally and no personal data is sent to the provider. Thus, no database of personal data is created and we do not need to rely on database anonymization techniques to ensure users' privacy. Further, the OBU's operations depend only on the data it collects, independently of the number of vehicles in the system. Finally, our protocol can be integrated into a stand-alone OBU without the need of external devices to carry out the cryptographic protocols.

Another key advantage of PrETP is that it provides a sound mechanism to protect the interests of both TC and TSP against fraud. Our threat model considers malicious drivers capable of tampering with the internal functionality of the OBU as well as with any of its interfaces. While protection against physical attacks, e.g. side channel or faults, is ultimately required to ensure the overall security of PrETP, in the following we focus on a subset of tampering attacks which may not directly involve the use of cryptography. In particular, we define the security goals of PrETP as the detection of:

Vehicles with inactive OBUs. Drivers should not be able to shut down their OBUs at will to simulate they drove less.

OBUs reporting false GPS location data. Drivers should not be able to spoof the GPS signal and simulate a cheaper route than the actual roads on which they are driving.

OBUs using incorrect road prices. Drivers should not be able to assign arbitrary prices to the roads on which they are driving.

OBUs reporting false final fees. Drivers should not be able to report an arbitrary fee, but only the result from the correct calculations in the OBU.

In order to perform these detections, reliable information about the vehicle's whereabouts is required. We consider that the TC can perform random "spot checks" that are recorded as proof of the time and location where a vehicle has been seen. Such spot checks can be carried out by using an automatic license plate reader, a police control, or even challenging the OBUs using short-range communications. Without loss of generality in this work we assume that the proof is gathered using an automatic license plate reader. This proof can be used to challenge the vehicle's OBU to verify its functioning. In order to be able to respond to this challenge, the OBU slices the trajectories recorded in segments, and computes the subfees corresponding to them, such that these subfees add up to the final fee transmitted to the TSP. For each segment, the TSP receives a payment tuple that consists of a commitment to location data and time, a homomorphic commitment to the subfee, and a proof that the committed subfee is computed according to the policy. These payment tuples, explained in detail in the next section, bind the reported final fee to the committed values such that the OBU cannot claim having used other locations or prices in its computations. Furthermore, they are signed by the OBU to prevent a malicious TSP from framing an honest driver.

The verification process, depicted in Fig. 5.3, is initiated when the TC gathers a proof of location of a vehicle. Then it forwards this information to the TSP, along with a request to check the correct functioning of the vehicle's OBU. To this end, the TSP challenges the OBU to open a commitment containing the location and time appearing in the proof. The TSP verifies that both challenge and response match, for instance as explained in [83, 173], and reports to the TC whether or not the functioning of the OBU is correct. We assume that the TC (e.g., the government in the EETS architecture) is honest and does not use fake proofs to challenge OBUs.

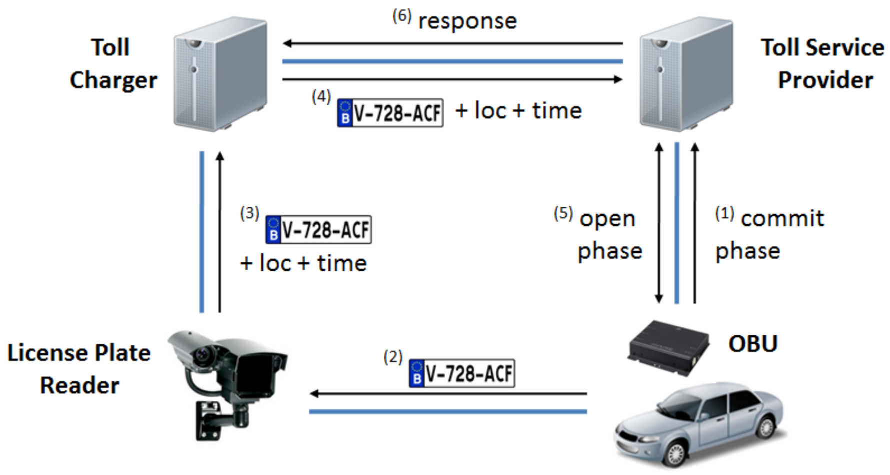


Figure 5.3: PrETP enforcement spot-check model.

To the best of our knowledge, the only protocol that so far employs spot checks to verify both correctness of the location and of the fee calculation is due to Jonge and Jacobs [83]. In this solution, OBUs commit to segments of location data and their corresponding subfees when reporting the total fee to the TSP. They employ hash functions as commitments. Upon being challenged to ratify the information in the spot check, OBUs must provide the hash pre-image of the corresponding segment, and demonstrate that indeed the location was used to compute the final fee.

Jonge and Jacobs' protocol is limited by the fact that using hash-based commitments one cannot prove that the commitments to the subfees add to the total fee. As solution, they propose that the OBU also commits to the subfees corresponding to bigger time intervals following a tree structure. Each tax period is divided into months, each month is divided into weeks, and so forth, and subfees for each month, week, day,... are calculated and committed. Then, instead of asking the OBU to open only one commitment containing the instant specified in TC's proof, the TSP asks the OBU to open all the commitments in the tree that include that instant. This indeed proves that the sum is correct at the cost of revealing much more information to the TSP.

PrETP avoids this information leakage. The reason is that, in our OP scheme, commitments are homomorphic and thus allow TSP to check that the commitments to the subfees add to the total fee without additional data. The use of homomorphic commitments was also proposed and briefly sketched in [83]. However, their scheme does not prevent the OBU from committing to a "negative" price, which would give a malicious OBU the possibility of reducing the final fee by sending only one wrong commitment, thus with an overwhelming probability of not being detected by the spot checks.

5.3.1.2 Optimistic Payment

In this section we sketch the technical concepts necessary to understand the construction of Optimistic Payment, and we outline our efficient implementation of the protocol.

5.3.1.2.1 Technical Preliminaries

Signature Schemes. A signature scheme consists of the algorithms $\mathsf{SigKeygen}$, $\mathsf{SigSign}$ and $\mathsf{SigVerify}$. $\mathsf{SigKeygen}$ outputs a secret key sk and a public key pk. $\mathsf{SigSign}(sk, x)$ outputs a signature s_x of message x. $\mathsf{SigVerify}(pk, x, s_x)$ outputs accept if s_x is a valid signature of x and reject otherwise. A signature scheme must be correct and unforgeable [112]. Informally speaking, correctness implies that the SigVerify algorithm always accepts an honestly generated signature. Unforgeability means that no p.p.t adversary should be able to output a message-signature pair (x, s_x) unless he has previously obtained a signature on x.

Commitment schemes. A non-interactive commitment scheme consists of the algorithms ComSetup, Commit and Open. $\mathsf{ComSetup}(1^k)$ generates the parameters of the commitment scheme $params_{Com}$. $\mathsf{Commit}(params_{Com}, x)$ outputs a commitment c_x to x and auxiliary information $open_x$. A commitment is opened by revealing $(x, open_x)$ and checking whether $\mathsf{Open}(params_{Com}, c_x, x, open_x)$ is true. A commitment scheme has a hiding property and a binding property. Informally speaking, the hiding property ensures that a commitment c_x to x does not reveal any information about x, whereas the binding property ensures that c_x cannot be opened to another value x'. Given two commitments c_{x_1} and c_{x_2} with openings $(x_1, open_{x_1})$ and $(x_2, open_{x_2})$ respectively, the additively homomorphic property ensures that, if $c = c_{x_1} \cdot c_{x_2}$, then $\mathsf{Open}(params_{Com}, c, x_1 + x_2, open_{x_1} + open_{x_2})$ is true.

Proofs of Knowledge. A zero-knowledge proof of knowledge is a two-party protocol between a prover and a verifier. The prover proves to the verifier knowledge of some secret values that fulfill some statement without disclosing the secret values to the verifier. For instance, let x be the secret key of a public key $y = g^x$, and let the prover know (x, g, y), while the verifier only knows (g, y). By means of a proof of knowledge, the prover can convince the verifier that he knows x such that $y = g^x$, without revealing any information about x.

5.3.1.2.2 Intuition Behind Our Construction

During each tax period *tag*, the OBU slices the trajectories of the driver in segments formed by a structure containing GPS location data and time. Additionally, this data structure can contain information about any other parameter that influences the price to be paid for driving on the segment. We represent this data structure as a tuple $(loc, time)$. The TSP establishes a function $f : (loc, time) \to \Upsilon$ that maps every possible tuple $(loc, time)$ to a price $p \in \Upsilon$. For each segment, the OBU calculates f on input $(loc, time)$ to get a price p, and computes a payment tuple that consists of a randomized hash h on the data structure $(loc, time)$, a homomorphic commitment c_p to its price, and a proof π that the committed price belongs to Υ. The randomization of the hash is needed in order to prevent dictionary attacks to recover $(loc, time)$.

At the end of the tax period, the OBU and the TSP engage in a two-party protocol. The OBU adds the fees of all the segments to obtain a total fee *fee*.

The OBU adds all the openings $open_p$ to obtain an opening $open_{fee}$. Next, the OBU composes a payment message m that consists of $(tag, fee, open_{fee})$ and all the payment tuples (h, c_p, π). The OBU signs m and sends both the message m and its signature s_m to the TSP. The TSP verifies the signature and, for each payment tuple, verifies the proof π. Then the TSP, by using the homomorphic property of the commitment scheme, adds the commitments c_p of all the payment tuples to obtain a commitment c'_{fee}, and checks that $(fee, open_{fee})$ is a valid opening for c'_{fee}. This check ensures that the final fee is the sum of the committed prices, while the verification of the proof ensures that the committed price belongs to the image of f.

When the TC sends the TSP a proof ϕ that a car was at some position at a given time, the TSP relays ϕ to the OBU. The OBU first verifies that the request is signed by the TC, and then it searches for a payment tuple (h, c_p, π) for which $\mu(\phi, (loc, time))$ outputs accept. Here, $\mu : (\phi, (loc, time)) \to \{\text{accept}, \text{reject}\}$ is a function established by the TSP that outputs accept when the information in ϕ and in $(loc, time)$ are similar in accordance with some metric, such as the one proposed in [173]. Once the payment tuple is found, the OBU sends the number of the tuple to the TSP together with the preimage $(loc, time)$ of h and the opening $(p, open_p)$ of c_p. The TSP checks that $(p, open_p)$ is the valid opening of c_p, that $(loc, time)$ is the preimage of h and that $\mu(\phi, (loc, time))$ outputs accept.

Intuitively, this protocol ensures the four security properties enumerated in the previous section. Drivers cannot shut down their OBUs, nor report false GPS data as they run the risk of not having committed to a segment containing the $(loc, time)$ in the challenge ϕ. We note that after sending (m, s_m) to the TSP, OBUs cannot claim that they were at any position $(loc', time')$ different from the ones used to compute the message m. Similarly, OBUs cannot use incorrect road prices without being detected, as the TSP can check whether the correct price for a segment $(loc, time)$ was used once the commitments are opened. The homomorphic property ensures that the reported final fee is not arbitrary, but the sum of all the committed subfees. Moreover, by making the OBU prove that the committed prices belong to the image of f, we avoid that a malicious OBU could decrease the final fee by sending only one wrong commitment to a negative price in the payment message, which would give it an overwhelming probability of not being detected by the spot checks. Additionally, the fact that the OBU signs the payment message m ensures that no malicious TSP can frame an OBU by modifying the received commitments, and that a malicious OBU cannot plead innocent by invoking the possibility of being framed by a malicious TSP. Similarly, the fact that the TC signs the challenge ϕ prevents a malicious TSP sending fake proofs to the OBU, e.g. with the aim of learning its location. Finally, the privacy of the drivers is preserved as the OBU does not need to disclose more location information than that in the payment tuple that matches the proof ϕ (already known to TSP).

5.3.1.2.3 Efficient Instantiation: High Level Specification

We now outline at high level our efficient instantiation of Optimistic Payment. We employ the integer commitment scheme due to Damgård and Fujisaki [80] and the CL-RSA signature scheme proposed by Camenisch and Lysyanskaya [58]. Both schemes use cryptographic keys based on special RSA modulus n of length l_n. A commitment c_x to a value x is computed as $c_x = {g_0}^x {g_1}^{open_x} \mod n$, where the opening $open_x$ is a random number of length l_n and the bases (g_0, g_1) correspond to the commitment public parameters. Given a public key $pk = (n, R, S, Z)$, a CL-RSA signature has the form (A, e, v), with lengths l_n, l_e, and l_v respectively, such that $Z \equiv A^e R^x S^v (\text{mod } n)$. To prove that a price belongs to Υ, we use a non-interactive proof of possession of a CL-RSA signature on the price. We also employ a collision resistant hash function $H : \{0,1\}^* \to \{0,1\}^{l_c}$.

Initialization. The pricing policy $f : (loc, time) \to \Upsilon$, where each price $p \in \Upsilon$ has associated a valid CL-RSA signature (A, e, v) generated by the TSP, the cryptographic key pair $(pk_{\text{OBU}}, sk_{\text{OBU}})$, the public key of the TSP (n, R, S, Z), the public key of TC, and the public parameters (g_0, g_1) of the commitment scheme are stored on the OBU. Similarly, the TSP possesses its own secret key (sk_{TSP}) and knows all the public keys in the system.

Tax period. Table 5.1 depicts the calculations and interactions between the OBU and the TSP under normal functioning during the tax period. We denote the operations carried out by the OBU as $\mathsf{Pay}()$, and the operations executed by the TSP as $\mathsf{VerifyPayment}()$. While driving, the OBU collects location data and slices it in segments $(loc, time)$ according to the policy. For each of the N collected segments, the OBU generates a payment tuple $(h_k,\ c_{p_k},\ \pi_k)$. This iterative step is broken down in lines 1 to 21 in Table 5.1. The most resource consuming operation is the computation of π_k, which proves the possession of a valid CL-RSA signature on the price p_k (lines 9 to 20). The length of the random values used in this step are:

$$\begin{aligned}
r_x &\leftarrow \{0,1\}^{l_x+l_c+l_z} &, r_{open_x} &\leftarrow \{0,1\}^{l_n+l_c+l_z} \\
r_w &\leftarrow \{0,1\}^{l_n+l_c+l_z} &, r_{open_w} &\leftarrow \{0,1\}^{l_n+l_c+l_z} \\
r_e &\leftarrow \{0,1\}^{l_e+l_c+l_z} &, r_{w\cdot e} &\leftarrow \{0,1\}^{l_n+l_e+l_c+l_z} \\
r_v &\leftarrow \{0,1\}^{l_v+l_c+l_z} &, r_{open_w\cdot e} &\leftarrow \{0,1\}^{l_n+l_e+l_c+l_z}.
\end{aligned}$$

	OBU		**TSP**
	$\mathsf{Pay}()$ algorithm		$\mathsf{VerifyPayment}()$ algorithm
1	// **Main loop**		
2	*For all* $1 \leq k \leq N$ *tuples do:*		
3	$p_k = f(loc_k, time_k)$		
4	// **Hash**		
5	$h_k = H((loc_k, time_k))$		
6	// **Commitment**		
7	$open_{p_k} \leftarrow \{0,1\}^{l_n}$		
8	$c_{p_k} = {g_0}^{p_k} {g_1}^{open_{p_k}} \mod n$		
9	// **Proof**		
10	$open_w, w \leftarrow \{0,1\}^{l_n}$		
11	$\tilde{A} = A{g_0}^w \mod n$		
12	$c_w = {g_0}^w {g_1}^{open_w} \mod n$		$\mathrm{OBU}\mathsf{verify}(pk_{\mathrm{OBU}}, m, s_m)$
13	$r_\alpha \leftarrow \{0,1\}^{l_\alpha}$		// **Main loop**
14	$t_{c_{p_k}} = {g_0}^{r_{p_k}} {g_1}^{r_{open_{p_k}}}$	$\xrightarrow{(m, s_m)}$	*For all* $1 \leq k \leq N$ *tuples do:*
15	$t_Z = \tilde{A}^{r_e} R^{r_{p_k}} S^{r_v} (g_0^{-1})^{r_{w \cdot e}}$		$t'_{c_{p_k}} = c_{p_k}^{ch} {g_0}^{s_{p_k}} {g_1}^{s_{open_x}}$
16	$t_{c_w} = g_0^{r_w} g_1^{r_{open_w}}$		$t'_Z = Z^{ch} \tilde{A}^{s_e} R^{s_{p_k}} S^{s_v} (1/g_0)^{s_{w \cdot e}}$
17	$t = c_w^{r_e} (g_0^{-1})^{r_{w \cdot e}} (g_1^{-1})^{r_{open_w \cdot e}}$		$t'_{c_w} = c_w^{ch} {g_0}^{s_w} {g_1}^{s_{open_w}}$
18	$ch = H(\beta \| t_{c_{p_k}} \| t_Z \| t_{c_w} \| t)$		$t' = C_w^{s_e} (1/g_0)^{s_{w \cdot e}} (1/g_1)^{s_{open_{w \cdot e}}}$
19	$s_\alpha = r_\alpha - ch \cdot \alpha$		$ch' = H(\beta \| t'_{c_{p_k}} \| t'_Z \| t'_{c_w} \| t')$
20	$\pi_k = (\tilde{A}, c_w, ch, s_\alpha)$		$ch'? = ch$
21	*End for*		$s_e \in \{0,1\}^{l_e + l_c + l_z}$
22	// **Fee reporting**		$s_{p_k} \in \{0,1\}^{l_p + l_c + l_z}$
23	$fee = \sum_{k=1}^{N} p_k$		*End for*
24	$open_{fee} = \sum_{k=1}^{N} open_{p_k}$		// **Commitment validation**
25	$m = [tag, fee, open_{fee},$		$c'_{fee} = \prod_{k=1}^{N} c_{p_k}$
	$(h_k, c_{p_k}, \pi_k)_{k=1}^{N}]$		$c_{fee} = {g_0}^{fee} {g_1}^{open_{fee}} \mod n$
26	$s_m = \mathrm{OBU}\mathsf{sign}(sk_{\mathrm{OBU}}, m)$		$c_{fee}? = c'_{fee}$
	$\alpha \in \{p_k, open_{p_k}, e, v, w, open_w, w \cdot e, open_{w \cdot e}\}$		
	$\beta = (n \| g_0 \| g_1 \| \tilde{A} \| R \| S \| g_0^{-1} \| g_1^{-1} \| c_{p_k} \| Z \| c_w \| 1)$		

Table 5.1: Protocol between OBU and TSP during taxing phase

At the end of the tax period the OBU generates and signs the payment message m including the tag tag, the total fee, the opening $open_{fee}$, and all the payment tuples (h_k, c_{p_k}, π_k), lines 22 to 26. Finally it sends (m, s_m) to the TSP.

Upon reception of a payment message, the TSP executes the VerifyPayment() algorithm. First the TSP verifies the signature s_m using the OBU's public key pk_{OBU}. Next, it proceeds to the verification of the proof π_k included in each of the N payment tuples contained in m, lines 13 to 23. In each iteration it performs a series of modular exponentiations, and uses the intermediate results to compute the hash ch'. Then, it checks whether ch' is the same as the value ch contained in π_k. If this verification, together with the two range proofs in lines 21 and 22, is successful, the TSP is convinced that all the prices p_k used by the OBU are indeed a valid image of f. Finally, the TSP validates the commitments c_{p_k} to ensure that the aggregation of all subfees add up to the final fee (lines 24 to 26). For this, it calculates c'_{fee} as the product of all commitments c_{p_k}, and computes the commitment c_{fee} using the values fee and $open_{fee}$ provided by the OBU. If both values are the same, the TSP is convinced that the final fee reported by the OBU adds up to the sum of all subfees reported in the payment tuples.

Proof Challenge. We denote as OBUopen() and Check() the algorithms carried out by the OBU and the TSP, respectively, when the former is challenged with ϕ. When running the OBUopen() algorithm, the OBU searches for the pre-image $(loc_k, time_k)$ of a hash h_k containing the location and time satisfying ϕ, and sends this information to the service provider along with the price p_k and the opening $open_{p_k}$.

Upon reception of this message, the TSP executes the Check() algorithm. First, it verifies whether the segment $(loc_k, time_k)$ actually contains the location in ϕ. Then, it computes the value $h'_k = H(loc_k, time_k)$ and checks whether the OBU had committed to this value in one of the payment tuples reported during the tax period. Lastly, the TSP uses $open_{p_k}$ to open the commitment c_{p_k} and verifies whether $p'_k = f(loc_k, time_k)$ equals the price p_k reported by the OBU during the OBUopen() algorithm. If all verifications succeed, the TSP is convinced that the location data used by the OBU in the fee calculation and the price assigned by the OBU to the segment $(loc_k, time_k)$ are correct.

5.3.1.3 PrETP Evaluation

In this section we evaluate the performance of PrETP. We start by describing the test scenario and both our OBU and TSP prototypes. Next, we analyze the performance of the prototypes for different configuration parameters. Finally, we study the communication overhead in PrETP, and compare it to existing ETP systems.

5.3.1.3.1 Test Scenario

Policy model. The first step in the implementation of PrETP consists in specifying a policy model in the form of the mapping function $f : (loc, time) \rightarrow \Upsilon$. We decide to follow the same criteria as currently existing insurance schemes, i.e., road prices are determined by two parameters: type of road and time of the day. More specifically, we define three categories of roads ('highway', 'primary', and 'others') and three time slots during the day. For each of the possible nine combinations we assign a price per kilometer p and we create a valid signature (A, e, v) using the TSP's secret key. We note that the choice of this policy is arbitrary and that PrETP, as well as OP, can accommodate other price strategies.

Location data. We provide the OBU with a set of location data describing a real trajectory of a vehicle, as this allows us to compare the same test scenario for different configuration parameters. These data are obtained by driving with our prototype for one hour in an urban area, covering a total distance of 24 kilometers. We note that such dataset is sufficient to validate the performance of PrETP, since results for different driving scenarios (e.g., faster or slower) can easily be extrapolated from the results presented in this section.

Parameters of the instantiation. The performance of OP depends directly on the length of the protocol instantiation parameters, and in particular, on the size of the cryptographic keys of the entities (l_n). In our experiments we consider three case studies: medium security (l_n = 1024 bits), high security (l_n = 1536 bits), and very high security (l_n = 2048 bits). The value l_p is determined by the length of the prices p, which in turn determines the value of l_e. Therefore, both lengths are constant for all security cases. The value of l_v varies depending on the value of l_n. Finally, the rest of parameters (l_h, l_r, l_z, and l_c) are set as the output length of the chosen hash function primitive (see Sect. 5.3.1.3.2). These lengths determine the size of the random numbers generated in line 13 in Protocol 5.1. Table 5.2 summarizes the parameter lengths considered for each security level.

OBU Platform. In order to make our prototype as realistic as possible, we implement PrETP using as starting point the embedded design described in [27], which performs the conversion of raw GPS data into a final fee internally. We

Table 5.2: Length of the parameters (in bits)

Parameter	l_n	l_e	l_v	l_p	l_r,l_h,l_z,l_c
Normal Sec.	1 024	128	1 216	32	160
High Sec.	1 536	128	1 728	32	160
Very high Sec.	2 048	128	2 240	32	160

extend and adapt this prototype with the functionalities of OP to make it compatible with PrETP.

At high-level, the elements of our OBU prototype [27] are: a processing unit, a GPS receiver, a GSM modem, and an external memory module. We use as benchmark the Keil MCB2388 evaluation board, which contains an NXP LPC2388 32-bit ARM7TDMI microcontroller. This microcontroller implements a RISC architecture, it runs at 72 MHz, and it offers 512 Kbytes of on-chip program memory and 98 Kbytes of internal SRAM. As external memory, we use an off-the-shelf 1 GByte SD Card connected to the microcontroller. Finally, we use a Telit GM862-GPS as both GPS receiver and GSM modem.

As our platform does not contain any cryptographic coprocessors, we implement all functionalities exclusively in software. Although we could consider the addition of a hardware coprocessor to the prototype in order to carry out the most expensive cryptographic computations, we choose the option that minimizes the production costs of the OBU. Besides, this approach allows us to identify the bottlenecks in the protocol implementation, leaving the door open to hardware-based improvements if needed. Similarly, note that our implementations do not contain countermeasures against side channel or fault attacks. The goal of this demonstrator is to give an intuition on the cost and performance of an OBU prototype. We stress however that any commercially deployed OBU should provide means to protect its cryptographic implementations against physical attacks. Otherwise, other attack scenarios may arise that compromise the security and privacy properties of the overall system.

We have constructed a cryptographic library with the primitives required by our instantiation of the OP protocol, namely: i) a modular exponentiation technique, ii) a one-way hash function, and iii) a random number generator. For the first primitive we use the ACL [28] library, a collection of arithmetic and modular routines specially designed for ARM microcontrollers. As hash function we choose RIPEMD-160 [90], with an output length l_h of 160 bits. As our platform does not provide any physical random number generator, we use the Salsa20 [35] stream cipher in keystream mode as third primitive.

In order to keep the OBU flexible and easily scalable, we arrange data in different memory areas depending on their lifespan. Long-term parameters ($pk_{\mathrm{OBU}}, sk_{\mathrm{OBU}}, pk_{\mathrm{TSP}}$, commitment parameters) are directly embedded into the microcontroller's program memory, while short-term parameters (payment tuples, $(loc, time)$ segments) and updatable parameters (digital road map, policy f) are stored separately on the SD Card. We note that our library provides a byte-oriented interface with the SD Card, resulting in a considerable overhead when reading/writing values.

TSP Platform. We implement our TSP prototype on a commodity computer equipped with an Intel Core2 Duo E8400 processor at 3 GHz, and 4 Gbyte of RAM. We use C as programming language, and the GMP library for large-integer cryptographic operations.

5.3.1.3.2 Performance Evaluation

OBU performance. The most time-consuming operations carried out by the OBU during the taxing phase are the $\mathsf{Mapping}()$ algorithm and the $\mathsf{Pay}()$ algorithm. The $\mathsf{Mapping}()$ algorithm is executed every time a new GPS string is available in the microcontroller. Its function is to search in the digital road map the type of road given the GPS coordinates. When the vehicle drives for a kilometer, the OBU maps the segment to the adequate price p_k as specified in the policy. At this point, the $\mathsf{Pay}()$ algorithm is executed in order to create the payment tuple. For each segment, the OBU generates: i) a hash value h_k of the location data, ii) a commitment c_{p_k} to the price p_k, and iii) a proof π_k proving that the price p_k is genuinely signed by the TSP (and thus belongs to the image of f). To protect users' privacy we also require that no sensitive data is stored in the SD Card in plaintext form. For this purpose we use the AES block cipher in CCM mode with a key length of 128 bits. We denote this operation as E_k. At the end of the taxing phase, the OBU adds all the prices p_k mapped to each segment to obtain the fee, and all the openings $open_k$ to obtain $open_{fee}$. Finally, the OBU constructs and signs the payment message m and sends it to the TSP.

As it does not involve the key, the computing time of the $\mathsf{Mapping}()$ algorithm is independent of the security scenario. Further, this time only depends on the duration of the trip and is independent of the speed of the vehicle: the $\mathsf{Mapping}()$ algorithm is always executed 3 600 times per hour, taking a total of 839.11 seconds in our prototype. However, for each of the segments this time can vary depending on the number of points that have to be processed, i.e., depending on the speed of the vehicle. In our experiments it requires 76.10

seconds for the longest segment, i.e., the one where the vehicle spent more time to drive one kilometer and thus $(loc_k, time_k)$ contains the larger number of points.

Similarly, the execution time for h_k and E_k depends exclusively on the length of the segments $(loc_k, time_k)$, as it is proportional to the number of GPS points in the segments. The amount of points per segment varies not only with the average speed of the car but also depending on the length of the segments defined in the pricing policy. In our experiments, computing h_k and E_k take 0.08 seconds and 0.43 seconds, respectively, for the shortest and the longest segments. For the Mapping() algorithm and both h_k and E_k operations, more than 90% of the time is spent in the communication with the SD card.

On the other hand, the execution time for c_{p_k} and π_k is constant for all segments, as it does not depend on the length of a particular slice (see lines 6 to 20 in the protocol depicted in Table 5.1). In order to calculate c_{p_k}, the OBU needs to generate a random opening $open_{p_k}$ and perform two modular exponentiations and a modular multiplication. The computation of π_k involves the generation of ten random numbers and a hash value, and the execution of fourteen modular exponentiations, nine modular multiplications, eight additions, and eight multiplications. The bottleneck of both operations is determined by the modular operations. Although we could take advantage of fixed-base modular exponentiation techniques, we choose to use multi-exponentiation algorithms [88], which have less storage requirements. Multi-exponentiation based algorithms, which compute values of the form $a^b c^d (\bmod n)$ in one step, allow us to considerably speed up the process. The average execution times for computing c_{p_k} are 0.76 seconds, 2.25 seconds, and 5.69 seconds for medium, high, and very high security respectively. For π_k, these times are 6.20 seconds, 19.45 seconds, and 41.64 seconds, respectively.

Table 5.3 summarizes the timings for all OBU operations and routines for a journey of one hour. We note that, even when 2048-bit RSA keys are used, the OBU can perform all operations needed to create the payment tuples in real time. While the trip lasted one hour, the Mapping() and Pay() algorithms only required 1 982.41 seconds. The computation time is dominated by the Pay() algorithm, which depends on the number of GPS strings in each segment $(loc, time)$. This number varies with the speed of the vehicle and the pricing policy. If a vehicle is driving at a constant speed, policies that establish prices for small distances result in segments containing less GPS points than policies that consider long distances. Similarly, given a policy fixing the size of the segments, driving faster produces segments with less points than driving slower. In both cases, π_k has to be computed fewer times and the Pay() algorithm runs faster. Thus, the policy can be used as tuning parameter to guarantee the real-time operation of the OBU.

Table 5.3: Execution times (in seconds) for an hour journey of 24 km, for all possible security scenarios.

		Medium Security		**High Security**		**Very high Security**	
Algorithm		**Segment**	**Full trip**	**Segment**	**Full trip**	**Segment**	**Full trip**
Mapping()		76.10 s	839.11 s	76.10 s	839.11 s	76.10 s	839.11 s
Pay()		7.88 s	183.91 s	22.13 s	528.47 s	47.79 s	1 143.30 s
	h_k	0.08 s	1.08 s	0.08 s	1.08 s	0.08 s	1.08 s
	E_k	0.43 s	6.35 s	0.43 s	6.35 s	0.43 s	6.35 s
	c_{p_k}	0.76 s	18.19 s	2.25 s	54.08 s	5.69 s	136.82 s
	π_k	6.20 s	158.09 s	19.45 s	466.96 s	41.64 s	999.05 s

Using the values in Table 5.3, for each of the levels of security we can calculate the time our OBU is idle – in our case $(3\,600 - 839.11)$ seconds, with 839.11 seconds being the time required by the Mapping() algorithm. Then, considering our current policy, we can estimate the number of times the Pay() algorithm could be executed, which in turn represents the number of kilometers that could have been driven by a car in one hour, i.e., the average speed of the car. For normal security, our OBU could operate in real time even if a vehicle was driving at 350 km/h. This speed decreases to 124 km/h when 1536-bit keys are used, and to 57 km/h if the keys have length 2048 bits. Only when using high security parameters our OBU would have problems to operate in the field. However, as mentioned before, including a cryptographic coprocessor in the platform would suffice to solve this problem whenever high security is required. Moreover, in our tests we consider a worst-case scenario in which all GPS strings are processed upon reception. In fact, processing fewer strings would suffice to determine the location of the vehicle. As the execution time required by the Mapping() algorithm would decrease linearly, OBUs would be able to support higher vehicle speeds.

In the OBUopen() algorithm, only executed upon request from TC, the OBU searches its memory for a segment $(loc, time)$ in accordance to the proof sent by the TSP. Here, the time accuracy provided by the GPS system is used to ensure synchronization between the data in ϕ and the segment $(loc, time)$. The main bottleneck of this operation is the decryption of the location data corresponding to the correct segment. On average, our prototype can decrypt such a segment in 0.27 seconds.

TSP performance. The most consuming task the TSP must perform corresponds to the VerifyPayment() algorithm, which has to be executed each time the TSP receives a payment message. This algorithm involves three operations: the verification of the proof π_k for each segment, the multiplication of all commitments c_{p_k} to obtain c_{fee}, and the opening of c_{fee} in order to check whether it corresponds to the reported final fee. The most costly operation is the verification of π_k, in particular the calculation of the parameters (t'_{c_m}, t'_Z, t'_{c_w}, t') which requires a total of eleven modular exponentiations (lines 14 to 22 in protocol depicted in Table 5.1).

Table 5.4 shows the performance of the VerifyPayment() algorithm for each of the considered security levels when segments have length one kilometer. We also provide an estimation of the time required to process all the proofs sent by OBU during a month, assuming that a vehicle drives an average of 18 000 km per year (1 500 km per month).

VerifyPayment()	**Segment**	**One Month**
Medium Sec.	0.0105 s	15.750 s
High Sec.	0.0295 s	44.250 s
Very high Sec.	0.0587 s	88.050 s

Table 5.4: Timings (in seconds) for the execution of VerifyPayment() in the TSP.

These results allow us to extrapolate the number of OBUs that can be supported by a single TSP in each security scenario for different segment lengths. Intuitively, the capacity of TSP increases when segments are larger, as the payment messages contain fewer proofs π_k. The number of OBUs supported by a single TSP is presented in Table 5.5. For a segment length of 1 km, the TSP is able to support 164 000, 58 000, and 29 000 vehicles depending on the chosen security level. Even when l_n is 2048 bits, only 36 servers are needed to accommodate one million OBUs. This number can be reduced by parallelizing tasks at the server side, or by using fast cryptographic hardware for the modular exponentiations.

Segment size	**Medium Sec.**	**High Sec.**	**Very high Sec.**
0.5 km	82 000	29 000	14 000
0.75 km	123 000	43 000	22 000
1 km	164 000	58 000	29 000
2 km	329 000	117 000	58 000
3 km	493 000	175 000	88 000

Table 5.5: Number of OBUs supported by a single TSP.

5.3.1.3.3 Communication Overhead

We now compare the communication overhead of PrETP with respect to straightforward ETP implementations and VPriv [173]. Both in straightforward ETP implementations and in VPriv the OBU sends all GPS strings to the TSP. Let us consider that vehicles drive 1 500 km per month at an average speed of 80 km/h. Then, transmitting the full GPS information to the the TSP requires 2.05 Mbyte (considering a shortened GPS string of 32 bytes containing only latitude, longitude, date and time). VPriv requires more bandwidth than straightforward ETP systems, as extra communications are necessary to carry out the interactive verification protocol. Using PrETP, the communication overhead comes from the payment tuples that must be sent along with the fee. For each segment, the OBU sends the payment tuple (h, c_p, π) to the TSP. When sent uncompressed, this implies an overhead of approximately 1.5 Kbyte per segment, i.e., less than 2 Mbyte per month, for medium security (l_n=1024 bits). Additionally, less than 50 Kbyte have to be sent occasionally to respond a verification challenge after a vehicle has been seen at a spot check. We believe this overhead is not excessive for the additional security and privacy properties offered by PrETP.

The communication overhead in PrETP is dominated by the payment message m sent by the OBU to the TSP, the length of which depends on the number of segments covered by the driver. Therefore, the segment length can be seen as a parameter of the system that tunes the tradeoff between privacy and communication overhead. The smaller the segments, the larger the communication overhead, because more tuples (h_k, c_{p_k}, π_k) need to be sent. Allowing larger segments reduces the communication cost but also reduces privacy because the OBU must disclose a bigger segment when responding a verification challenge.

Further, the communication overhead can be almost eliminated by having the OBU sending only the hash of the payment message at the end of each tax period and leave the correct operation verification subject to random checks. Following the spirit of the random “spot checks” used for checking the input and prices, the OBUs could occasionally be challenged to prove its correct functioning by sending the payment message corresponding to the preimage of the hash sent at the end of a random tax period.

5.4 Conclusions

The disclosure of personal data in monitoring systems conflicts with the users' right to privacy. Moreover, it can pose inconveniences and extra investments to service providers, as the law demands that personal data is stored and processed under strong security guarantees.

Security and privacy concerns are among the main reasons that discourage the use of electronic communication services [137]. Users confronted to a prominent display of private information prefer service providers that offer better privacy guarantees [216]. Consequently, it is of interest for service providers to design systems where the amount of personal information that users need to disclose is minimized.

Electronic Toll Pricing schemes are soon becoming a reality. Because of its application for public road taxing, this system is expected to have a significant social and economical impact in the near future. It is therefore necessary from the start to design solutions that take into account both the privacy of users and the interests of the service provider. Previous work relies on too expensive solutions or on unrealistic requirements, to fulfill both properties. Our contribution fills this gap by presenting a system in which on-board units can prove that they operate correctly while leaking the minimum amount of information.

Chapter 6

Conclusions and Open Problems

The integration of security and privacy into embedded systems is an active research area that needs to keep track of technological developments. The paradigm of the Internet of Things envisioning a new wave of computing devices with interconnection capabilities is a clear exponent of the future challenges that need to be addressed. While passive identification systems such as RFID tags represent the beginning of this vision, further technologies are gradually being integrated into the IoT paradigm.

On the one hand, monitoring devices are representative of a change of landscape in which devices become active. Equipped with sensing, processing, and communication capabilities, these technologies range from miniature elements (i.e. *smart dust*) to larger in-home or in-vehicle devices. Integration of security and privacy must be carefully balanced not only with common cost *vs.* speed trade-offs but, particularly for small sensors, with increasingly tight constraints on *power* and *energy* consumption.

On the other hand, the pioneering role of *smartphones* in ubiquitous connectivity makes them a representative illustration of the trend towards the IoT. The widespread adoption of these devices and the appearance of standard short-range communication interfaces such as NFC, is stimulating the migration of security-related applications (e.g. payment, identification) towards such platforms. Traditionally, these applications involve the use of secure tokens (e.g. credit cards, electronic IDs) in which a small number of cryptographic operations is carried out in close, protected environments. The transfer of such functionalities to larger platforms, involving potentially untrusted/unsecure operating systems and multiple third-party elements, requires the study and analysis of new security models with a more global, system-wide view of the problem.

Besides these envisioned changes in the landscape of embedded systems, in the following we enumerate some foreseen security and privacy related issues that have either not been covered in this dissertation and represent open challenges.

Fault Injection.

In the last years, the cost for acquiring (semi-) and invasive fault injection setups based on laser beams has drastically decreased. Solutions equipped with powerful multimode diode lasers with adaptive pulse lengths and power modulations are already commercially available. Mounted on an XY stage, these devices allow precise fault injection over accurate spots of a circuit. Practical attacks injecting multiple faults within a single cryptographic execution or even using two simultaneously operated lasers have been recently shown. These possibilities open the door to more powerful adversarial models capable of defeating traditional countermeasures based on e.g. double method re-computation. Therefore, the design of suitable and efficient countermeasures against such type of attacks has to be addressed.

The migration of security-related functionalities into larger platforms carries within an inherent threat by providing adversaries with multiple attack points. In addition to protecting the execution of cryptographic operations, one needs to consider the potential effect of faults injected in the surrounding elements. This not only includes neighboring hardware blocks or cores, but also operating systems and other application-specific software components. Protecting these elements requires a system-wide view of the problem, posing extra challenges to the already complex issue of fault detection. A related research area to evaluate such system-wide countermeasures consists in the design of automated *simulation* tools to emulate the occurrence of randomly induced faults and determine whether or not they can be successfully detected.

Higher-Order Masking Techniques.

Because of its proven soundness, the design of masking schemes implementable at any order d is an active research topic of interest for both industry and academia. It is long known that the cost of applying a $d+1$-variate attack against a masking scheme grows exponentially with the noise as the masking order d increases. More recent works have shown that the leakage information of non-linear masking constructions, e.g. polynomial masking or IP masking, are several orders of magnitude smaller than linear masking constructions, e.g. boolean masking, for similar levels of noise. These characteristics illustrate the potential of non-linear masking constructions to effectively defeat practical side-channel attacks. Given a reasonable amount of shares and noise, the resources required by an adversary (in terms of trace collection and data processing) may yield leakage exploitation unfeasible in practice.

There are however two important issues that require further work. First, while devising non-linear masking constructions is easy, the design of operations in the masked domain secure at any order d is not. This is exemplified by the recent attacks against IP masking [179] and polynomial masking [75] schemes. Whether these schemes may be fixed is a topic for future research. Second, existing operations in the masked domain incur significantly more computational complexity than e.g. boolean masking schemes [188], resulting in close to prohibitive overheads. In other words, the trade-off between provided security and efficiency must be further balanced. This can be done by devising new, more efficient algorithms or, alternatively, by adapting the underlying functionalities of target devices to offer better support for these schemes. Exploring the hardware/software co-design space can help extending generic architectures (e.g. via instruction extensions) to add support for commonly used operations in non-linear masking constructions. In the case of IP masking for instance, an example of such operation is given by field multiplication.

Privacy-Preserving Systems.

As the amount of systems and applications in today's ubiquitous society continues to increase, so does people's awareness of the problem posed by massive data collection. The revocation of the Dutch smart metering bill [77] or the controversy caused by the European FP7 project INDECT (INtelligent information system supporting observation, searching and DEteCTion for security of citizens in urban environment) [209] are examples of a paradigm shift with respect to the views on privacy.

Efforts to develop guidelines on the integration of privacy protection as a *design requirement* in a system's lifecycle are being carried out. However, so far no methodology has emerged and most proposals are based on particular use cases such as Electronic Toll Pricing. As services have their own requirements and goals, each case seems to require specific solutions adapted to the context in which they will be deployed. Whether a generalization can emerge from all these particular solutions is still an open question.

Bibliography

[1] AT88SC0204 ChipResetter. `http://chipreset.atw.hu/6/index61.html`.

[2] Coinamatic. `http://www.coinamatic.com`.

[3] ISO/IEC 7816-3: Identification cards - integrated circuit(s) cards with contacts - part 3: Electronic signals and transmission protocols (1997).

[4] Labgear HDSR300 High Definition Satellite Receiver. User Guide. `http://www.free-instruction-manuals.com/pdf/p4789564.pdf`.

[5] Data Encryption Standard (DES). Federal Information Processing Standards (FIPS) Publication 46, 1977.

[6] Specification for the Advanced Encryption Standard (AES). Federal Information Processing Standards (FIPS) Publication 197, 2001.

[7] NSA Prism program taps in to user data of Apple, Google and others. The Guardian, 2013.

[8] D. G. Abraham, G. M. Dolan, G. P. Double, and J. V. Stevens. Transaction Security System. *IBM Systems Journal*, 30(2):206–229, 1991.

[9] M. Agoyan, J.-M. Dutertre, A.-P. Mirbaha, D. Naccache, A.-L. Ribotta, and A. Tria. How to flip a bit? In *International On-Line Testing Symposium - IOLTS 2010*, pages 235–239. IEEE Computer Society, 2010.

[10] M. Agoyan, J.-M. Dutertre, D. Naccache, B. Robisson, and A. Tria. When Clocks Fail: On Critical Paths and Clock Faults. In D. Gollmann, J.-L. Lanet, and J. Iguchi-Cartigny, editors, *Smart Card Research and Advanced Application - CARDIS 2010*, volume 6035 of *Lecture Notes in Computer Science*, pages 182–193. Springer, 2010.

[11] M.-L. Akkar. *Attaques et méthodes de protections de systèmes cryptographiques embarqués*. PhD thesis, Université de Versailles Saint-Quentin, 2004.

[12] M.-L. Akkar, R. Bevan, and L. Goubin. Two Power Analysis Attacks against One-Mask Methods. In B. K. Roy and W. Meier, editors, *Fast Software Encryption - FSE 2004*, volume 3017 of *Lecture Notes in Computer Science*, pages 332–347. Springer, 2004.

[13] M.-L. Akkar and C. Giraud. An Implementation of DES and AES, Secure against Some Attacks. In Çetin Kaya Koç, D. Naccache, and C. Paar, editors, *Cryptographic Hardware and Embedded Systems - CHES 2001*, volume 2162 of *Lecture Notes in Computer Science*, pages 309–318. Springer, 2001.

[14] D. Anderson. Understanding CryptoMemory - The World's Only Secure Serial EEPROM. `http://www.atmel.com/atmel/acrobat/doc5064.pdf`.

[15] R. Anderson and M. Kuhn. Tamper Resistance: a Cautionary Note. In *2nd USENIX Workshop on Electronic Commerce*, pages 1–11. USENIX Association, 1996.

[16] R. Anderson and M. Kuhn. Low Cost Attacks on Tamper Resistant Devices. In B. Christianson, B. Crispo, M. Lomas, and M. Roe, editors, *5th International Workshop on Security Protocols*, volume 1361 of *Lecture Notes in Computer Science*, pages 125–136. Springer, 1998.

[17] Atmel. CryptoMemory features. `http://www.atmel.com/microsite_cryptomemory/iwe/index.html?source=tout_other2`.

[18] Atmel. CryptoMemory Specification. `http://www.atmel.com/atmel/acrobat/doc5211.pdf`.

[19] Atmel. News Release. `http://www.cryptomemorykey.com/pdfs/AtmelPR.pdf`.

[20] Atmel Corporation. Plug-and-Play Crypto Chip for Host-Side Security. `http://www.atmel.com/dyn/corporate/view_detail.asp?ref=&FileName=Cryptocompanion_2_26.html&SEC_NAME=Product`.

[21] J. Balasch, B. Ege, T. Eisenbarth, B. Gérard, Z. Gong, T. Güneysu, S. Heyse, S. Kerckhof, F. Koeune, T. Plos, T. Pöppelmann, F. Regazzoni, F.-X. Standaert, G. V. Assche, R. V. Keer, L. van Oldeneel tot Oldenzeel, and I. von Maurich. Compact Implementation and Performance Evaluation of Hash Functions in ATtiny Devices. In S. Mangard, editor, *Smart Card*

Research and Advanced Applications - CARDIS 2012, volume 7771 of *Lecture Notes in Computer Science*, pages 158–172. Springer, 2012.

[22] J. Balasch, S. Faust, B. Gierlichs, and I. Verbauwhede. Theory and Practice of a Leakage Resilient Masking Scheme. In X. Wang and K. Sako, editors, *Advances in Cryptology - ASIACRYPT 2012*, volume 7658 of *Lecture Notes in Computer Science*, pages 758–775. Springer, 2012.

[23] J. Balasch, B. Gierlichs, and I. Verbauwhede. An In-depth and Black-box Characterization of the Effects of Clock Glitches on 8-bit MCUs. In L. Breveglieri, S. Guilley, I. Koren, D. Naccache, and J. Takahashi, editors, *Workshop on Fault Diagnosis and Tolerance in Cryptography - FDTC 2011*, pages 105–114. IEEE Computer Society, 2011.

[24] J. Balasch, B. Gierlichs, R. Verdult, L. Batina, and I. Verbauwhede. Power Analysis of Atmel CryptoMemory - Recovering Keys from Secure EEPROMs. In O. Dunkelman, editor, *Topics in Cryptology - CT-RSA 2012*, volume 7178 of *Lecture Notes in Computer Science*, pages 19–34. Springer, 2012.

[25] J. Balasch, B. Preneel, A. Rial, M. Scheir, and I. Verbauwhede. Anonymous E-Cash for Resource Constrained Devices. Cosic internal report, 2012.

[26] J. Balasch, A. Rial, C. Troncoso, C. Geuens, B. Preneel, and I. Verbauwhede. PrETP: Privacy-Preserving Electronic Toll Pricing. In *USENIX Security Symposium 2010*, pages 63–78. USENIX Association, 2010.

[27] J. Balasch, I. Verbauwhede, and B. Preneel. An Embedded Platform for Privacy-Friendly Road Charging Applications. In *Design, Automation and Test in Europe - DATE 2010*, pages 867–872. IEEE Computer Society, 2010.

[28] J. Ban. Cryptographic library for ARM7TDMI processors. Master's thesis, T.U. Kosice, 2007.

[29] F. Bao, R. H. Deng, Y. Han, A. B. Jeng, A. D. Narasimhalu, and T.-H. Ngair. Breaking Public Key Cryptosystems on Tamper Resistant Devices in the Presence of Transient Faults. In B. Christianson, B. Crispo, T. M. A. Lomas, and M. Roe, editors, *Security Protocols Workshop*, volume 1361 of *Lecture Notes in Computer Science*, pages 115–124. Springer, 1997.

[30] H. Bar-El, H. Choukri, D. Naccache, M. Tunstall, and C. Whelan. The Sorcerer's Apprentice Guide to Fault Attacks. *Proceedings of the IEEE*, 94(2):370–382, 2006.

[31] A. Barenghi, G. Bertoni, L. Breveglieri, M. Pellicioli, and G. Pelosi. Low Voltage Fault Attacks to AES. In J. Plusquellic and K. Mai, editors, *Hardware-Oriented Security and Trust - HOST 2010*, pages 7–12. IEEE Computer Society, 2010.

[32] A. Battistello and C. Giraud. Fault Analysis of Infective AES Computations. In W. Fischer and J.-M. Schmidt, editors, *Fault Diagnosis and Tolerance in Cryptography - FDTC 2013*, pages 101–107. IEEE Computer Society, 2013.

[33] J. P. Benhammou and M. Jarboe. Security at an affordable price. In *Atmel Applications Journal*, pages 29–30. Atmel, 2004.

[34] D. J. Bernstein. Cache-timing attacks on AES, 2005.

[35] D. J. Bernstein. The Salsa20 Family of Stream Ciphers. In M. J. B. Robshaw and O. Billet, editors, *New Stream Cipher Designs - The eSTREAM Finalists*, volume 4986 of *Lecture Notes in Computer Science*, pages 84–97. Springer, 2008.

[36] G. Bertoni, L. Breveglieri, I. Koren, P. Maistri, and V. Piuri. Error Analysis and Detection Procedures for a Hardware Implementation of the Advanced Encryption Standard. *IEEE Transactions on Computers*, 52(4):492–505, 2003.

[37] A. Berzati, C. Canovas, J.-G. Dumas, and L. Goubin. Fault Attacks on RSA Public Keys: Left-To-Right Implementations Are Also Vulnerable. In M. Fischlin, editor, *Topics in Cryptology - CT-RSA 2009*, volume 5473 of *Lecture Notes in Computer Science*, pages 414–428. Springer, 2009.

[38] A. Berzati, C. Canovas, and L. Goubin. In(security) Against Fault Injection Attacks for CRT-RSA Implementations. In L. Breveglieri, S. Gueron, I. Koren, D. Naccache, and J.-P. Seifert, editors, *Fault Diagnosis and Tolerance in Cryptography - FDTC 2008*, pages 101–107. IEEE Computer Society, 2008.

[39] A. Berzati, C. Canovas, and L. Goubin. Perturbating RSA Public Keys: An Improved Attack. In E. Oswald and P. Rohatgi, editors, *Cryptographic Hardware and Embedded Systems - CHES 2008*, volume 5154 of *Lecture Notes in Computer Science*, pages 380–395. Springer, 2008.

[40] I. Biehl, B. Meyer, and V. Müller. Differential Fault Attacks on Elliptic Curve Cryptosystems. In M. Bellare, editor, *Advances in Cryptology - CRYPTO 2000*, volume 1880 of *Lecture Notes in Computer Science*, pages 131–146. Springer, 2000.

[41] E. Biham and A. Shamir. Differential Cryptanalysis of DES-like Cryptosystems. In A. Menezes and S. A. Vanstone, editors, *Advances in Cryptology - CRYPTO '90*, volume 537 of *Lecture Notes in Computer Science*, pages 2–21. Springer, 1990.

[42] E. Biham and A. Shamir. Differential Fault Analysis of Secret Key Cryptosystems. In B. S. K. Jr., editor, *Advances in Cryptology - CRYPTO '97*, volume 1294 of *Lecture Notes in Computer Science*, pages 513–525. Springer, 1997.

[43] A. Biryukov, I. Kizhvatov, and B. Zhang. Cryptanalysis of the Atmel Cipher in SecureMemory, CryptoMemory and CryptoRF. In J. Lopez and G. Tsudik, editors, *Proceedings of ACNS 2011*, volume 6715 of *LNCS*, pages 91–109. Springer, 2011.

[44] J. Blömer, J. Guajardo, and V. Krummel. Provably Secure Masking of AES. In H. Handschuh and M. A. Hasan, editors, *Selected Areas in Cryptography - SAC 2004*, volume 3357 of *Lecture Notes in Computer Science*, pages 69–83. Springer, 2004.

[45] J. Blömer, M. Otto, and J.-P. Seifert. A new CRT-RSA algorithm secure against bellcore attacks. In S. Jajodia, V. Atluri, and T. Jaeger, editors, *Computer and Communications Security - CCS 2003*, pages 311–320. ACM, 2003.

[46] J. Blömer, M. Otto, and J.-P. Seifert. Sign Change Fault Attacks on Elliptic Curve Cryptosystems. In L. Breveglieri, I. Koren, D. Naccache, and J.-P. Seifert, editors, *Fault Diagnosis and Tolerance in Cryptography - FDTC 2006*, volume 4236 of *Lecture Notes in Computer Science*, pages 36–52. Springer, 2006.

[47] J. Blömer and J.-P. Seifert. Fault Based Cryptanalysis of the Advanced Encryption Standard (AES). In R. N. Wright, editor, *Financial Cryptography*, volume 2742 of *Lecture Notes in Computer Science*, pages 162–181. Springer, 2003.

[48] A. Bogdanov. Linear Slide Attacks on the KeeLoq Block Cipher. In D. Pei, M. Yung, D. Lin, and C. Wu, editors, *Proceeedings of Inscrypt 2007*, volume 4990 of *LNCS*, pages 66–80. Springer, 2007.

[49] A. Bogdanov. Multiple-Differential Side-Channel Collision Attacks on AES. In E. Oswald and P. Rohatgi, editors, *Cryptographic Hardware and Embedded Systems - CHES 2008*, volume 5154 of *Lecture Notes in Computer Science*, pages 30–44. Springer, 2008.

[50] D. Boneh, R. A. DeMillo, and R. J. Lipton. On the Importance of Checking Cryptographic Protocols for Faults (Extended Abstract). In W. Fumy, editor, *Advances in Cryptology - EUROCRYPT '97*, volume 1233 of *Lecture Notes in Computer Science*, pages 37–51. Springer, 1997.

[51] E. F. Brickell, J. Camenisch, and L. Chen. Direct anonymous attestation. In V. Atluri, B. Pfitzmann, and P. D. McDaniel, editors, *Computer and Communications Security - CCS 2004*, pages 132–145. ACM, 2004.

[52] E. Brier, B. Chevallier-Mames, M. Ciet, and C. Clavier. Why One Should Also Secure RSA Public Key Elements. In L. Goubin and M. Matsui, editors, *Cryptographic Hardware and Embedded Systems - CHES 2006*, volume 4249 of *Lecture Notes in Computer Science*, pages 324–338. Springer, 2006.

[53] E. Brier, C. Clavier, and F. Olivier. Correlation Power Analysis with a Leakage Model. In M. Joye and J.-J. Quisquater, editors, *Cryptographic Hardware and Embedded Systems - CHES 2004*, volume 3156 of *Lecture Notes in Computer Science*, pages 16–29. Springer, 2004.

[54] E. Brier and M. Joye. Weierstraß Elliptic Curves and Side-Channel Attacks. In D. Naccache and P. Paillier, editors, *Public Key Cryptography - PKC 2002*, volume 2274 of *Lecture Notes in Computer Science*, pages 335–345. Springer, 2002.

[55] J. Brouchier, T. Kean, C. Marsh, and D. Naccache. Temperature Attacks. *IEEE Security & Privacy*, 7(2):79–82, 2009.

[56] D. Brumley and D. Boneh. Remote timing attacks are practical. *Computer Networks*, 48(5):701–716, 2005.

[57] J. Camenisch, S. Hohenberger, and A. Lysyanskaya. Compact E-Cash. In R. Cramer, editor, *Advances in Cryptology - EUROCRYPT 2005*, volume 3494 of *Lecture Notes in Computer Science*, pages 302–321. Springer, 2005.

[58] J. Camenisch and A. Lysyanskaya. A Signature Scheme with Efficient Protocols. In S. Cimato, C. Galdi, and G. Persiano, editors, *Security in Communication Networks - SCN 2002*, volume 2576 of *Lecture Notes in Computer Science*, pages 268–289. Springer, 2002.

[59] D. Canright and L. Batina. A Very Compact "Perfectly Masked" S-Box for AES (corrected). Cryptology ePrint Archive, Report 2009/011, 2009. `http://eprint.iacr.org/`.

[60] C. Carlet, L. Goubin, E. Prouff, M. Quisquater, and M. Rivain. Higher-Order Masking Schemes for S-Boxes. In A. Canteaut, editor, *Fast Software Encryption - FSE 2012*, volume 7549 of *Lecture Notes in Computer Science*, pages 366–384. Springer, 2012.

[61] A. Cavoukian. Privacy by Design: The 7 Foundational Principles, 2010.

[62] S. Chari, C. S. Jutla, J. R. Rao, and P. Rohatgi. Towards Sound Approaches to Counteract Power-Analysis Attacks. In M. J. Wiener, editor, *Advances in Cryptology - CRYPTO '99*, volume 1666 of *Lecture Notes in Computer Science*, pages 398–412. Springer, 1999.

[63] S. Chari, J. R. Rao, and P. Rohatgi. Template Attacks. In B. S. K. Jr., Çetin Kaya Koç, and C. Paar, editors, *Cryptographic Hardware and Embedded Systems - CHES 2002*, volume 2523 of *Lecture Notes in Computer Science*, pages 13–28. Springer, 2003.

[64] D. Chaum. Blind Signatures for Untraceable Payments. In D. Chaum, R. L. Rivest, and A. T. Sherman, editors, *Advances in Cryptology - CRYPTO '82*, pages 199–203. Plenum Press, New York, 1982.

[65] Z. Chen and Y. Zhou. Dual-Rail Random Switching Logic: A Countermeasure to Reduce Side Channel Leakage. In L. Goubin and M. Matsui, editors, *Cryptographic Hardware and Embedded Systems - CHES 2006*, volume 4249 of *Lecture Notes in Computer Science*, pages 242–254. Springer, 2006.

[66] H. Choukri and M. Tunstall. Round Reduction Using Faults. pages 13–24, 2005.

[67] M. Ciet and M. Joye. Elliptic Curve Cryptosystems in the Presence of Permanent and Transient Faults. *Designs, Codes and Cryptography*, 36(1):33–43, 2005.

[68] M. Ciet and M. Joye. Practical Fault Countermeasures for Chinese Remaindering Based RSA. In L. Breveglieri and I. Koren, editors, *Fault Diagnosis and Tolerance in Cryptography - FDTC 2005*, pages 124–131. Springer, 2005.

[69] C. Clavier. Secret External Encodings Do Not Prevent Transient Fault Analysis. In P. Paillier and I. Verbauwhede, editors, *Cryptographic Hardware and Embedded Systems - CHES 2007*, volume 4727 of *Lecture Notes in Computer Science*, pages 181–194. Springer, 2007.

[70] C. Clavier, J.-S. Coron, and N. Dabbous. Differential Power Analysis in the Presence of Hardware Countermeasures. In Çetin Kaya Koç and

C. Paar, editors, *Cryptographic Hardware and Embedded Systems - CHES 2000*, volume 1965 of *Lecture Notes in Computer Science*, pages 252–263. Springer, 2000.

[71] C. Clavier and M. Joye. Universal Exponentiation Algorithm. In Çetin Kaya Koç, D. Naccache, and C. Paar, editors, *Cryptographic Hardware and Embedded Systems - CHES 2001*, volume 2162 of *Lecture Notes in Computer Science*, pages 300–308. Springer, 2001.

[72] K. J. Compton, B. Timm, and J. VanLaven. A Simple Power Analysis Attack on the Serpent Key Schedule. *IACR Cryptology ePrint Archive*, 2009:473, 2009.

[73] J.-S. Coron. Resistance against Differential Power Analysis for Elliptic Curve Cryptosystems. In Çetin Kaya Koç and C. Paar, editors, *Cryptographic Hardware and Embedded Systems - CHES 1999*, volume 1717 of *Lecture Notes in Computer Science*, pages 292–302. Springer, 1999.

[74] J.-S. Coron, E. Prouff, and M. Rivain. Side Channel Cryptanalysis of a Higher Order Masking Scheme. In P. Paillier and I. Verbauwhede, editors, *CHES*, volume 4727 of *Lecture Notes in Computer Science*, pages 28–44. Springer, Cryptographic Hardware and Embedded Systems - CHES 2007.

[75] J.-S. Coron, E. Prouff, and T. Roche. On the Use of Shamir's Secret Sharing against Side-Channel Analysis. In S. Mangard, editor, *Smart Card Research and Advanced Applications - CARDIS 2012*, volume 7771 of *Lecture Notes in Computer Science*, pages 77–90. Springer, 2012.

[76] N. Courtois and L. Goubin. An Algebraic Masking Method to Protect AES Against Power Attacks. In D. Won and S. Kim, editors, *Information Security and Cryptology - ICISC 2005*, volume 3935 of *Lecture Notes in Computer Science*, pages 199–209. Springer, 2005.

[77] C. Cuijpers and B.-J. Koops. Smart Metering and Privacy in Europe: Lessons from the Dutch Case. In S. Gutwirth, R. Leenes, P. de Hert, and Y. Poullet, editors, *European Data Protection: Coming of Age*, pages 269–293. Springer, 2013.

[78] J. Daemen and V. Rijmen. AES Proposal: Rijndael, 1998.

[79] J. Daemen and V. Rijmen. *The Design of Rijndael: AES - The Advanced Encryption Standard.* Information Security and Cryptography. Springer, 2002.

[80] I. Damgård and E. Fujisaki. A Statistically-Hiding Integer Commitment Scheme Based on Groups with Hidden Order. In Y. Zheng, editor, *Advances in Cryptology - ASIACRYPT 2002*, volume 2501 of *Lecture Notes in Computer Science*, pages 125–142. Springer, 2002.

[81] G. Danezis and S. F. Gürses. A critical review of 10 years of Privacy Technology. 2010.

[82] C. De Cannière. Trivium: A Stream Cipher Construction Inspired by Block Cipher Design Principles. In S. K. Katsikas, J. Lopez, M. Backes, S. Gritzalis, and B. Preneel, editors, *Information Security Conference - ISC 2006*, volume 4176 of *Lecture Notes in Computer Science*, pages 171–186. Springer, 2006.

[83] W. de Jonge and B. Jacobs. Privacy-Friendly Electronic Traffic Pricing via Commits. In P. Degano, J. D. Guttman, and F. Martinelli, editors, *Formal Aspects in Security and Trust*, volume 5491 of *Lecture Notes in Computer Science*, pages 143–161. Springer, 2008.

[84] A. Dehbaoui, J.-M. Dutertre, B. Robisson, and A. Tria. Electromagnetic Transient Faults Injection on a Hardware and a Software Implementations of AES. In G. Bertoni and B. Gierlichs, editors, *Fault Diagnosis and Tolerance in Cryptography - FDTC 2012*, pages 7–15. IEEE Computer Society, 2012.

[85] J.-F. Dhem, F. Koeune, P.-A. Leroux, P. Mestré, J.-J. Quisquater, and J.-L. Willems. A Practical Implementation of the Timing Attack. In J.-J. Quisquater and B. Schneier, editors, *Proceedings of CARDIS*, pages 167–182. Springer, 1998.

[86] C. Diaz, O. Tene, and S. F. Ǵurses. Hero or Villain: The Data Controller in Privacy Law and Technologies. *Ohio State Law Journal*, 74(6):923–964, 2013.

[87] W. Diffie and M. E. Hellman. New directions in cryptography. *IEEE Transactions on Information Theory*, 22(6):644–654, 1976.

[88] V. S. Dimitrov, G. A. Jullien, and W. C. Miller. Complexity and Fast Algorithms for Multiexponentiations. *IEEE Trans. Computers*, 49(2):141–147, 2000.

[89] R. Dingledine, N. Mathewson, and P. F. Syverson. Tor: The Second-Generation Onion Router. In *USENIX Security Symposium 2004*, pages 303–320. USENIX Association, 2004.

[90] H. Dobbertin, A. Bosselaers, and B. Preneel. RIPEMD-160: A Strengthened Version of RIPEMD. In D. Gollmann, editor, *Fast Software Encryption - FSE 1996*, volume 1039 of *Lecture Notes in Computer Science*, pages 71–82. Springer, 1996.

[91] P. Dusart, G. Letourneux, and O. Vivolo. Differential Fault Analysis on A.E.S. In J. Zhou, M. Yung, and Y. Han, editors, *Applied Cryptography and Network Security - ACNS 2003*, volume 2846 of *Lecture Notes in Computer Science*, pages 293–306. Springer, 2003.

[92] S. Dziembowski and S. Faust. Leakage-Resilient Circuits without Computational Assumptions. In R. Cramer, editor, *Theory of Cryptography - TCC 2012*, volume 7194 of *Lecture Notes in Computer Science*, pages 230–247. Springer, 2012.

[93] S. Dziembowski and S. Faust. Leakage-Resilient Cryptography from the Inner-Product Extractor. In D. H. Lee and X. Wang, editors, *ASIACRYPT*, volume 7073 of *Lecture Notes in Computer Science*, pages 702–721. Springer, Advances in Cryptology - ASIACRYPT 2011.

[94] T. Eisenbarth, T. Kasper, A. Moradi, C. Paar, M. Salmasizadeh, and M. T. M. Shalmani. On the Power of Power Analysis in the Real World: A Complete Break of the KeeLoqCode Hopping Scheme. In D. Wagner, editor, *Advances in Cryptology - CRYPTO 2008*, volume 5157 of *Lecture Notes in Computer Science*, pages 203–220. Springer, 2008.

[95] J. Fan and I. Verbauwhede. An Updated Survey on Secure ECC Implementations: Attacks, Countermeasures and Cost. In D. Naccache, editor, *Cryptography and Security: From Theory to Applications - Essays Dedicated to Jean-Jacques Quisquater on the Occasion of His 65th Birthday*, volume 6805 of *Lecture Notes in Computer Science*, pages 265–282. Springer, 2012.

[96] B. Feix and A. Venelli. Defeating with Fault Injection a Combined Attack Resistant Exponentiation. In E. Prouff, editor, *Constructive Side-Channel Analysis and Secure Design - COSADE 2013*, volume 7864 of *Lecture Notes in Computer Science*, pages 32–45. Springer, 2013.

[97] J. Ferrigno and M. Hlavác. When AES blinks: introducing optical side channel. *IET Information Security*, 2(3):94–98, 2008.

[98] P.-A. Fouque, R. Lercier, D. Réal, and F. Valette. Fault Attack on Elliptic Curve Montgomery Ladder Implementation. In L. Breveglieri, S. Gueron, I. Koren, D. Naccache, and J.-P. Seifert, editors, *Fault Diagnosis and Tolerance in Cryptography - FDTC 2008*, pages 92–98. IEEE Computer Society, 2008.

[99] T. Fukunaga and J. Takahashi. Practical Fault Attack on a Cryptographic LSI with ISO/IEC 18033-3 Block Ciphers. In L. Breveglieri, I. Koren, D. Naccache, E. Oswald, and J.-P. Seifert, editors, *Fault Diagnosis and Tolerance in Cryptography - FDTC 2009*, pages 84–92. IEEE Computer Society, 2009.

[100] G. Fumaroli, A. Martinelli, E. Prouff, and M. Rivain. Affine Masking against Higher-Order Side Channel Analysis. In A. Biryukov, G. Gong, and D. R. Stinson, editors, *Selected Areas in Cryptography - SAC 2010*, volume 6544 of *Lecture Notes in Computer Science*, pages 262–280. Springer, 2010.

[101] K. Gandolfi, C. Mourtel, and F. Olivier. Electromagnetic Analysis: Concrete Results. In Çetin Kaya Koç, D. Naccache, and C. Paar, editors, *Cryptographic Hardware and Embedded Systems - CHES 2001*, volume 2162 of *Lecture Notes in Computer Science*, pages 251–261. Springer, 2001.

[102] F. D. Garcia, G. de Koning Gans, R. Muijrers, P. van Rossum, R. Verdult, R. W. Schreur, and B. Jacobs. Dismantling Mifare Classic. In S. Jajodia and J. Lopez, editors, *Proceedings of ESORICS 2008*, volume 5283 of *LNCS*, pages 97–114. Springer, 2008.

[103] F. D. Garcia, P. van Rossum, R. Verdult, and R. W. Schreur. Dismantling SecureMemory, CryptoMemory and CryptoRF. In A. Keromytis and V. Shmatikov, editors, *Proceedings of ACM CCS 2010*, pages 250–259. ACM Press, 2010.

[104] L. Genelle, C. Giraud, and E. Prouff. Securing AES Implementation against Fault Attacks. In L. Breveglieri, I. Koren, D. Naccache, E. Oswald, and J.-P. Seifert, editors, *Fault Diagnosis and Tolerance in Cryptography - FDTC 2009*, pages 51–62. IEEE Computer Society, 2009.

[105] L. Genelle, E. Prouff, and M. Quisquater. Secure Multiplicative Masking of Power Functions. In J. Zhou and M. Yung, editors, *Applied Cryptography and Network Security - ACNS 2010*, volume 6123 of *Lecture Notes in Computer Science*, pages 200–217. Springer, 2010.

[106] L. Genelle, E. Prouff, and M. Quisquater. Thwarting Higher-Order Side Channel Analysis with Additive and Multiplicative Maskings. In B. Preneel and T. Takagi, editors, *Cryptographic Hardware and Embedded Systems - CHES 2011*, volume 6917 of *Lecture Notes in Computer Science*, pages 240–255. Springer, 2011.

[107] B. Gierlichs, L. Batina, B. Preneel, and I. Verbauwhede. Revisiting Higher-Order DPA Attacks: Multivariate Mutual Information Analysis.

In J. Pieprzyk, editor, *Topics in Cryptology - CT-RSA 2010*, volume 5985 of *Lecture Notes in Computer Science*, pages 221–234. Springer, 2010.

[108] B. Gierlichs, L. Batina, P. Tuyls, and B. Preneel. Mutual Information Analysis. In E. Oswald and P. Rohatgi, editors, *Cryptographic Hardware and Embedded Systems - CHES 2008*, volume 5154 of *Lecture Notes in Computer Science*, pages 426–442. Springer, 2008.

[109] B. Gierlichs, K. Lemke-Rust, and C. Paar. Templates vs. Stochastic Methods. In L. Goubin and M. Matsui, editors, *Cryptographic Hardware and Embedded Systems - CHES 2006*, volume 4249 of *Lecture Notes in Computer Science*, pages 15–29. Springer, 2006.

[110] B. Gierlichs, J.-M. Schmidt, and M. Tunstall. Infective Computation and Dummy Rounds: Fault Protection for Block Ciphers without Check-before-Output. In A. Hevia and G. Neven, editors, *Progress in Cryptology - LATINCRYPT 2012*, volume 7533 of *Lecture Notes in Computer Science*, pages 305–321. Springer, 2012.

[111] C. Giraud. DFA on AES. In H. Dobbertin, V. Rijmen, and A. Sowa, editors, *AES Conference*, volume 3373 of *Lecture Notes in Computer Science*, pages 27–41. Springer, 2004.

[112] S. Goldwasser, S. Micali, and R. L. Rivest. A Digital Signature Scheme Secure Against Adaptive Chosen-Message Attacks. *SIAM Journal on Computing*, 17(2):281–308, 1988.

[113] J. D. Golic and C. Tymen. Multiplicative Masking and Power Analysis of AES. In B. S. K. Jr., Çetin Kaya Koç, and C. Paar, editors, *Cryptographic Hardware and Embedded Systems - CHES 2002*, volume 2523 of *Lecture Notes in Computer Science*, pages 198–212. Springer, 2002.

[114] L. Goubin and A. Martinelli. Protecting AES with Shamir's Secret Sharing Scheme. In B. Preneel and T. Takagi, editors, *Cryptographic Hardware and Embedded Systems - CHES 2011*, volume 6917 of *Lecture Notes in Computer Science*, pages 79–94. Springer, 2011.

[115] L. Goubin and J. Patarin. DES and Differential Power Analysis (The "Duplication" Method). In Çetin Kaya Koç and C. Paar, editors, *Cryptographic Hardware and Embedded Systems - CHES'99*, volume 1717 of *Lecture Notes in Computer Science*, pages 158–172. Springer, 1999.

[116] S. Govindavajhala and A. W. Appel. Using Memory Errors to Attack a Virtual Machine. In *IEEE Symposium on Security and Privacy*, pages 154–165. IEEE Computer Society, 2003.

[117] J. Großschädl, E. Oswald, D. Page, and M. Tunstall. Side-Channel Analysis of Cryptographic Software via Early-Terminating Multiplications. In D. Lee and S. Hong, editors, *ICISC*, volume 5984 of *Lecture Notes in Computer Science*, pages 176–192. Springer, 2010.

[118] M. Gruteser and B. Hoh. On the Anonymity of Periodic Location Samples. In D. Hutter and M. Ullmann, editors, *Security in Pervasive Computing - SPC 2005*, volume 3450 of *Lecture Notes in Computer Science*, pages 179–192. Springer, 2005.

[119] S. F. Gürses, C. Troncoso, and C. Diaz. Engineering Privacy by Design, booktitle = Computers, Privacy and Data Protection – CPDP 2011, year = 2011.

[120] J. A. Halderman, S. D. Schoen, N. Heninger, W. Clarkson, W. Paul, J. A. Calandrino, A. J. Feldman, J. Appelbaum, and E. W. Felten. Lest We Remember: Cold Boot Attacks on Encryption Keys. In P. C. van Oorschot, editor, *USENIX Security Symposium 2008*, pages 45–60. USENIX Association, 2008.

[121] L. Hemme. A Differential Fault Attack Against Early Rounds of (Triple-)DES. In M. Joye and J.-J. Quisquater, editors, *Cryptographic Hardware and Embedded Systems - CHES 2004*, volume 3156 of *Lecture Notes in Computer Science*, pages 254–267. Springer, 2004.

[122] B. Hoh, M. Gruteser, H. Xiong, and A. Alrabady. Enhancing Security and Privacy in Traffic-Monitoring Systems. *IEEE Pervasive Computing*, 5(4):38–46, 2006.

[123] M. Hutter and J.-M. Schmidt. The Temperature Side Channel and Heating Fault Attacks. In *Smart Card Research and Advanced Application - CARDIS 2013*, 2013. To appear.

[124] Y. Ishai, A. Sahai, and D. Wagner. Private Circuits: Securing Hardware against Probing Attacks. In D. Boneh, editor, *Advances in Cryptology - CRYPTO 2003*, volume 2729 of *Lecture Notes in Computer Science*, pages 463–481. Springer, 2003.

[125] M. Joye, A. K. Lenstra, and J.-J. Quisquater. Chinese Remaindering Based Cryptosystems in the Presence of Faults. *Journal of Cryptology*, 12(4):241–245, 1999.

[126] M. Joye, P. Paillier, and B. Schoenmakers. On Second-Order Differential Power Analysis. In J. R. Rao and B. Sunar, editors, *Cryptographic Hardware and Embedded Systems - CHES 2005*, volume 3659 of *Lecture Notes in Computer Science*, pages 293–308. Springer, 2005.

[127] M. Joye, J.-J. Quisquater, F. Bao, and R. H. Deng. RSA-type Signatures in the Presence of Transient Faults. In M. Darnell, editor, *Cryptography and Coding*, volume 1355 of *Lecture Notes in Computer Science*, pages 155–160. Springer, 1997.

[128] M. Joye and M. Tunstall, editors. *Fault Analysis in Cryptography*. Information Security and Cryptography. Springer, 2012.

[129] M. Joye and S.-M. Yen. The Montgomery Powering Ladder. In B. S. K. Jr., Çetin Kaya Koç, and C. Paar, editors, *Cryptographic Hardware and Embedded Systems - CHES 2002*, volume 2523 of *Lecture Notes in Computer Science*, pages 291–302. Springer, 2002.

[130] D. Karaklajic. *Securing Cryptographic Hardware Against Fault Attacks*. PhD thesis, KU Leuven, 2012.

[131] D. Karaklajic, J.-M. Schmidt, and I. Verbauwhede. Hardware Designer's Guide to Fault Attacks. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 21(12):2295–2306, 2013.

[132] M. G. Karpovsky, K. J. Kulikowski, and A. Taubin. Robust Protection against Fault-Injection Attacks on Smart Cards Implementing the Advanced Encryption Standard. In *Dependable Systems and Networks - DSN 2004*, pages 93–101. IEEE Computer Society, 2004.

[133] A. Kerckhoffs. La cryptographie militaire (military cryptography). *Journal des sciences militaires*, IX:5–83, 1883.

[134] C. H. Kim and J.-J. Quisquater. Fault Attacks for CRT Based RSA: New Attacks, New Results, and New Countermeasures. In D. Sauveron, C. Markantonakis, A. Bilas, and J.-J. Quisquater, editors, *Information Security Theory and Practices - WISTP 2007*, volume 4462 of *Lecture Notes in Computer Science*, pages 215–228. Springer, 2007.

[135] C. H. Kim, J. H. Shin, J.-J. Quisquater, and P. J. Lee. Safe-Error Attack on SPA-FA Resistant Exponentiations Using a HW Modular Multiplier. In K.-H. Nam and G. Rhee, editors, *Information Security and Cryptology - ICISC 2007*, volume 4817 of *Lecture Notes in Computer Science*, pages 273–281. Springer, 2007.

[136] H. Kim, S. Hong, and J. Lim. A Fast and Provably Secure Higher-Order Masking of AES S-Box. In B. Preneel and T. Takagi, editors, *Cryptographic Hardware and Embedded Systems - CHES 2011*, volume 6917 of *Lecture Notes in Computer Science*, pages 95–107. Springer, 2011.

[137] P. Koargonkar and L. Wolin. A Multivariate Analysis of Web Usage. *Journal of Advertising Research*, pages 53–68, March/April 1999.

[138] N. Koblitz. Elliptic Curve Cryptosystems. *Mathematics of Computation*, 48(177):203–209, 1987.

[139] P. Kocher. Leak Resistant Cryptographic Indexed Key Update. US Patent 6539092.

[140] P. C. Kocher. Timing Attacks on Implementations of Diffie-Hellman, RSA, DSS, and Other Systems. In N. Koblitz, editor, *Advances in Cryptology - CRYPTO '96*, volume 1109 of *LNCS*, pages 104–113. Springer, 1996.

[141] P. C. Kocher, J. Jaffe, and B. Jun. Differential Power Analysis. In M. J. Wiener, editor, *Advances in Cryptology - CRYPTO '99*, volume 1666 of *Lecture Notes in Computer Science*, pages 388–397. Springer, 1999.

[142] F. Koeune and J. jacques Quisquater. A timing attack against Rijndael. Technical Report CG-1999/1, Université catholique de Louvain, 1999.

[143] O. Kömmerling and M. G. Kuhn. Design principles for Tamper-Resistant Smartcard Processors. In *USENIX Workshop on Smartcard Technology*, pages 9–20. USENIX Association, 1999.

[144] J. Krämer, D. Nedospasov, A. Schlösser, and J.-P. Seifert. Differential Photonic Emission Analysis. In E. Prouff, editor, *Constructive Side-Channel Analysis and Secure Design - COSADE 2013*, volume 7864 of *Lecture Notes in Computer Science*, pages 1–16. Springer, 2013.

[145] J. Krumm. Inference Attacks on Location Tracks. In A. LaMarca, M. Langheinrich, and K. N. Truong, editors, *Pervasive Computing*, volume 4480 of *Lecture Notes in Computer Science*, pages 127–143. Springer, 2007.

[146] M. G. Kuhn. Optical Time-Domain Eavesdropping Risks of CRT Displays. In *IEEE Symposium on Security and Privacy*, pages 3–18. IEEE Computer Society, 2002.

[147] J. Lee and N. Pahl. Bypassing Smart-Card Authentication and Blocking Debiting: Vulnerabilities in Atmel CryptoMemory based Stored-Value Systems. DEFCON 18, 2010.

[148] V. Lomné, T. Roche, and A. Thillard. On the Need of Randomness in Fault Attack Countermeasures - Application to AES. In G. Bertoni and B. Gierlichs, editors, *Fault Diagnosis and Tolerance in Cryptography - FDTC 2012*, pages 85–94. IEEE Computer Society, 2012.

[149] T. Malkin, F.-X. Standaert, and M. Yung. A Comparative Cost/Security Analysis of Fault Attack Countermeasures. In L. Breveglieri, I. Koren, D. Naccache, and J.-P. Seifert, editors, *Fault Diagnosis and Tolerance in*

Cryptography - FDTC 2006, volume 4236 of *Lecture Notes in Computer Science*, pages 159–172. Springer, 2006.

[150] S. Mangard. A simple power-analysis (SPA) attack on implementations of the AES key expansion. In P. J. Lee and C. H. Lim, editors, *Information security and cryptology - ICISC'02*, volume 2587 of *LNCS*, pages 343–358, 2003.

[151] S. Mangard, E. Oswald, and T. Popp. *Power Analysis Attacks - Revealing the Secrets of Smart Cards.* Springer, 2007.

[152] S. Mangard, T. Popp, and B. M. Gammel. Side-Channel Leakage of Masked CMOS Gates. In A. Menezes, editor, *Topics in Cryptology - CT-RSA 2005*, volume 3376 of *Lecture Notes in Computer Science*, pages 351–365. Springer, 2005.

[153] S. Mangard, N. Pramstaller, and E. Oswald. Successfully Attacking Masked AES Hardware Implementations. In J. R. Rao and B. Sunar, editors, *Cryptographic Hardware and Embedded Systems - CHES 2005*, volume 3659 of *Lecture Notes in Computer Science*, pages 157–171. Springer, 2005.

[154] M. Matsui. Linear Cryptoanalysis Method for DES Cipher. In T. Helleseth, editor, *Advances in Cryptology - EUROCRYPT '93*, volume 765 of *Lecture Notes in Computer Science*, pages 386–397. Springer, 1993.

[155] M. Medwed and E. Oswald. Template Attacks on ECDSA. In K.-I. Chung, K. Sohn, and M. Yung, editors, *Workshop on Information Security Applications - WISA 2008*, volume 5379 of *Lecture Notes in Computer Science*. Springer, 2008.

[156] M. Medwed, C. Petit, F. Regazzoni, M. Renauld, and F.-X. Standaert. Fresh Re-keying II: Securing Multiple Parties against Side-Channel and Fault Attacks. In E. Prouff, editor, *Smart Card Research and Advanced Applications - CARDIS 2011*, volume 7079 of *Lecture Notes in Computer Science*, pages 115–132. Springer, 2011.

[157] M. Medwed, F.-X. Standaert, J. Großschädl, and F. Regazzoni. Fresh Re-keying: Security against Side-Channel and Fault Attacks for Low-Cost Devices. In D. J. Bernstein and T. Lange, editors, *Progress in Cryptology - AFRICACRYPT 2010*, volume 6055 of *Lecture Notes in Computer Science*, pages 279–296. Springer, 2010.

[158] A. J. Menezes, S. A. Vanstone, and P. C. V. Oorschot. *Handbook of Applied Cryptography.* CRC Press, Inc., 1996.

[159] T. S. Messerges. Securing the AES Finalists Against Power Analysis Attacks. In B. Schneier, editor, *Fast Software Encryption - FSE 2000*, volume 1978 of *Lecture Notes in Computer Science*, pages 150–164. Springer, 2000.

[160] T. S. Messerges. Using Second-Order Power Analysis to Attack DPA Resistant Software. In Çetin Kaya Koç and C. Paar, editors, *Cryptographic Hardware and Embedded Systems - CHES 2000*, volume 1965 of *Lecture Notes in Computer Science*, pages 238–251. Springer, 2000.

[161] T. S. Messerges, E. A. Dabbish, and R. H. Sloan. Investigations of power analysis attacks on smartcards. In *USENIX Workshop on Smartcard Technology - WOST '99*, pages 151–162. USENIX Association, 1999.

[162] T. S. Messerges, E. A. Dabbish, and R. H. Sloan. Examining Smart-Card Security under the Threat of Power Analysis Attacks. *IEEE Transactions on Computers*, 51(5):541–552, 2002.

[163] V. S. Miller. Use of Elliptic Curves in Cryptography. In H. C. Williams, editor, *Advances in Cryptology - CRYPTO '85*, volume 218 of *Lecture Notes in Computer Science*, pages 417–426. Springer, 1985.

[164] A. Moradi, O. Mischke, and T. Eisenbarth. Correlation-Enhanced Power Analysis Collision Attack. In S. Mangard and F.-X. Standaert, editors, *Cryptographic Hardware and Embedded Systems - CHES 2010*, volume 6225 of *Lecture Notes in Computer Science*, pages 125–139. Springer, 2010.

[165] S. Murphy and M. J. B. Robshaw. Essential Algebraic Structure within the AES. In M. Yung, editor, *Advances in Cryptology - CRYPTO 2002*, volume 2442 of *Lecture Notes in Computer Science*, pages 1–16. Springer, 2002.

[166] S. Nikova, V. Rijmen, and M. Schläffer. Secure Hardware Implementation of Non-linear Functions in the Presence of Glitches. In P. J. Lee and J. H. Cheon, editors, *Information Security and Cryptology - ICISC 2008*, volume 5461 of *Lecture Notes in Computer Science*, pages 218–234. Springer, 2008.

[167] K. Nohl, D. Evans, Starbug, and H. Plötz. Reverse-engineering a cryptographic RFID tag. In *Proceedings of USENIX 2008*, pages 185–193. USENIX Association, 2008.

[168] NVIDIA. Checklist for Building a PC that Plays HD DVD or Blu-ray Movies. `ftp://download.nvidia.com/downloads/pvzone/Checklist_for_Building_a_HDPC.pdf`.

[169] E. Oswald, S. Mangard, C. Herbst, and S. Tillich. Practical Second-Order DPA Attacks for Masked Smart Card Implementations of Block Ciphers. In D. Pointcheval, editor, *Topics in Cryptology - CT-RSA 2006*, volume 3860 of *Lecture Notes in Computer Science*, pages 192–207. Springer, 2006.

[170] E. Oswald, S. Mangard, N. Pramstaller, and V. Rijmen. A Side-Channel Analysis Resistant Description of the AES S-Box. In H. Gilbert and H. Handschuh, editors, *Fast Software Encryption - FSE 2005*, volume 3557 of *Lecture Notes in Computer Science*, pages 413–423. Springer, 2005.

[171] E. Peeters, F.-X. Standaert, and J.-J. Quisquater. Power and electromagnetic analysis: Improved model, consequences and comparisons. *Integration*, 40(1):52–60, 2007.

[172] G. Piret and J.-J. Quisquater. A Differential Fault Attack Technique against SPN Structures, with Application to the AES and KHAZAD. In C. D. Walter, Çetin Kaya Koç, and C. Paar, editors, *Cryptographic Hardware and Embedded Systems - CHES 2003*, volume 2779 of *Lecture Notes in Computer Science*, pages 77–88. Springer, 2003.

[173] R. A. Popa, H. Balakrishnan, and A. J. Blumberg. VPriv: Protecting Privacy in Location-Based Vehicular Services. In *USENIX Security Symposium 2009*, pages 335–350. USENIX Association, 2009.

[174] T. Popp, M. Kirschbaum, T. Zefferer, and S. Mangard. Evaluation of the Masked Logic Style MDPL on a Prototype Chip. In P. Paillier and I. Verbauwhede, editors, *Cryptographic Hardware and Embedded Systems - CHES 2007*, volume 4727 of *Lecture Notes in Computer Science*, pages 81–94. Springer, 2007.

[175] T. Popp and S. Mangard. Masked Dual-Rail Pre-charge Logic: DPA-Resistance Without Routing Constraints. In J. R. Rao and B. Sunar, editors, *Cryptographic Hardware and Embedded Systems - CHES 2005*, volume 3659 of *Lecture Notes in Computer Science*, pages 172–186. Springer, 2005.

[176] H. E. Products. Microsoft Zune HD 16GB, what's inside? `http://www2.electronicproducts.com/Microsoft_Zune_HD_16GB-whatsinside_text-89.aspx`.

[177] E. Prouff and M. Rivain. Theoretical and Practical Aspects of Mutual Information Based Side Channel Analysis. In M. Abdalla, D. Pointcheval, P.-A. Fouque, and D. Vergnaud, editors, *Applied Cryptography and*

Network Security - ACNS 2009, volume 5536 of *Lecture Notes in Computer Science*, pages 499–518. Springer, 2009.

[178] E. Prouff, M. Rivain, and R. Bevan. Statistical Analysis of Second Order Differential Power Analysis. *IEEE Transactions on Computers*, 58(6):799–811, 2009.

[179] E. Prouff, M. Rivain, and T. Roche. On the Practical Security of a Leakage Resilient Masking Scheme. Cryptology ePrint Archive, Report 2013/396, 2013. `http://eprint.iacr.org/`.

[180] E. Prouff and T. Roche. Higher-Order Glitches Free Implementation of the AES Using Secure Multi-party Computation Protocols. In B. Preneel and T. Takagi, editors, *Cryptographic Hardware and Embedded Systems - CHES 2011*, volume 6917 of *Lecture Notes in Computer Science*, pages 63–78. Springer, 2011.

[181] J.-J. Quisquater and C. Couvreur. Fast decipherment algorithm for RSA public-key cryptosystem. *Electronics Letters*, 18(21):905–907, 1982.

[182] J.-J. Quisquater and D. Samyde. ElectroMagnetic Analysis (EMA): Measures and Counter-Measures for Smart Cards. In I. Attali and T. P. Jensen, editors, *Research in Smart Cards - E-smart 2001*, volume 2140 of *Lecture Notes in Computer Science*, pages 200–210. Springer, 2001.

[183] J.-J. Quisquater and D. Samyde. Eddy current for Magnetic Analysis with Active Sensor. In *Esmart 2002, Nice, France*, 9 2002.

[184] C. Rechberger and E. Oswald. Practical Template Attacks. In C. H. Lim and M. Yung, editors, *Information Security Applications - WISA 2004*, volume 3325 of *Lecture Notes in Computer Science*, pages 440–456. Springer, 2004.

[185] O. Reparaz, B. Gierlichs, and I. Verbauwhede. Selecting Time Samples for Multivariate DPA Attacks. In E. Prouff and P. Schaumont, editors, *Cryptographic Hardware and Embedded Systems - CHES 2012*, volume 7428 of *Lecture Notes in Computer Science*, pages 155–174. Springer, 2012.

[186] A. Rial, J. Balasch, and B. Preneel. A Privacy-Preserving Buyer-Seller Watermarking Protocol Based on Priced Oblivious Transfer. *IEEE Transactions on Information Forensics and Security*, 6(1):202–212, 2011.

[187] M. Rivain. Differential Fault Analysis on DES Middle Rounds. In C. Clavier and K. Gaj, editors, *Cryptographic Hardware and Embedded Systems - CHES 2009*, volume 5747 of *Lecture Notes in Computer Science*, pages 457–469. Springer, 2009.

[188] M. Rivain and E. Prouff. Provably Secure Higher-Order Masking of AES. In S. Mangard and F.-X. Standaert, editors, *CHES*, volume 6225 of *Lecture Notes in Computer Science*, pages 413–427. Springer, Cryptographic Hardware and Embedded Systems, CHES 2010.

[189] R. L. Rivest, A. Shamir, and L. M. Adleman. A Method for Obtaining Digital Signatures and Public-Key Cryptosystems. *Communications of the ACM*, 21(2):120–126, 1978.

[190] P. Schaumont. *A Practical Introduction to Hardware/Software Codesign.* Springer, 2010.

[191] P. Schaumont and I. Verbauwhede. Domain-Specific Codesign for Embedded Security. *IEEE Computer*, 36(4):68–74, 2003.

[192] W. Schindler, K. Lemke, and C. Paar. A Stochastic Model for Differential Side Channel Cryptanalysis. In J. R. Rao and B. Sunar, editors, *Cryptographic Hardware and Embedded Systems - CHES 2005*, volume 3659 of *Lecture Notes in Computer Science*, pages 30–46. Springer, 2005.

[193] A. Schlösser, D. Nedospasov, J. Krämer, S. Orlic, and J.-P. Seifert. Simple Photonic Emission Analysis of AES - Photonic Side Channel Analysis for the Rest of Us. In E. Prouff and P. Schaumont, editors, *Cryptographic Hardware and Embedded Systems - CHES 2012*, volume 7428 of *Lecture Notes in Computer Science*, pages 41–57. Springer, 2012.

[194] J.-M. Schmidt and C. Herbst. A Practical Fault Attack on Square and Multiply. In L. Breveglieri, S. Gueron, I. Koren, D. Naccache, and J.-P. Seifert, editors, *Fault Diagnosis and Tolerance in Cryptography- FDTC 2008*, pages 53–58. IEEE Computer Society, 2008.

[195] J.-M. Schmidt and M. Hutter. Optical and EM Fault-Attacks on CRT-based RSA: Concrete Results. In J. W. Karl C. Posch, editor, *Austrian Workhop on Microelectronics - Austrochip 2007*, pages 61 – 67. Verlag der Technischen Universität Graz, 2007.

[196] J.-M. Schmidt, M. Tunstall, R. M. Avanzi, I. Kizhvatov, T. Kasper, and D. Oswald. Combined Implementation Attack Resistant Exponentiation. In M. Abdalla and P. S. Barreto, editors, *Progress in Cryptology - LATINCRYPT 2010*, volume 6212 of *Lecture Notes in Computer Science*, pages 305–322. Springer, 2010.

[197] K. Schramm, G. Leander, P. Felke, and C. Paar. A Collision-Attack on AES: Combining Side Channel- and Differential-Attack. In M. Joye and J.-J. Quisquater, editors, *Cryptographic Hardware and Embedded Systems - CHES 2004*, volume 3156 of *Lecture Notes in Computer Science*, pages 163–175. Springer, 2004.

[198] K. Schramm and C. Paar. Higher Order Masking of the AES. In D. Pointcheval, editor, *Topics in Cryptology - CT-RSA 2006*, volume 3860 of *Lecture Notes in Computer Science*, pages 208–225. Springer, 2006.

[199] K. Schramm, T. J. Wollinger, and C. Paar. A New Class of Collision Attacks and Its Application to DES. In T. Johansson, editor, *Fast Software Encryption - FSE 2003*, volume 2887 of *Lecture Notes in Computer Science*, pages 206–222. Springer, 2003.

[200] J.-P. Seifert. On authenticated computing and RSA-based authentication. In V. Atluri, C. Meadows, and A. Juels, editors, *Computer and Communications Security - CCS 2005*, pages 122–127. ACM, 2005.

[201] N. Selmane, S. Guilley, and J.-L. Danger. Practical Setup Time Violation Attacks on AES. In *European Dependable Computing Conference - EDCC 2008*, pages 91–96. IEEE Computer Society, 2008.

[202] A. Shamir. How to Share a Secret. *Communications of the ACM*, 22(11):612–613, 1979.

[203] A. Shamir and E. Tromer. Acoustic cryptanalysis. On nosy people and noisy machines. Web site: http://cs.tau.ac.il/ tromer/acoustic/ [Last accessed: 19/11/2013].

[204] S. P. Skorobogatov and R. J. Anderson. Optical Fault Induction Attacks. In B. S. K. Jr., Çetin Kaya Koç, and C. Paar, editors, *Cryptographic Hardware and Embedded Systems - CHES 2002*, volume 2523 of *Lecture Notes in Computer Science*, pages 2–12. Springer, 2002.

[205] Slashdot.org. DirecTV's Secret War On Hackers. Web site: http://slashdot.org/story/01/01/25/1343218/directvs-secret-war-on-hackers [Last accessed: 17/10/2013].

[206] F.-X. Standaert, T. Malkin, and M. Yung. A Unified Framework for the Analysis of Side-Channel Key Recovery Attacks. In A. Joux, editor, *Advances in Cryptology - EUROCRYPT 2009*, volume 5479 of *Lecture Notes in Computer Science*, pages 443–461. Springer, 2009.

[207] F.-X. Standaert, N. Veyrat-Charvillon, E. Oswald, B. Gierlichs, M. Medwed, M. Kasper, and S. Mangard. The World Is Not Enough: Another Look on Second-Order DPA. In M. Abe, editor, *Advances in Cryptology - ASIACRYPT 2010*, volume 6477 of *Lecture Notes in Computer Science*, pages 112–129. Springer, 2010.

[208] D. Suzuki, M. Saeki, and T. Ichikawa. Random Switching Logic: A Countermeasure against DPA based on Transition Probability. *IACR Cryptology ePrint Archive*, 2004:346, 2004.

[209] T. Telegraph. EU funding 'Orwellian' artificial intelligence plan to monitor public for "abnormal behaviour" . Web site: http://tinyurl.com/mr5mwj [Last accessed: 20/11/2013].

[210] S. Tillich and C. Herbst. Attacking State-of-the-Art Software Countermeasures-A Case Study for AES. In E. Oswald and P. Rohatgi, editors, *Cryptographic Hardware and Embedded Systems - CHES 2008*, volume 5154 of *Lecture Notes in Computer Science*, pages 228–243. Springer, 2008.

[211] K. Tiri, M. Akmal, and I. Verbauwhede. A dynamic and differential CMOS logic with signal independent power consumption to withstand differential power analysis on smart cards. In *Solid-State Circuits Conference - ESSCIRC 2002*, pages 403–406, 2002.

[212] K. Tiri and I. Verbauwhede. A Logic Level Design Methodology for a Secure DPA Resistant ASIC or FPGA Implementation. In *Design, Automation and Test in Europe - DATE 2004*, pages 246–251. IEEE Computer Society, 2004.

[213] E. Trichina, D. D. Seta, and L. Germani. Simplified Adaptive Multiplicative Masking for AES. In B. S. K. Jr., Çetin Kaya Koç, and C. Paar, editors, *CHES*, volume 2523 of *Lecture Notes in Computer Science*, pages 187–197. Springer, Cryptographic Hardware and Embedded Systems - CHES 2002.

[214] C. Troncoso, G. Danezis, E. Kosta, J. Balasch, and B. Preneel. PriPAYD: Privacy-Friendly Pay-As-You-Drive Insurance. *IEEE Transactions on Dependable and Secure Computing*, 8(5):742–755, 2011.

[215] C. Troncoso, G. Danezis, E. Kosta, and B. Preneel. PriPAYD: privacy friendly pay-as-you-drive insurance. In P. Ning and T. Yu, editors, *Privacy in the Electronic Society - WPES 2007*, pages 99–107. ACM, 2007.

[216] J. Tsai, S. Egelman, L. Cranor, and A. Acquisti. The Effect of Online Privacy Information on Purchasing Behavior: An Experimental Study, working paper. In *The 6th Workshop on the Economics of Information Security*, 2007.

[217] M. Tunstall, D. Mukhopadhyay, and S. Ali. Differential Fault Analysis of the Advanced Encryption Standard Using a Single Fault. In C. A. Ardagna and J. Zhou, editors, *Information Security Theory and Practice*

- *WISTP 2011*, volume 6633 of *Lecture Notes in Computer Science*, pages 224–233. Springer, 2011.

[218] L. Uhsadel, M. Ullrich, A. Das, D. Karaklajic, J. Balasch, I. Verbauwhede, and W. Dehaene. Teaching HW/SW Co-Design With a Public Key Cryptography Application. *IEEE Transactions on Education*, 56(4):478 – 483, 2013.

[219] J. G. J. van Woudenberg, M. F. Witteman, and F. Menarini. Practical Optical Fault Injection on Secure Microcontrollers. In L. Breveglieri, S. Guilley, I. Koren, D. Naccache, and J. Takahashi, editors, *Fault Diagnosis and Tolerance in Cryptography - FDTC 2011*, pages 91–99. IEEE Computer Society, 2011.

[220] J. VanLaven, M. Brehob, and K. J. Compton. Side Channel Analysis, Fault Injection and Applications - A Computationally Feasible SPA Attack on AES via Optimized Search. In R. Sasaki, S. Qing, E. Okamoto, and H. Yoshiura, editors, *Security and Privacy in the Age of Ubiquitous Computing - SEC 2005*, pages 577–588. Springer, 2005.

[221] I. Verbauwhede, D. Karaklajic, and J.-M. Schmidt. The Fault Attack Jungle - A Classification Model to Guide You. In L. Breveglieri, S. Guilley, I. Koren, D. Naccache, and J. Takahashi, editors, *Fault Diagnosis and Tolerance in Cryptography - FDTC 2011*, pages 3–8. IEEE, 2011.

[222] R. Verdult, F. D. Garcia, and J. Balasch. Gone in 360 Seconds: Hijacking with Hitag2. In *USENIX Security Symposium 2012*, pages 237–252. USENIX Association, 2012.

[223] N. Veyrat-Charvillon, M. Medwed, S. Kerckhof, and F.-X. Standaert. Shuffling against Side-Channel Attacks: A Comprehensive Study with Cautionary Note. In X. Wang and K. Sako, editors, *Advances in Cryptology - ASIACRYPT 2012*, volume 7658 of *Lecture Notes in Computer Science*, pages 740–757. Springer, 2012.

[224] H. Viksler. Web Laundry (In)Security . `http://ihackiam.blogspot.com/2010/09/web-laundry-insecurity.html`.

[225] J. Waddle and D. Wagner. Towards Efficient Second-Order Power Analysis. In M. Joye and J.-J. Quisquater, editors, *Cryptographic Hardware and Embedded Systems - CHES 2004*, volume 3156 of *Lecture Notes in Computer Science*, pages 1–15. Springer, 2004.

[226] D. Wagner. Cryptanalysis of a provably secure CRT-RSA algorithm. In V. Atluri, B. Pfitzmann, and P. D. McDaniel, editors, *Computer and Communications Security - CCS 2004*, pages 92–97. ACM, 2004.

[227] M. F. Witteman, J. G. J. van Woudenberg, and F. Menarini. Defeating RSA Multiply-Always and Message Blinding Countermeasures. In A. Kiayias, editor, *Topics in Cryptology - CT-RSA 2011*, volume 6558 of *Lecture Notes in Computer Science*, pages 77–88. Springer, 2011.

[228] P. Wright. *Spycatcher: The Candid Autobiography of a Senior Intelligence Officer*. Viking Press, 1987.

[229] L. Xiao and H. M. Heys. A simple power analysis attack against the key schedule of the Camellia block cipher. *Inf. Process. Lett.*, 95(3):409–412, 2005.

[230] S.-M. Yen and M. Joye. Checking Before Output May Not Be Enough Against Fault-Based Cryptanalysis. *IEEE Transactions on Computers*, 49(9):967–970, 2000.

[231] S.-M. Yen, D. Kim, and S.-J. Moon. Cryptanalysis of Two Protocols for RSA with CRT Based on Fault Infection. In L. Breveglieri, I. Koren, D. Naccache, and J.-P. Seifert, editors, *Fault Diagnosis and Tolerance in Cryptography - FDTC 2006*, volume 4236 of *Lecture Notes in Computer Science*, pages 53–61. Springer, 2006.

[232] S.-M. Yen, S. Kim, S. Lim, and S.-J. Moon. RSA Speedup with Residue Number System Immune against Hardware Fault Cryptanalysis. In K. Kim, editor, *Information Security and Cryptology - ICISC 2001,*, volume 2288 of *Lecture Notes in Computer Science*, pages 397–413. Springer, 2001.

[233] S.-M. Yen, S. Kim, S. Lim, and S.-J. Moon. A Countermeasure against One Physical Cryptanalysis May Benefit Another Attack. In K. Kim, editor, *Information Security and Cryptology - ICISC 2001*, volume 2288 of *Lecture Notes in Computer Science*, pages 414–427. Springer, 2002.

[234] J. F. Ziegler and W. A. Lanford. Effect of Cosmic Rays on Computer Memories. *Science*, 206(4420):776–788, 1979.

List of Publications

International Journals

[1] L. Uhsadel, M. Ullrich, A. Das, D. Karaklajic, J. Balasch, I. Verbauwhede, and W. Dehaene, "Teaching HW/SW co-design with a public key cryptography application", In *IEEE Transactions on Education - IEEE ToE 56(4)*, pages 478 - 483. IEEE, 2013.

[2] A. Rial, J. Balasch, and B. Preneel, "A Privacy-Preserving Buyer–Seller Watermarking Protocol Based on Priced Oblivious Transfer", In *IEEE Transactions on Information Forensics and Security - IEEE TIFS 6(1)*, pages 202-212. IEEE, 2011.

[3] C. Troncoso, G. Danezis, E. Kosta, J. Balasch, and B. Preneel, "PriPAYD: Privacy Friendly Pay-As-You-Drive Insurance (Journal version)", In *IEEE Transactions on Dependable and Secure Computing - IEEE TDSC 8(5)*, pages 742-755. IEEE, 2011.

International Conferences

[1] J. Balasch, S. Faust, B. Gierlichs, and I. Verbauwhede, "Theory and Practice of a Leakage Resilient Masking Scheme", In *Advances in Cryptology - ASIACRYPT 2012*, Volume 7658 of Lecture Notes in Computer Science (LNCS), pages 758-775. Springer-Verlag, 2012.

[2] J. Balasch, B. Ege, T. Eisenbarth, B. Gérard, Z. Gong, T. Güneysu, S. Heyse, S. Kerckhof, F. Koeune, T. Plos, T. Pöppelmann, F. Regazzoni, F-X. Standaert, G. Van Assche, R. Van Keer, L. van Oldeneel tot Oldenzeel, and I. von Maurich, "Compact Implementation and Performance Evaluation of Hash Functions in ATtiny Devices", In *Proceedings of the 11th Smart Card Research and Advanced Application Conference - CARDIS 2012*,

Volume 7771 of Lecture Notes in Computer Science (LNCS), pages 158-172. Springer-Verlag, 2012.

[**3**] R. Verdult, F. D. Garcia, and J. Balasch, "Gone in 360 Seconds: Hijacking with Hitag2", In *Proceedings of the 21st Usenix Security Symposium - USENIX Security 2012*, pages 237-252. USENIX Association, 2012.

[**4**] J. Balasch, B. Gierlichs, R. Verdult, L. Batina, and I. Verbauwhede, "Power Analysis of Atmel CryptoMemory - Recovering Keys from Secure EEPROMs", In *Topics in Cryptology - CT-RSA 2012, The Cryptographers' Track at the RSA Conference*, Volume 7178 of Lecture Notes in Computer Science (LNCS), pages 19-34. Springer-Verlag, 2012.

[**5**] J. Balasch, B. Gierlichs, and I. Verbauwhede, "An in-Depth and black-Box Characterization of the Effects of Clock Glitches on 8-bit MCUs", In *Proceedings of the 8th International Workshop on Fault Diagnosis and Tolerance in Cryptography - FDTC 2011*, pages 105-114. IEEE, 2011.

[**6**] J. Balasch, A. Rial, C. Troncoso, C. Geuens, B. Preneel, and I. Verbauwhede, "PrETP: Privacy-Preserving Electronic Toll Pricing", In *Proceedings of the 19th Usenix Security Symposium - USENIX Security 2010*, pages 63-78. USENIX Association, 2010.

[**7**] J. Balasch, I. Verbauwhede, and B. Preneel, "An Embedded Platform for Privacy-Friendly Road Charging Applications", In *Proceedings of Design, Automation and Test in Europe - DATE 2010*, pages 867-872. IEEE, 2010.

Technical Reports

[**1**] W. Biesmans, J. Balasch, A. Rial, and I. Verbauwhede, "Priced Oblivious Transfer Scheme for Pay-per-view Broadcast Mobile TV", COSIC internal report, 6 pages, 2013.

[**2**] M. Scheir, J. Balasch, A. Rial, B. Preneel, and I. Verbauwhede, "Anonymous E-Cash for Resource Constrained Devices", COSIC internal report, 10 pages, 2012.

[**3**] B. Car, J. Balasch, A. Rial, B. Preneel, and I. Verbauwhede, "A Zero-Knowledge Proofs Compiler for Hardware-Software Co-Design", COSIC internal report, 20 pages, 2012.

Curriculum

Josep Balasch was born on the 3rd of July 1982, in Vic, Spain. In 2008 he obtained a Master in Telecommunication Engineering from the Polytechnic University of Catalonia, Barcelona, Spain. His MSc Thesis entitled "Smart Card Implementation of Anonymous Credentials" was carried out at the KU Leuven, Belgium, as part of the Erasmus exchange framework.

In September 2009, he joined the COSIC research group at the department of Electrical Engineering (ESAT) of KU Leuven. His research has been funded by a joint PhD grant between the universities KU Leuven, Belgium, and Radboud Universiteit Nijmegen, the Netherlands.

Between July and October 2012 he was a graduate technical intern at the Intel's Security Center of Excellence (SeCoE) in Hillsboro, Oregon (United States).