PhD-FSTC-2011-08
The Faculty of Sciences, Technology and Communication

DISSERTATION

Defense held on 29/06/2011 in Luxembourg
to obtain the degree of

DOCTEUR DE L'UNIVERSITÉ DU LUXEMBOURG
EN INFORMATIQUE

by

Ilya KIZHVATOV
Born on 22 July 1982 in Barnaul (USSR)

Physical Security of Cryptographic Algorithm Implementations

Dissertation defense committee

Dr. Alex Biryukov, vice chairman
Associate Professor, Université du Luxembourg

Dr. Jean-Sébastien Coron, dissertation supervisor
Associate Professor, Université du Luxembourg

Dr. Volker Müller, chairman
Associate Professor, Université du Luxembourg

Dr. Emmanuel Prouff
Cryptography & Security Research Manager, Oberthur Technologies, France

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

Abstract

This thesis deals with physical attacks on implementations of cryptographic algorithms and countermeasures against these attacks. Physical attacks exploit properties of an implementation such as leakage through physically observable parameters (side-channel analysis) or susceptibility to errors (fault analysis) to recover secret cryptographic keys. In the absence of adequate countermeasures such attacks are often much more efficient than classical cryptanalytic attacks. Particularly vulnerable to physical attacks are embedded devices that implement cryptography in a variety of security-demanding applications.

In the area of side-channel analysis, this thesis addresses attacks that exploit observations of power consumption or electromagnetic leakage of the device and target symmetric cryptographic algorithms (at the notable example of the Advanced Encryption Standard (AES)). First, this work proposes a new combination of two well-known techniques of such attacks: differential side-channel analysis and side-channel collision attacks. The combination is more efficient than each of the attacks individually. As a further improvement, new dimension reduction techniques for side-channel acquisitions are introduced for side-channel collision detection and compared using an information-theoretic metric. Second, this work studies attacks exploiting leakage induced by microprocessor cache mechanism. We present an algorithm for cache-collision attacks that can recover the secret key in the presence of uncertainties in cache event detection from side-channel acquisitions, which may happen in a noisy measurement environment. Third, practical side-channel attacks are discovered against the AES engine of the AVR XMEGA, a recent versatile microcontroller for a variety of embedded applications.

In the area of fault analysis, this thesis extends existing attacks against the RSA digital signature algorithm implemented with the Chinese remainder theorem to a setting where parts of the signed message are unknown to the attacker. The new attacks are applicable in particular to the randomized ISO/IEC 9796-2 encoding variant used in the EMV standard, and to the PKCS#1 v1.5 standard in the setting when the message is totally unknown. Both standards are widely used in modern smart card applications.

In the area of countermeasures, this work proposes a new algorithm for random delay generation in embedded software. Random delays can be inserted into the execution flow of a cryptographic algorithm to break synchronization in physical attacks and therefore increase their complexity. The new algorithm is based on the idea of generating individual random delays non-independently. It is more efficient than the previously suggested algorithms since it introduces more uncertainty for the attacker with less performance overhead.

The results presented in this thesis are practically validated in experiments with general-purpose 8-bit AVR and 32-bit ARM microcontrollers that are used in many embedded devices.

Acknowledgements

This thesis would not have existed without help and support of many people.

I would like to thank my co-advisors Alex Biryukov and Jean-Sébastien Coron, who accepted me as a Ph.D. student and have offered their guidance and advice throughout the 3.5 years of my work. They have set an excellent example of top class research and have given the degree of academic freedom that encouraged independent exploration. I am grateful to Emmanuel Prouff and François-Xavier Standaert for serving on the thesis jury, and to Volker Müller for chairing the jury.

I am profoundly indebted to Vladimir Anashin and Alexandr Kalinovsky for introducing me to cryptography and physics, respectively, during my undergraduate studies at the Russian State University for the Humanities..

I greatly appreciate effective collaborations with the external co-authors: Roberto Avanzi, Andrey Bogdanov, Antoine Joux, Timo Kasper, David Naccache, David Oswald, Pascal Paillier, Andrey Pyshkin, Jörn-Marc Schmidt, and Michael Tunstall. My discussions will the participants of the ECRYPT VAM2 research retreats were both fruitful and inspiring.

My many thanks go to the colleagues at the Laboratory of Algorithmics, Cryptology and Security of the University of Luxembourg for the superb working environment. I have enjoyed doing research with my co-authors Jean-François Gallais, Johann Großschädl, Zhe Liu, and Bin Zhang. André Stemper provided the top class technical assistance with the measurement setup. I would like to thank Dmitry Khovratovich and Ivica Nikolić for countless scientific discussions and after-work activities. It was a pleasure to be on the team with David Galindo, Gaëtan Leurent, Avradip Mandal, Arnab Roy, Deike Priemuth-Schmid, and Ralf-Philipp Weinmann. I appreciate the help of LACS secretaries Mireille Kies, Ragga Eyjolfsdottir, and Fabienne Schmitz with administrative tasks and paperwork. I am grateful to the rector Rolf Tarrach for his dedicated hours for listening to students' problems and suggestions.

It was a great experience to be with the Instrumental Ensemble of the University of Luxembourg conducted by Ivàn Boumans, and with the climbing group lead by Gaston Malané.

I am indebted to my great friends Aliena, Andrey, Katya, Lena, Lena, Marina, Nastya, Oleg, Tolik, and Yulia, with whom we were travelling, who were visiting me in Luxembourg, providing accommodation, or just staying in touch.

My deepest appreciation goes to Olga for her patience and support.

The strongest support and encouragement have come from my family. My deepest gratitude is meant for them.

Ilya Kizhvatov, July 2011

Contents

List of Tables

List of Figures

Chapter 1

Introduction

This thesis deals with implementation security aspects of cryptography. It is not about how to design a secure cipher, but rather how to *implement* an existing one so that the implementation is still secure. Physical security of a cryptographic implementation along with an adequate speed and size are often harder to achieve today than pure algorithmic security. Practical importance of implementation issues has led in the recent decades to the development of cryptographic engineering, a distinct discipline concerned particularly with security and efficiency of cryptographic implementations.

Contents

1.1 Cryptography and applications

For centuries *cryptography* (initially the art of keeping information secret) was done with paper and pencil or at most with appliances like the ancient Greek scytale or the Girolamo Cardano's XIV century grille. The amounts of transferred information were small. From the modern perspective ciphers were rather unsophisticated but it must have been far easier to eavesdrop a secret rather than to recover it from an encrypted message (a task for *cryptanalysis*). In the beginning of the XX century, the advent of telecommunications caused a dramatic increase in the amount of exchanged information and in the rate of its transmission. Naturally, *cryptographic hardware* appeared in the form of mechanical and electromechanical machines that performed semi-automated or automated encryption of a stream of symbols, with the most famous example to be the German Enigma.

Cryptography however was still the prerogative of the military and the diplomatic corps. Security was of paramount importance at the time of the two World Wars and the Cold War because technical development, and especially advances in radiocommunications, made sophisticated means and techniques of eavesdropping possible, which spies immediately employed. Computers were created in defense and intelligence centers to "break" increasingly strong ciphers. It was military and intelligence engineers who first discovered that eavesdropping can be successfully applied not only to voice communications but also against cryptographic hardware, which made it possible to recover secrets where mathematical methods of "breaking" ciphers alone were failing.

At the end of the XX century, computers became much smaller and cheaper and made their way to the public market. At the same time, the end of the Cold War led to the relaxation of the major diplomatic strain. State regulations were loosened and cryptography followed computers.

Today cryptography has evolved beyond just ciphers. In addition to secrecy (or more formally *confidentiality*), cryptography provides *data integrity*, *authentication*, *electronic signatures*, *anonymity*, *access control* and many other security objectives. Applications utilize different *cryptographic protocols*, e.g. Transport Layer Security (TLS) protocol that is used in a web browser or an e-mail client when connecting to a personal electronic mailbox on a remote server which in turn use *cryptographic algorithms*, for instance Advanced Encryption Standard (AES) cipher or Rivest, Shamir and Adleman (RSA) algorithm to achieve the above objectives.

Cryptographic algorithms run on many types of computing devices with various security features. Enterprise servers and personal computers are first to come to one's mind, but there are numerous *embedded devices*—compact appliances with an on-board computer serving a dedicated task, for instance, electronic keys, radio frequency identification (RFID) tags, smart cards[1], mobile phones, wireless routers. Modern transport vehicles are literally stuffed with microcomputers, and embedded devices are spread even more widely in industrial applications. The recent term *pervasive computing* describes this situation well. Computing platforms are diverse, including pure hardware as application-specific integrated circuits (ASIC), reconfigurable hardware (field-programmable gate arrays, FPGA), software running on a microprocessor, and combined solutions.

Several concrete examples of embedded devices along with their secure applications are:

- subscriber identity module (SIM) card that personalizes a mobile handset, enables access to the cellular network and provides privacy of voice and data sent over the air;

---

[1] Smart cards are usually considered separately from more complex embedded devices, but for now we omit this difference.

- a bank card with a built-in chip, used to withdraw cash from an ATM or to pay in a shop;
- an electronic passport with a built-in contactless chip;
- an electronic key used to open a car and start its engine.
- a pay-TV set-top box that provides the subscriber with set of TV channels that she paid for;
- a smart meter that records consumption of electric energy in a household or a company building and regularly sends the results to the central energy office;
- a sensor node for environmental surveillance that transfers data securely over the air;
- an industrial or medical apparatus with a secure firmware update to prevent unauthorized modifications.

1.2 Research motivation

Advances in cryptography currently enable building cryptographic algorithms that are reasonably secure against a variety of purely mathematical *cryptanalytic attacks*. However there are two problems when it comes to applications.

First, cryptographic algorithms should be fast but consume a reasonable amount of device resources, and the overall cost of the system must be adequate. Second, modern electronic devices that implement cryptography are more advanced than previous electromechanical machines, but they are still vulnerable to physical eavesdropping and tampering.

Physical cryptanalytic attacks pose a particular threat to embedded devices that implement cryptography, since embedded devices are almost always exposed to the environment that is out of the owner's control and is therefore considered untrusted. For example, a payment terminal accepting bank cards in a shop can be modified in a malicious way to extract information from the card, and this can be later used for stealing money from the owner's account, and real cases of such fraud have taken place in the recent years. Conventional computers such as desktops and servers are also at risk: they are exposed to the network, and this situation makes possible to remotely observe execution timing and protocol messages which can be used in a more general class of *implementation attacks*.

In general, there is an inherent trade-off between security (both algorithmic and physical) and efficiency of cryptographic implementations. Expensive, slow and insecure systems will equally likely lose in the market competition. Today, designing physically secure and efficient implementations of cryptographic algorithms is a challenge for cryptography and cryptographic engineering in particular. In this thesis, we specifically address implementation security.

1.3 Implementation security

In a nutshell, an implementation attack can be explained by an example of lock picking from [57]. Let us assume that we are to open a safe protected by a combination lock of several discs. A straightforward way to open the lock is to start trying every combination of the discs' positions. At some point we will succeed, but this might take a long time. The process can be sped up with a stethoscope: due to the mechanics of some locks a click will be heard when a single disc is placed in a correct position. The full correct combination can be recovered sequentially by listening to the individual discs. A similar approach can be used for picking ordinary locks: the small backlash between the lock parts usually makes it possible to determine the correct positions of all individual pins independently. The unintentional (for the lock designers) acoustic signal or mechanical response coming from the lock enables us to open the lock. The lock physically leaks information related to the correct combination and we exploit this leakage to find that combination. Same things take place when we turn to cryptography.

1.3.1 Historical and modern examples

To show what implementation attacks are in practice, how they evolved and how diverse they are, we give several examples. We refer the reader to [90] and [143] for a more detailed historical overview.

WWI: Eavesdropping field telephones. The earliest reported example of exploiting compromising emanations dates back to the World War I and is described in [13, 166]. The German army was successfully overhearing telephone communications of the adversaries from a distance using ground currents. This was possible since the field phones were connected only with a single insulated wire to reduce the weight of the cabling; the return current was flowing through the ground. Eavesdroppers were using a valve amplifier to pick up a voltage drop between the two "search electrodes" put into the ground close to the front line, on the other side of which an adversary's observer with a phone would usually be located. Though this was eavesdropping of voice communications rather than an attack against a cryptographic implementation, the same principle is used nowadays in side-channel attacks where a voltage drop-off is measured over a resistor inserted in the ground line of a target device.

As a technical protective measure, field telephone circuits were converted to a twisted pair cable configuration. So this historical case also serves as an early example of the trade-off between design cost (cable length and weight of cable drums) and security, and of a countermeasure.

WWII: Bell Labs electromagnetic side-channel attack. This example is from a recently declassified NSA paper [154]. The papers reports that a researcher noticed in 1943, quite by accident, the electromagnetic leakage of a Bell-telephone mixing device 131-B2 which was used during World Was II for encryption in the backbone systems for the US Army and Navy. Each step of the machine was causing a spike on an oscilloscope in a distant part of the lab. By examining the spikes, it was possible to read the plaintext encrypted by the device. This was verified in a dedicated experiment at a distance of about 25 meters.

Bell Labs explored the phenomenon and suggested the basic suppression measures (shielding, filtering and masking), which however made the machine bulky and rendered it difficult to use, so in practice it was recommended just to keep a 100-feet controlled zone around the working machine. After the war the problem was apparently forgotten but then rediscovered in 1951 by the Central Intelligence Agency (CIA).

MI5/GCHQ acoustic side-channel attacks. The next example is described in [162] and dates back to 1956, when the British intelligence service MI5 together with GCHQ (Government Communications Headquarters) tried to decipher messages sent from the Egyptian Embassy in London and encrypted with Hagelin machine. The intelligence figured out that the initial positions of several Hagelin wheels (a parameter crucial in cryptanalysis) could have been determined by registering the sound of the machine during the reset of the wheels with an acoustic microphone and an oscilloscope. The hidden microphone was installed in the Embassy, which made possible reading the Egyptian cipher during the Suez Crisis. The same technique was used by MI5/GCHQ in 1960-63 to read a French cipher from the French Embassy in London.

TEMPEST in the USA. In 1950s, US government started a classified program under the name TEMPEST. This program was aimed at studying and preventing compromising emissions. Later the word became a synonym for compromising emanations. The TEMPEST was the source of the red/black separation principle, red standing for classified, black—for unclassified data. According to [90], by 1990 the yearly market of TEMPEST-certified equipment in the US exceeded 1 billion dollars. TEMPEST standards that contain concrete details still remain classified. A collection of declassified documents is available [110].

Public attention to compromising emanations was drawn by van Eck in his paper [157] that describes a practical technique of cathode ray tube (CRT) display contents eavesdropping from a distance.

Implementation attacks in open research and industry. In the open community, intensive study of implementation attacks started in 1990s. In 1996 Boneh, DeMillo and Lipton reported an ingenious attack against RSA based on introducing a single error into the computation and exploiting the erroneous result [29]. This *fault attack* worked against RSA implementations that used Chinese remainder theorem for efficiency reasons. In 1997 Kocher presented attacks against RSA and several other cryptographic algorithms [87]. These *timing attacks* exploited variations in the execution time of the algorithms that depended on the secret data. In 1999, he proposed an attack that exploited multiple observations of the power consumption of a smart card to recover the secret encryption key [88]. This *differential* *power analysis attack* (DPA) by now remains the most practical and inexpensive physical attack against many devices implementing cryptography. Since then implementation attacks and countermeasures have been a prominent topic both in research and in industry; being today the leading topics of CHES (Cryptographic Hardware and Embedded Systems), FDTC (Fault Detection and Tolerance in Cryptography) and COSADE (Constructive Side-Channel Analysis and Secure Design) workshops. After 10 years of research in the area, a book about power analysis of smart cards was published [101], followed by several other books on cryptographic engineering covering the subject of implementation security [34, 159, 62].

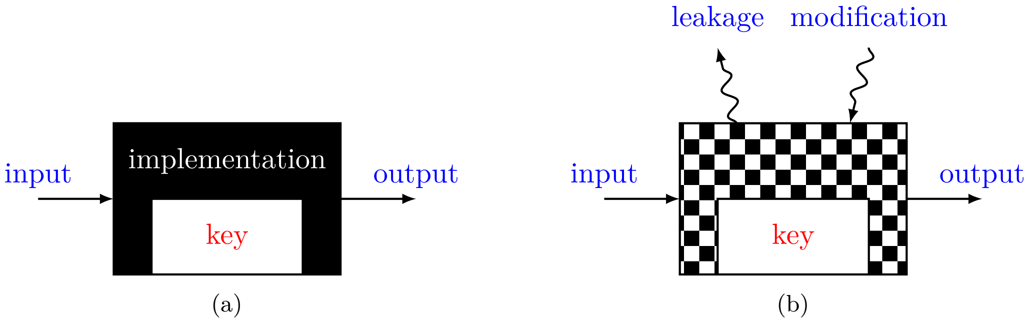
Information leakage from optical emanations. In 2002, paper [96] reported that LED status indicators on data communication equipment, under certain conditions, were shown to carry a modulated optical signal that was significantly correlated with information being processed by the device. The attacker could gain access to all data going through the device, including plaintext in the case of data encryption systems. Experiments showed that it was possible to intercept data under realistic conditions at a considerable distance. Many different sorts of devices, including modems and Internet Protocol routers, were found to be vulnerable.

Leakage of encrypted VoIP communications. This recent example demonstrates that not only devices but also protocols may exhibit leakage. In 2007 paper [161] showed that the encrypted Voice over IP network traffic carries enough information to reliably identify the language of a conversation. Another work [50] demonstrated on the concrete example of a widely used VoIP application (Skype) that it is possible to classify isolated phonemes and identify specific sentences.

1.3.2 Classical and physical cryptanalytic scenarios

Cryptographic algorithms are designed to withstand cryptanalysis following the keystone rule formulated in the end of the XIX-th century and known as (second) Kerckhoffs' principle [79], which states that a cryptosystem should be secure even if everything about the system, except the *key*, is public knowledge. A cryptographic key is a piece of information which in the case of a cipher is used to uniquely encrypt a *plaintext* to obtain the *ciphertext*, and perform the inverse action. Among the keyed algorithms, there are *symmetric* cryptographic algorithms where encryption and decryption are performed with the same secret key, and *public key* cryptographic algorithms that use a pair of a public and a private key. Secret keys and private keys are targets of key recovery attacks, with which this work deals. There are unkeyed cryptographic algorithms, e.g. hash functions, but we do not consider them in this work, though physical attacks on some of them, for instance on physical random number generators, are possible.

A *brute force* attack by trying every key against a plaintext-ciphertext pair is always possible against cryptographic algorithms. So one speaks about an attack when the complexity expressed in some basic operations such as encryption algorithm executions or processor clock ticks is below the brute force complexity. With the modern computational resources, attacks are practical only up to about 2^{64} basic operations (this is a rough estimate, for more details refer to [55]). Typical key sizes for symmetric algorithms range today from 80 to 256 bits. Since a large amount of keys of such length can hardly be memorized by humans, they are stored within the device implementing an algorithm.



(a) (b)

Figure 1.1: Classical (a) and physical (b) cryptanalytic scenarios

In the *classical cryptanalytic scenario* an attacker tries to recover the secret key knowing the algorithm description and having (depending on the concrete scenario) some inputs to the algorithm, the corresponding outputs, or both. A physical implementation of an algorithm is considered to be a *black box* in the sense that only inputs and outputs are observed, but not the internal variables or any other information about what is happening during the execution.

In contrary to the classical scenario, in the *physical cryptanalysis scenario* an adversary observes or manipulates some physical parameters of the device running the algorithm. For instance, one can observe power consumption, electromagnetic radiation, execution duration, or tamper with the supply voltage or the clock signal. Since the access is typically not complete, a term *grey box* is sometimes used, though not being quite correct, since the box in this case is rather semi-transparent, which reveals some part of the inner workings. A usual target of implementation attacks is to extract a secret key which is stored by the device. The two scenarios are schematically shown in Figure 1.1.

Note that for the practical implementation security the violation of the Kerckhoffs' principle is justified if done properly. Indeed, the lack of knowledge about the implementation in most cases would require an adversary to spend some non-negligible effort on reverse-engineering. But obscurity in this case should be an additional, and not the *only* way to achieve security. In fact, recovering unknown cryptographic algorithms by reverse engineering is the other target of physical attacks. Note also that countermeasures, for instance randomization of the execution time, do not count as such violation.

1.3.3 Classification of implementation attacks

All implementation attacks fall into two large groups depending on whether an adversary just observes some parameters of the implementation or influences its execution. *Passive implementation attacks* exploit results of observing an implementation while it works (largely) as intended. The adversary can feed inputs but does not interfere with the execution of an algorithm or a protocol. This group includes *side-channel attacks*, where physical leakage of a device during an algorithm execution is observed, and higher-level *logical attacks*, where some parameters of a protocol execution, for instance error messages, are observed. *Active implementation attacks* exploit results of influencing an implementation, or manipulating with it, such that it behaves abnormally. They comprise *fault attacks* where errors are introduced into an execution of a cryptographic algorithm or a protocol, and other attacks that do not directly exploit the properties of a cryptographic algorithm or a protocol, such as dumping the device's memory that contains the secret key.

Another dimension of classification concerns the level of intrusion into an implementation. There are *non-invasive* attacks where a device remains intact and only the environmental parameters are observed or changed, *semi-invasive* attacks where device's case or package is removed, for instance a microchip is *decapsulated*, to get better access to the internals, and *invasive* attacks where a direct contact to device internals is established.

These two dimensions are not independent in practice, and the classification is not complete. side-channel attacks are mainly non-invasive, whereas semi-invasive and invasive attacks are mainly active, but there are special cases. For instance, attacks exploiting electromagnetic radiation of a device often benefit from partially removing the package of an integrated circuit that enables to register stronger fields close to the silicon chip surface. *Probing attacks* [69] where the adversary physically connects to wires of an integrated circuit (decapsulating the device and cutting into the silicon chip) to read the transferred bits, can hardly be categorized as either side-channel (since this is a direct connection) or fault attacks (no errors are introduced). Attacks exploiting *data remanence* [68, 146] are another special category of implementation attacks. *Bug attacks* [19] that exploit production- or design-stage defects of a device are similar to fault attacks but are passive. Furthermore, active and passive techniques can be combined [4, 40, 147].

A large subset of implementation attacks are *physical attacks* that are characterized by a physical contact with a device. Since this mainly means proximity to the device, side-channel attacks observing timing variations of the implementation remotely over the network can be viewed as a special case.

Figure 1.2 shows the approximate taxonomy of implementation attacks. The attacks with which this thesis deals are highlighted in grey. They are electromagnetic and power analysis side-channel attacks, and non-invasive fault attacks (the countermeasure developed in this thesis is also applicable to other implementation attacks).

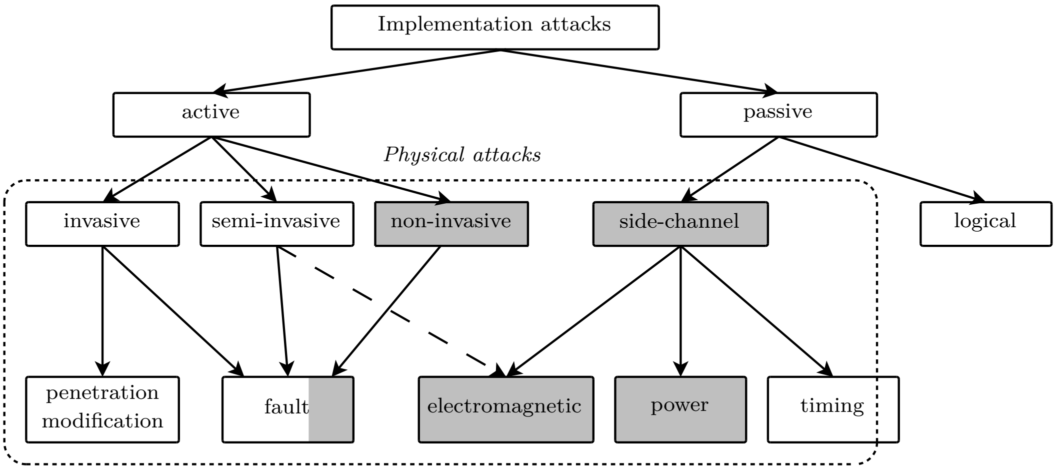


Figure 1.2: Approximate taxonomy of implementation attacks (based on [54])

1.3.4 Side-channel analysis

Side-channel attacks are passive attacks where the adversary only observes the physical parameters of the device and uses the observations to perform the key recovery. Despite the fact that such scenario is weaker than active attacks where the adversary can influence the execution of a cryptographic algorithm, side-channel attacks usually cost less since they require cheaper equipment. Another advantage of side-channel attacks is that they can be hard to detect since they are mainly non-invasive and do not interfere with the normal device operation.

For many details on side-channel analysis, we refer the reader to the book [101], which reflects the state of the art in the field for the year 2007. Books [159] and [34] both contain introductory chapters on side-channel attacks; another overview is given in a recent paper [86].

Leakage sources and leakage models

Physical observables that are usually exploited in side-channel attacks are timing [87], power consumption [88], and electromagnetic radiation [122]. This is because they are relatively easy to register for a variety of devices. Other more specific leakage types were exploited in research papers, for instance, acoustic [145], optical [56, 148] and thermal [32], but to our knowledge they are rarely used in practice for attacking electronic devices.

Attack algorithms differ depending on the leakage source and not on the physical leakage type, since different sources leak different kinds of information through the same physical observables. So a side channel can be viewed as a combination of a leakage source with the physical leakage type.

An often exploited source is switching gates and wires of electronic circuits, which leak information about the processed internal variables through power consumption and electromagnetic radiation. The (data-dependent) sequence of operations of an algorithm is another source. It can leak information both through timing and through power consumption and EM radiation. Yet another source is the features of a microprocessor such as cache [115], branch prediction unit [1], or early terminating instructions [66] all exploited by a special group of *microarchitectural side-channel attacks.* The leakage from these sources can be observed in power, electromagnetic, and timing side channels. Note that power and electromagnetic leakages are linked, since moving electric charges in the device is a source of electromagnetic radiation.

Some attacks require some knowledge of the dependency between the value of an internal variable processed by the algorithm and the corresponding physical observable. This dependency is called a *leakage model.* The two commonly used are *Hamming weight model* and *Hamming distance model.* In the former, the leakage is assumed to depend on the Hamming weight of a processed variable, that is, on the number of ones in the binary representation of the variable. This model holds for most of the electronic circuits since the variable with a higher Hamming weight will cause more individual gates in the circuit to switch to a high state and therefore consume more power. The Hamming distance model holds for the circuits where a gate switches to a new state without an intermediate precharge. In this situation, the number of gates that switched is proportional to the number of differing bit values in the old and the new variable. Namely, if a byte a is loaded into a register after a byte b, the Hamming distance between the two is the Hamming weight of the XOR difference $a \oplus b$. Dependencies between the leakage and the intermediate variable are not necessarily linear. Other models have been proposed that take into consideration the properties of some devices, for instance the *switching distance* model [118].

Attack flow

A typical side-channel attack consists of several stages.

1. *Profiling stage (optional).* This stage takes place if an attacker has a copy of the attacked device which can be used to characterize the leakage. The result of this stage is a *profile* (a characteristic) for the leakage of the device.

2. *Online stage.* In this stage, the attacker performs measurements (acquisitions) of a certain physical observable while the target device is processing some inputs. The result of this stage is a set of side-channel traces and the corresponding algorithm inputs or outputs. A *side-channel trace* is a vector of leakage samples observed over a period of time. In some attacks, knowledge of the inputs or outputs may not be required.

3. *Offline stage.* In this stage, the traces and the corresponding inputs or outputs are used in a certain attack algorithm to recover the key. The result of this stage is one or several key candidates.

   This stage usually includes pre-processing of side-channel traces. Typically pre-processing is required to reduce the dimensionality of the traces (i.e. reduce the number of leakage samples per trace, selecting the most informative

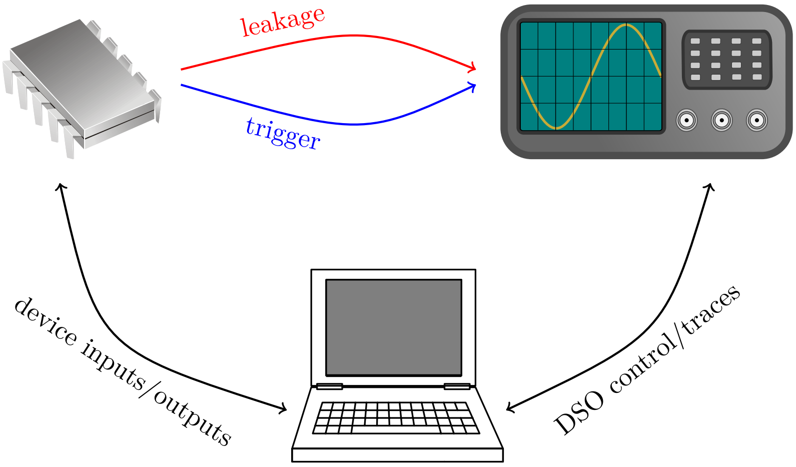


Figure 1.3: High-level scheme of a typical measurement setup for power analysis and EM attacks

ones), to align traces in time dimension, or to apply filters for de-noising and extraction of specific features.

These are the basic stages. Paper [86] suggest a more fine-grain flow.

Each stage has the corresponding complexity. The complexity of the online stage, or the online *measurement complexity*, is the number of acquisitions performed, or equivalently the number of (not necessarily) different inputs submitted to the target device during the online stage. This is an important parameter, since in many cases it is crucial to have a successful attack with as little measurements as possible.

Measurement setup

Acquisition of physical parameters such as power consumption or electromagnetic radiation is done with a measurement setup. Figure 1.3 shows a scheme of a typical measurement setup for side-channel attacks.

The setup consist of a digital sampling oscilloscope (DSO), a host PC and a target device. The host PC governs the acquisition campaign, sending inputs to the device, controlling the DSO and receiving side-channel traces from it. The DSO performs side-channel acquisitions, digitizing the analogue signal and storing it as a sequence of samples. Power consumption is typically registered as a voltage drop-off on a resistor inserted in the ground or voltage line of a target device. EM radiation is measured with an electric or a magnetic filed probe; the signal from the probe is usually weak and requires amplification before entering the DSO. Modern DSOs are PC-based and thus can carry out the functions of the host PC, simplifying the scheme.

A very important aspect of side-channel acquisitions is synchronization, since most attacks require corresponding leakage samples across different side-channel traces to be aligned in time dimension. In research settings where one has full control over the studied implementation, a dedicated trigger signal is usually used that is generated by an implementation at the start of the target algorithm execution. In real-world applications, the explicit trigger source is often not available; there exist passive [127] and active [149] solutions to achieve synchronization. However, traces would often be misaligned, so additional alignment is necessary during the pre-processing step.

Measurement setups that were used in this work varied from task to task and are described in detail in corresponding Chapters. When building the setups, we largely relied on the detailed description of a measurement setup and measurement process presented in [144].

Attack algorithms

Simple side-channel analysis The first thing the adversary can do in a side-channel attack is to explore a single side-channel trace, or several side-channel traces *individually*. This setting is called simple side-channel analysis. It was first described in [88]. In the case of a specific leakage, one speaks of simple power analysis (SPA) or simple electromagnetic analysis (SEMA).

An example of SPA and SEMA is shown in Figure 1.4 which presents power and electromagnetic traces acquired while an 8-bit AVR microcontroller was executing a modular multiplication of two long integers followed by a modular squaring. We can see that the two operations yield two different patterns both in the power and in the EM trace. Note that the pattern difference in the EM trace is stronger since EM leakage is not affected by the chip and board capacitances that act as a low-pass filter when measuring the power consumption.

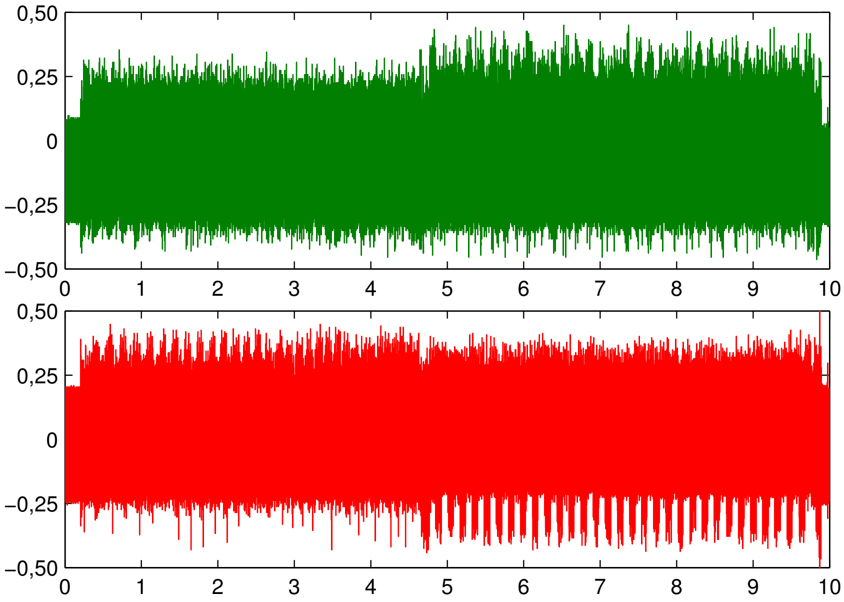


Figure 1.4: SPA (top) and SEMA (bottom) example (from [95])

This was an example of exploiting the dependency of the leakage on the operations or instructions. Consequently, if the flow of the algorithm (the sequence of instructions) depends on the secret key, one can recover the key or part of it. This is mostly the case for the public key algorithms, which are a usual target of SPA. A recent paper [48] shows that despite its simplicity, SPA can overcome certain countermeasures. In the case of secret key algorithm implementations, the instruction dependency is normally not present. Data dependency can still be used in SPA, which was shown in [99]. Simple power analysis can be also used as a preliminary step to identify the location of the entire algorithm execution and of the algorithm parts in time.

Differential side-channel analysis. Differential power analysis (DPA) technique was suggested in [88] and ahs been extensively studied since then. The name unites many specific attacks that all follow the same general scheme. Usual targets for DPA attacks are symmetric ciphers implemented using lookup tables, since the latter usually implement nonlinear functions that facilitate DPA [120].

DPA is a *divide-and-conquer* attack, namely, it independently recovers the individual chunks of the key. This is possible because in a cryptographic algorithm, small chunks of the secret key would be at some point processed independently. The side-channel leakage is used to build a distinguisher for the chunks of the key.

In a DPA attack, the adversary guesses the value for the chunk of the key and, for a set of known input (or output) values, calculates the values of the intermediate variable (called a *target variable*) that depends on this key chunk. Then he calculates the predicted leakage of the device using an appropriate leakage model. Finally, using a certain statistical test, he compares the predicted leakage with the observed leakage. From all the possible candidates for the chunk of key, those exhibiting higher correlation values are more likely to be correct.

The original DPA paper [88] used difference-of-means test as a distinguisher. Later, Pearson's correlation was suggested along with the Hamming weight leakage model in [31]. This attack is known under the name of *correlation power analysis* (CPA). These and many other suggested attack variants are *univariate*, which means they work when the leakage of *individual* samples is correlated to an intermediate variable. In the recent works [102] and [5] it was shown that most univariate attacks are equivalent, namely they can be expressed as correlation power analyses with different leakage models. An important practical property of univariate DPA attacks is that they work without requiring the a priori knowledge of the precise time moment when the intermediate variable is processed. They do not necessarily require perfect synchronization of traces, though alignment and signal reconstruction can reduce the measurement complexity [39, 100].

Univariate DPA attacks do not work when intermediate variables of an algorithm are randomized by a masking countermeasure (see Section 1.3.4). In this case, one has to look at the joint behaviour of several leakage samples within a trace. This is done in *higher-order DPA attacks* [106] by combining several leakage samples using a *combining function*, for instance, the absolute difference of leakage samples. The value of the combining function is correlated to the intermediate variables, but combination results in the loss of information. This disadvantage of higher-order DPA attacks can be avoided in *mutual information analysis* (MIA) [63] which uses a generic distinguisher that does not necessarily require a leakage model. Note that correlation-based distinguishers are still superior to mutual-information based distinguishers when the leakage is a linear function of the attacker's predictions [121], which is often the case in practice. Other generic distinguishers were suggested in [137] and [11].

Template attacks. These attacks [35] work in the profiled scenario, i.e. when the leakage of the device can be characterized in advance. During the profiling stage, the adversary obtains leakage *templates* knowing the values of the intermediate variable. During the offline stage, the acquired leakage samples for the unknown value of an intermediate variable are compared with the templates to determine the most likely candidates for the value using a statistical test.

Template attacks are optimal an the information-theoretic sense with the respect to the exploitation of side-channel leakage. They do not necessarily require a leakage model. Knowledge of inputs or outputs for the attacked algorithm is also not necessary [71].

Like DPA, template attacks are based on the divide-and-conquer principle. For the evaluation and comparison of divide-and-conquer attacks, a framework of several metrics was suggested [152, 151].

Other attacks. *Collision attacks*, initially studied in [142], use the fact that an equality of intermediate values (an *internal collision*) can be detected by comparing the side-channel leakage corresponding to these values. A specific leakage model is not required for collision detection in side-channel traces. The key is recovered from a set of equations that result from observed collisions. That is, these attack are not of the divide-and-conquer type; we call them *analytic*. In Chapter 2 we show how to combine collision and divide-and-conquer attacks to obtain a new more efficient attack.

Algebraic attacks [126, 125] are also of the analytic type. Unlike collision attacks, they require a leakage model: the key is recovered from the set of equations that originate from Hamming weights of the internal variables.

Timing attacks were initially suggested in [87]. They exploit the dependency of the execution time of an algorithm on the secret key. This dependency originates from the execution flow mainly in public key algorithms. For symmetric key algorithms, the timing leakage is usually due to the cache mechanism of a microprocessor.

Cache attacks form a special group of attacks. It would be more correct to call them *cache-collision* attacks since the cache mechanism reveals collisions in the higher order bits of the internal variables. The cache can leak information through timing, which is exploited in *cache timing attacks* [115, 15], through pattern of accesses that can be monitored with a dedicated process [109, 114], or through power and EM side channels [16]. In Chapter 3 we suggest cache-collision attack algorithms that can work with the noisy side-channel traces.

Countermeasures

Countermeasures against side-channel attacks fall into two groups [101].

The countermeasures of the first group aim at making the leakage of the device independent of intermediate variables processed by the device. This is called *hiding*. The countermeasures of the second group make intermediate variables processed by the device independent of the internal variables of the algorithm. This is called *masking*. Hiding does not change variables processed by the device but alters the leakage so that it is less correlated to the variables, whereas masking does not change leakage characteristics but removes the informative signal from the instantaneous leakage. Masking uses the approach of *secret sharing*; *provably secure* masking schemes exist [130, 131].

Both hiding and masking can be implemented on different levels of an implementation: at the circuit level, where hardware countermeasures are only possible, and at the architecture level, where both hardware and software countermeasures are possible. At the circuit level both masking and hiding are implemented by *secure logic styles*. At the architecture level, masking can be implemented both in software and in hardware. Hiding can be implemented in time and amplitude dimensions. In the time dimension, insertion of random delays through dummy instructions and shuffling the order of operations are used in software; in hardware one can additionally randomize clock signal parameters. In the amplitude dimension, signal filtering and noise generation can be done hardware, and certain programming techniques can be used in software [101]; hiding is also possible on the microarchitectural level [65]. In Chapter 6 of this thesis, we present an efficient random delay generation algorithm as a software hiding countermeasure.

Additionally, there are *protocol-level* countermeasures, which are neither hiding nor masking. They usually limit the usage of a single cryptographic key to a certain number of executions [67, 105].

Finally, *leakage-resilient cryptography* [51, 153] aims at designing cryptographic primitives that are provably secure in the model where the bounded amount of information is leaked by an implementation.

1.3.5 Fault analysis

Fault attacks usually require more expensive equipment than side-channel attacks but due to their active nature are harder to counter. They consist of two stages. First, *fault injection* is performed to disturb the normal execution of an algorithm and introduce errors into it. Second, the result of the faulty behaviour (usually along with the result of the correct behaviour for the same inputs) is exploited in an attack, for instance, for key recovery.

Physical fault injection

Injecting faults into the execution of an algorithm on an electronic device can be done at different levels of invasion.

A non-invasive way to inject faults is tampering with the power supply or the clock signal of the device [9, 138]. A short change in the power supply line is usually referred to as *spike* (even if it is a power cut-off). A modification of the clock signal is called a *glitch*. In a microprocessor, a spike usually affects data transferred over the buses; depending on the bus type, it can lead to corruption of the data, or to the change or skipping of an instruction. A glitch usually leads to skipping an instruction. Note that a glitch in the clock signal can be induced by a spike in the power supply line. In the work described in Chapter 5 of this thesis, we have implemented fault injection using spikes. Strong electromagnetic radiation [123, 139] can also be used for non-invasive fault injection.

Decapsulating a chip makes optical fault injection possible [150, 9, 139]. A flash of light or a laser beam can disturb the operation of the chip. Laser beam can be focused on a particular region of the chip, for instance on the memory or register file, so laser fault injection provides an attacker with an additional spatial control compared to light flashes and to spikes/glitches where one can vary the timing only.

An injection may result either in a *transient fault* when it disappears if no fault injection is performed, or in a *permanent fault* when a non-volatile memory cell contents is changed or the hardware is damaged in a irrecoverable way and the fault persists even if fault injection is not performed any more. An example of an attack that creates and exploits permanent faults can be found in [140].

In some attacks the knowledge of the concrete effect of the fault is not necessary. Other attacks work under a certain *fault model*, that is, under assumptions that fault injection leads to a particular kind of corruption. One example of a fault model is a flip of a single bit within an operand. Another example is the change of all the bits in a byte or a word to 0 or to 1, which is known as the *stuck-at* fault model. The choice of a realistic fault model is crucial for developing an attack, otherwise the attack will not work in practice.

Exploitation of faults

Fault attacks fall into two main categories. Attacks of the first category use the faulty output values of the algorithm. For symmetric algorithms, such attacks are known as *differential fault analysis* (DFA). They are based on the techniques of differential cryptanalysis [20] and exploit differences between faulty and correct outputs of the cipher. DFA was introduced in [21] against DES and later improved and applied to AES and several other symmetric algorithms (see recent works [128] and [80] for a comprehensive bibliography).

For public key algorithms, there is no special name for the attacks of the first category. The most powerful is probably the attack on RSA presented in [29] and known as the *Bellcore attack*. This attack applies to implementations that use Chinese remainder theorem (CRT) for efficiency reasons. It recovers the private key from a single faulty output and does not require a specific fault model. In Chapter 5 of this thesis, we show how to extend the Bellcore attack to certain RSA signature schemes where messages are only partially known to the attacker. Several kinds of fault attacks on RSA implemented without CRT exist (see recent papers

[129] and [17] for an overview). In contrary to the Bellcore attack, they work under certain fault models and require several faulty outputs. Fault attacks on other public key algorithms were also studied [81].

Attacks of the second category use not the faulty output values themselves but the only fact whether the fault injection resulted in the faulty result or not. For symmetric algorithms, an example of such attacks can be found in [38] under the name *ineffective fault analysis* (IFA). For public key algorithms, they are referred to as *safe-error attacks* and can be further subdivided into *C-safe error* [164] and *M-safe-error* [83, 163] attacks. The former exploit the presence of dummy computations that are usually used to achieve constant running time and thus thwart timing and/or SPA attacks, the latter exploit memory or register operations. Safe-error fault attacks specifically apply to a standard countermeasure that returns an error message instead of the computation result if a fault is detected.

Other fault attacks target protected implementations. For instance, *second-order fault analysis* [82] uses two fault injections in a single algorithm run. The first fault injection causes computations to go wrong and produce the faulty result, the second fault is used to skip the checking procedure.

Countermeasures

Countermeasures to fault attacks can be implemented on the hardware level and/or on the algorithmic level.

On the hardware level, countermeasures aim mainly at preventing physical fault injection. Examples are filters in power supply and clock lines, sensors for detecting variations in power supply voltage and clock frequency to protect from spikes and glitches, and tamper-resistant techniques to protect from optical and other semi-invasive and invasive fault injection techniques. There are also solutions on the hardware level to detect and correct faults.

Algorithm-level countermeasures generally aim at detecting faulty values. They are based on introducing redundancy and checks into computations, and on randomization of the operands. This can be done in a generic way, or in an algorithm-specific way. Additionally, a generic countermeasure preventing fault injection is desynchronisation [3]. The desynchronisation algorithm we suggest in this work (see Chapter 6) can be used to decrease fault injection precision.

Countermeasures against semi-invasive and invasive attacks are generally referred to as *tamper resistance* and are related to physical design of computing devices.

1.4 Challenges and research directions

There are two main directions of research in implementation security. First, we should know what to protect from, so implementation attacks should be studied. A designer should find a weakness and fix it earlier than the malicious attacker who will exploit it. Therefore, one has to improve already existing attacks, find new attacks and identify sources of physical leakages and vulnerabilities, analyse physical security of existing devices.

Second, one has to develop and improve means of countering implementation attacks. Countermeasures must be secure, that is provide the good level of protection, and efficient, that is, cause a minimal possible performance overhead.

Other directions of research include finding the ways of estimating attack complexity to be able to compare between the attacks, designing provably secure countermeasures, and generalizing attacks.

1.5 Contributions

In response to the challenges mentioned above, in this thesis we elaborate on side-channel attacks, fault attacks and efficient countermeasures. The primary targets are software implementations of cryptographic algorithms on various embedded devices. The contributions are arranged into Chapters as follows.

- In Chapter 2 we present a novel technique of combining side-channel collision attacks with divide-and-conquer side-channel attacks such as differential power analysis. The combination of different side-channel techniques allows us to use more key-relevant information contained in the side-channel traces than by stand-alone application of the techniques. We theoretically compute the success probability and the expected computational complexity of combined collision attacks. We also suggest a new technique for dimension reduction of side-channel traces to improve collision detection using Euclidean distance, and an information-theoretic metric for the comparison of different collision detection methods. All these solutions lead to the reduction of the measurement complexity compared to the stand-alone attacks. We practically demonstrate that DPA-combined collision attacks are more efficient than both conventional collision attacks and DPA. The results of the Chapter are published in [28] (partially) and [27].

- In Chapter 3 we develop attacks exploiting the side-channel leakage of the microprocessor's cache mechanism that are tolerant to noisy measurement conditions. These attacks target symmetric encryption algorithms such as AES implemented in software using lookup tables. We suggest key recovery algorithms for AES that are tolerant to uncertainties in distinguishing cache misses from cache hits when observing power consumption or electromagnetic radiation of an embedded microprocessor. Previous works were assuming perfect cache event detection. Our algorithms also work in the setting when the cache is not clean of the lookup table lines prior to the execution. We describe the theoretical model of the attack. We investigate in practice the leakage of a cache on a 32-bit ARM microcontroller. The Chapter is based on the papers [60] and [59].

- Chapter 4 presents a practical side-channel attack against the cryptographic engine of AVR XMEGA, a recent versatile 8-bit microcontroller with a rich set of features for embedded applications. We discover that the AES engine of AVR XMEGA succumbs to an efficient differential power analysis attack. Our practical implementation of the attack requires about 3000 power consumption measurements to recover the full AES-128 secret key stored in the microcontroller. The attack was developed without any particular knowledge about the AES implementation. This result was published in [85].

- In Chapter 5 we propose new fault attacks against the randomized ISO/IEC 9796-2 padding scheme used for digital signature generation with the RSA algorithm. The existing fault attacks on RSA with the Chinese remainder theorem can factor an RSA modulus (and therefore recover the signer's private key) only when a deterministic padding scheme is used; they do not work when messages contain some randomness which is recovered only when verifying a correct signature. Our new attacks apply to a large class of partially known message configurations. They rely on Coppersmith's algorithm for finding small roots of multivariate polynomial equations. We illustrate the approach by successfully attacking several randomized versions of the ISO 9796-2 encoding standard and carry out practical fault injection experiments on an 8-bit microcontroller. The Chapter is based on the paper [43].
- In Chapter 6 we suggest a new algorithm for random delay generation in embedded software as a countermeasure against side-channel and fault attacks. Random delays can be inserted into the execution flow of a cryptographic algorithm to break synchronization in a physical attack. The idea behind our new algorithm is to generate individual random delays non-independently, which is different from the previously existing methods. This solution allows to introduce more uncertainty for the attacker with less performance overhead. We analyse the properties of the algorithm and show how to choose the correct parameters for its implementation. We also suggest an information-theoretically sound criterion for estimating the efficiency of the random delay countermeasure. We provide an efficient reference implementation for an 8-bit microcontroller. Practical side-channel attacks that we mount against an embedded software AES implementation protected with random delays show that the new algorithm is more efficient and secure than the previously existing methods. The Chapter is based on the papers [44] and [45].

Chapter 2

Combined side-channel collision attacks

A fundamental problem of extracting the highest possible amount of key-related information using the lowest possible number of measurements is central to side-channel attacks against embedded implementations of cryptographic algorithms. To address it, in this Chapter we propose a novel framework enhancing side-channel collision attacks with divide-and-conquer attacks such as differential power analysis (DPA). An information-theoretical metric is introduced for the evaluation of collision detection efficiency. Improved methods of dimension reduction for side-channel traces are developed based on a statistical model of Euclidean distance.

Experimental results of this work confirm that DPA-combined collision attacks are superior to both DPA-only and collision-only attacks. The new methods of dimension reduction lead to further complexity improvements. All attacks are treated for the case of AES-128 and are practically validated on a wide-spread 8-bit RISC microcontroller.

This is a joint work with Andrey Bogdanov, published in [28] (partially) and [27].

Contents

2.1 Introduction

2.1.1 Motivation

Keyed cryptographic algorithms employ secret information to protect user data and can provide its confidentiality, integrity, authenticity, non-repudiation—services crucial for almost any security-related application. Numerous analysis methods have been proposed for cryptographic algorithms. While the traditional mathematical attacks are solely based on the inputs and outputs of an algorithm, side-channel attacks rely upon the fact that any real-world implementation of the algorithm is not ideal and leaks some physically observable parameters that are dependent on the key processed. Such parameters can include time [87], power consumption [88], electro-magnetic radiation [122] and algorithm behaviour under actively induced execution faults. Since the attacker often has immediate physical access to embedded systems, they are most vulnerable to side-channel attacks. A fundamental problem of side-channel analysis is as follows:

Problem 1 (Fundamental for side-channel analysis)**.** *Extract the highest possible amount of key information given the lowest possible amount of side-channel information for a fixed implementation of a cryptographic algorithm.*

Side-channel collision attacks provide a natural basis for solving this problem, possessing the unique combination of three important properties which are not simultaneously present in any other side-channel analysis technique known today:

First, they are essentially based on the *algorithmic properties* of the attacked cryptographic algorithm, which allows the adversary to use more side-channel information from one algorithm execution. Second, they are *not based on any particular leakage model* which opens up the possibility of using relevant side-channel information not limited to a specific model. Third, they do *not require any significant a priori knowledge* of the implementation (a major limitation in many side-channel attacks), however, being able to profit from profiling. Side-channel collision attacks have also further attractive features such as that essential parts of the cryptographic algorithm can remain unknown to the attacker which makes many algorithmic masking techniques transparent to collision attacks.

In this work, we come up with two novel techniques significantly enhancing side-channel collision-based analysis and propose a general framework naturally incorporating them.

2.1.2 Collision attacks in the context

In this Subsection, we aim to draw attention to some of the beneficial features of collision attacks mentioned above that they exhibit in the context of other approaches to side-channel analysis.

Regarding the method of *extracting key-related information*, there are two large classes of side-channel attacks: *leakage-model* oriented and *pattern-matching* oriented. With respect to the *key-recovery procedure*, side-channel attacks fall into two categories: *divide-and-conquer* attacks (which provide distinguishers for small key chunks) and *analytic* attacks (which recover the entire key e.g. by solving systems of equations). Correspondingly, when classifying according to information extraction method and key-recovery procedure, one can speak about the four types of side-channel attacks represented in Table 2.1. Note that, optionally, side-channel attacks can use profiling, which we consider here without introducing a separate dimension in the table.

Differential power analysis (DPA) [88] and correlation power analysis (CPA) [31], a generalization of DPA, are probably the most wide-spread practical attacks on numerous embedded systems such as smart-card microcontrollers and dedicated ASICs. They are based on guessing a chunk of the key, classifying traces according to this hypothesis and performing a statistical test in a leakage model such as Hamming weight or Hamming distance. Statistical tests in DPA attacks are essentially correlation-based [102] and thus can capture only linear dependencies between the intermediate variables and the leakage. Similarly to DPA, mutual information analysis (MIA) [63, 121, 12] is based on subkey guessing and classifying traces. However, the test performed for each key hypothesis uses an information-theoretic metric which works with more generic leakage models.

Template attacks [35, 6] belong to another class of powerful side-channel attacks and are optimal in an information-theoretic sense. They do not rely on any particular leakage model but require a profiling stage and, as DPA, are mainly limited to key chunks. Stochastic methods [137, 64] can be seen as a version of template attacks allowing one to simplify template building, further increase the resolution and, thus, decrease the total number of measurements needed.

Algebraic side-channel attacks [126] use Hamming weights of intermediate variables detected by observing side-channel traces to simplify the systems of nonlinear equations on the full key. Thus, algebraic side-channel attacks imply that the implementation leaks Hamming-weight related side-channel information.

Side-channel collision attacks [24, 25, 28, 142, 141] use pattern-matching techniques (like template attacks) being however essentially based on the cryptanalytic properties of attacked cryptographic algorithms (by attacking key as a whole as in algebraic side-channel attacks) and not relying on any complex profiling stages (similarly to DPA).

Table 2.1: Side-channel attacks: methods of extracting key-related information and key-recovery procedure

	Leakage model	Pattern-matching
Divide-and-conquer	DPA[88] CPA [31] MIA [63]	template [35] stochastic [137]
Analytic	algebraic [126]	collision [24, 141]

Side-channel attacks with analytic key-recovery tend to be more efficient in terms of measurement complexity. Side-channel attacks using pattern-matching information extraction are independent of a concrete leakage model (such as Hamming weight or Hamming distance), thus, being a way more universal. Collision attacks share both these benefits.

At the same time, collision attacks have certain important limitations that in some cases might make them inapplicable in practice. Namely, they require knowledge of time moments when the leakage of a certain event occurs (e.g. S-box executions), they are more efficient on an 8-bit architecture than on architectures with wider buses, finally, they are highly sensitive to type-II errors in collision detection.

Recently, some side-channel techniques using more than one method of extracting key-related information have been proposed. For instance, differential cluster analysis [11] is a divide-and-conquer attack that generally uses pattern-matching but can benefit from the knowledge of leakage model. However, these attacks do not use the advantages of analytic key recovery.

In this work, we show that the analytic key-recovery procedures of collision attacks allow for extensions and propose a general framework for incorporating key-related information resulting from divide-and-conquer attacks such as DPA and template attacks into collision techniques.

2.1.3 Our contributions

In this work, we introduce the *combined collision attack* which is a novel technique for combining side-channel collision attacks with divide-and-conquer attacks such as DPA and template attacks, thus, using *both* divide-and-conquer and analytic key recovery as well as *both* leakage models and pattern-matching extraction (Section 2.3). This combination of very different side-channel techniques allows us to use more key-relevant information contained in the side-channel traces, omitted by each of these techniques when applied separately. We theoretically compute the success probability and expected computational complexity of combined collision attacks (Section 2.4).

Starting from the basic Euclidean distance, we propose new techniques of efficient dimension reduction for collision detection. We study some of their statistical properties in a formal way. We propose the usage of λ-*divergence* as a metric for the comparison of different collision detection methods (Section 2.5).

We practically demonstrate that DPA-combined collision attacks are more efficient than both conventional collision attacks and DPA (Section 2.6). On the theoretical side, this fact naturally implies that a DPA-combined collision attack uses more key information contained in the traces than both a stand-alone conventional collision attacks and a stand-alone DPA attack. On the practical side, the new findings allow us to further reduce the measurement complexity of side-channel collision attacks. We conclude with a discussion and open problems in Section 2.7.

2.2 Preliminaries

2.2.1 Internal collisions

An *internal collision* in a cryptographic algorithm $\mathcal{A}$ occurs with respect to a *target function* ϕ, if ϕ delivers the same output y given some two inputs x_1 and x_2: $y = \phi(x_1) = \phi(x_2)$ that are not necessarily equal.

Generally speaking, if $\mathcal{A}$ is an iterative cipher and ϕ is applied in its first iterations, it can be very difficult for the attacker to say if $\phi(x_1) = \phi(x_2)$ using black-box queries (plaintext-ciphertext pairs) only. However, side-channel leakage can help him to detect internal collisions. Assume that some function ψ processes the output of ϕ. If ϕ returns an equal value y for two inputs, $\mathcal{A}$ performs two identical calculations $\psi(y)$ in these two cases. If ϕ returns two unequal values y_1 and y_2, the corresponding calculations $\psi(y_1)$ and $\psi(y_2)$ are distinct. The attacker can observe this behaviour in the power consumption or electromagnetic radiation of the device implementing $\mathcal{A}$ during the application of ψ—similar side-channel traces for equal outputs of ϕ and diverse side-channel traces for unequal outputs of ϕ. Because of this, ψ is called a *collision detection function.*

Once an internal collision is detected, it can be interpreted as a key-dependent equation $\phi(x_1) = \phi(x_2)$ delivering some information about the key, if ϕ and/or x_1 as well as x_2 are key-dependent.

However, if there are several applications of ϕ within the algorithm $\mathcal{A}$ and ϕ is invertible, one does not need a separate collision detection function ψ and can employ ϕ as both a target function and a collision detection function. Moreover, a collision now leads to a much simpler algebraic equation $x_1 = x_2$. The latter observation has yielded the idea behind *generalized collisions* proposed in [24] and provides a major advantage over other collision-based attacks such as [141].

2.2.2 Previous work on collision attacks

The principle of using internal collisions in cryptanalysis is due to Hans Dobbertin and was also discussed in the early work [160]. In [142], a collision attack on DES was proposed, several adjacent DES S-boxes representing the target function ϕ. This attack was enhanced in [93] using the notion of *almost collisions* which are internal states of an algorithm that are very similar. In [141], the separate bytes of each of the four 4-byte linear MixColumn mappings in the first AES round are treated as target functions ϕ, S-boxes of the second round representing the collision detection function ψ. In [22], it is shown that similar side-channel collision attacks can be applied to AES-based MACs such as Alpha-MAC to mount selective forgery attacks that do not require any knowledge of the secret key. The results in [23] suggest that collision attacks can help overpass the random masking of some AES implementations. Overpassing random masking with collision attacks for the case of DES was done in [70] and improved in [84], but these works consider collisions in Hamming weights and therefore imply the leakage model. In [107] collision attacks were applied to a masked S-box implementation, exploiting the remaining minor leakage. Another work [117] also employed pattern matching in a step of the attack against a masked and shuffled S-box implementation to remove masking after recovering the masked subkeys and overcoming shuffling with DPA.

As mentioned above, side-channel collision attacks on AES were improved in [24] by introducing the notion of generalized collisions that occur if two S-boxes at some arbitrary positions of some arbitrary rounds process an equal byte value within several runs. Here both the target and collision detection functions ϕ and ψ coincide being the 8-bit AES S-box. The S-box remains the same for all executions, rounds and byte positions within the round (as opposed to DES). This increases the number of function instances to compare, i.e. the number of potential collisions to be used afterwards for key recovery.

While [24] treats the *linear collisions* (resulting in linear equations on the key) of AES which are generalized collisions that occur in the first AES round only, the work [28] also considers *nonlinear collisions* (respectively, resulting in nonlinear equations). A set of such collisions can be considered as a system of equations over a finite field. Ways to deal with unreliable collision detection are discussed in [25], including the techniques of binary and ternary voting.

2.2.3 Linear collision-based key recovery for AES

To simplify representation, we chose to study collision attacks at the notable example of the U.S. encryption standard AES [171]. More precisely, we use the key-recovery [24] for AES-128 based on linear collisions for this purpose. Note that all techniques of this work can be successfully applied to other ciphers as well as to other collision-based key-recovery techniques.

We use the following notation to represent the variables of AES. $K = \{k_j\}_{j=1}^{16}$, $k_j \in \mathbb{F}_{2^8}$ is the 16-byte user-supplied key (the initial AES subkey). AES plaintexts are denoted by $P^i = \{p_j^i\}_{j=1}^{16}$, $p_j^i \in \mathbb{F}_{2^8}$, where $i = 1, 2, \ldots$ is the number of an AES execution.

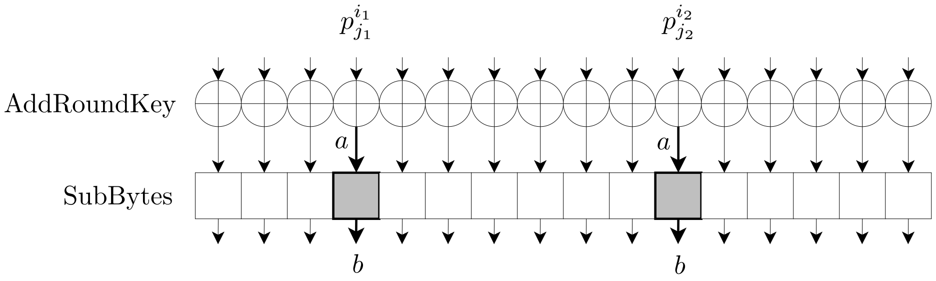


Figure 2.1: A linear collision for a pair of AES executions

Given a collision within the first round of AES (linear collision)

$$S(p_{j_1}^{i_1} \oplus k_{j_1}) = S(p_{j_2}^{i_2} \oplus k_{j_2}), \tag{2.1}$$

one obtains a linear equation with respect to the key over $\mathbb{F}_{2^8}$ of the form

$$k_{j_1} \oplus k_{j_2} = p_{j_1}^{i_1} \oplus p_{j_2}^{i_2} = \Delta_{j_1,j_2} \text{ for } j_1 \neq j_2. \tag{2.2}$$

If D collisions have been detected, they can be interpreted as a system of linear equations over $\mathbb{F}_{2^8}$:

$$\begin{cases} k_{j_1} \oplus k_{j_2} & = & \Delta_{j_1,j_2} \\ & \ldots & \\ k_{j_{2D-1}} \oplus k_{j_{2D}} & = & \Delta_{j_{2D-1},j_{2D}} \end{cases} \tag{2.3}$$

This system cannot have the full rank due to the binomial form of its equations. Moreover, for small numbers of inputs to AES the system is not connected and it can be divided into a set of h_0 smaller independent (with disjunct variables) connected subsystems with respect to the parts of the key. Each subsystem has one free variable. Let h_1 be the number of all missing variables, and $h = h_0 + h_1$. We call each of these h subsystems or missing variables a *chain*.

Without loss of generality, a chain ζ of length n can be represented as the following subsystem of the equation system (2.3):

$$\begin{cases} k_{j_1} \oplus k_{j_2} & = & \Delta_{j_1,j_2} \\ k_{j_2} \oplus k_{j_3} & = & \Delta_{j_2,j_3} \\ & \ldots & \\ k_{j_{n-2}} \oplus k_{j_{n-1}} & = & \Delta_{j_{n-2},j_{n-1}} \\ k_{j_{n-1}} \oplus k_{j_n} & = & \Delta_{j_{n-1},j_n}, \end{cases} \tag{2.4}$$

or alternatively as an n-tuple of byte indices $\zeta = (j_1, \ldots, j_n)$ in a short form. Each chain (2.4) has 2^8 possible solutions, since it is sufficient to guess one key byte in the chain to unambiguously determine all other $n-1$ bytes of the chain. If the system (2.3) has h chains, then it has 2^{8h} solutions.

That is, 2^{8h} guesses have to be performed, which is the offline computational complexity of this basic key-recovery method. Each key hypothesis is then tested using a known plaintext-ciphertext pair with the full AES to rule out wrong candidates. Note that 2^{8h} quickly becomes feasible as the number of distinct inputs P^i grows. In a nutshell, it has been shown in [24] that 6 randomly drawn plaintexts

P^i generate enough byte collisions in AES-128 to cover the full 128-bit key by just $h = 5$ or less chains. This drives the offline computational complexity down to at most 2^{40} AES runs.

2.2.4 Collision detection with Euclidean distance

For a collision attack to be successful, one has to decide if two S-boxes accept equal inputs using side-channel information obtained from the side-channel leakage (of the implementation) of the attacked cryptographic algorithm. Given two side-channel traces

$$\tau_1 = (\tau_{1,1}, \ldots, \tau_{1,l}) \in \mathbb{R}^l \text{ and } \tau_2 = (\tau_{2,1}, \ldots, \tau_{2,l}) \in \mathbb{R}^l,$$

respectively corresponding to a pair of S-box executions with some unknown inputs a_1 and a_2, it has to be decided whether $a_1 = a_2$ for collision detection.

The two traces can be considered as two vectors in the Euclidean space of dimension l. The Euclidean distance E between them is defined as

$$E(\tau_1, \tau_2) = \sum_{r=1}^{l} (\tau_{1,r} - \tau_{2,r})^2.$$

One expects that E will be higher for non-collisions and lower for collisions. Our experiments with a popular AVR microcontroller (μC) (see Figure 2.2; the implementation will be described in detail in Section 2.6.1) show that this intuition is indeed justified, at least when noise is somewhat reduced by averaging traces. Note that regular patterns observed in the non-collisions in Figure 2.2 are presumably due to a special implementation of the fifth bit of one of the buses in the 8-bit AVR core.

Most papers on collision attacks [141, 24, 25, 28] use the Euclidean distance between two noisy traces as the basic metric for collision detection. In [107] and [117], Pearson's correlation was employed. We have found that collision detection can be significantly improved by dimension reduction for side-channel traces based on the properties of the Euclidean distance, which is therefore the metric of our choice. We detail this in Section 2.5.

2.2.5 Differential side-channel analysis

The technique of differential side-channel analysis was suggested in [88] as DPA and has been extensively studied since then. Now the name unites several specific attacks that follow the same general scheme. As in the case of collision attacks, here we describe differential side-channel analysis at the example of AES-128.

As already mentioned, differential side-channel analysis is a divide-and-conquer attack. Namely, it recovers each of the AES key bytes $k_1, \ldots, k_{16}$ independently. For each byte k_j, the attack takes a set of known plaintexts P^i and a set of corresponding side-channel traces. The attack algorithms differ between specific attacks, but are all essentially based on comparing the hypothetical leakage of an intermediate key-dependent *target variable* (usually the first round S-box output) predicted for each of the 256 key byte candidates with the actually observed leakage (represented by

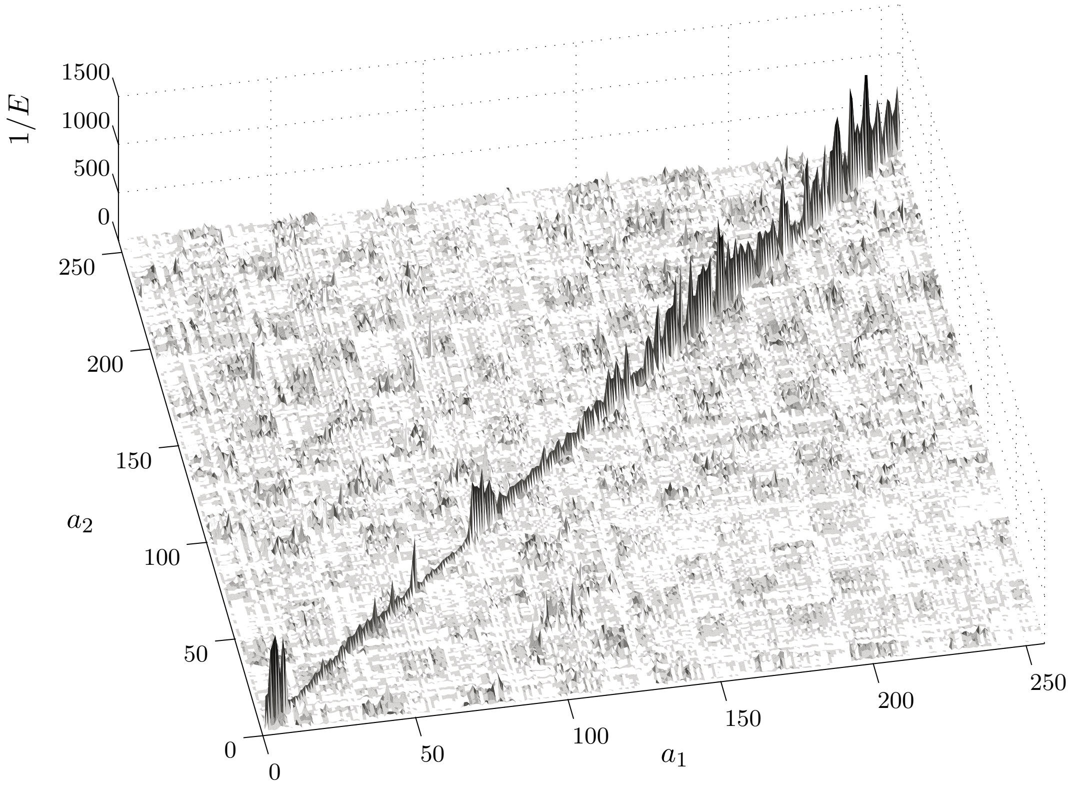


Figure 2.2: Inverse Euclidean distance $1/E$ between power consumption traces for all input pairs of the AES S-box as implemented on 8-bit RISC μC ATMega16

the side-channel traces) by means of a specific statistical test, e.g. correlation [31]. Higher values of the statistic suggest that the candidate is more likely to be correct. In the context of this work, it suffices to represent the output of the differential side-channel attack for a byte k_j of the key as the list $\mathcal{K}_j = (\kappa_j^1, \ldots, \kappa_j^{256})$ of 256 key byte candidates sorted by the output of the statistical test in the descending order. Then the correct key byte candidate is more likely to be closer to the top of the list.

In the simplest case, the attack is limited to considering only the topmost candidate κ_j^1 for each key byte, such that there is a single candidate for the full key. In general, if one considers m topmost candidates for each key byte from the corresponding list, there will be m^{16} full key candidates. An offline exhaustive search among these m^{16} candidates is then necessary to determine the correct one with a plaintext-ciphertext pair at hand. For example, if $m = 5$ candidates for each key byte are considered, the offline computational complexity is about 2^{37} AES runs.

2.3 Framework for combined attacks

In this Section, we propose a general framework for the side-channel analysis of cryptographic algorithms based upon internal collisions. This framework allows the adversary to amplify collision attacks by any divide-and-conquer side-channel attack. We refer to such amplified attacks as *combined collision attacks*.

Later, we will study the core test of the framework presented in this Section (Section 2.3.4) and evaluate it, combined with our novel collision detection techniques (Section 2.5), also in a practical setting (Section 2.6).

2.3.1 The idea in a nutshell

In the combined collision attack, we use two building blocks: the linear collision-based key recovery of Subsection 2.2.3 and a divide-and-conquer attack, such as DPA of Subsection 2.2.5. The major idea is to test chains of key bytes in collision attacks using key byte ranking from DPA. This reduces the required offline computational complexity of collision-based key recovery.

For instance, if just $h = 5$ chains cover the entire 16-byte key of AES-128, the pure collision attack has to perform $2^{8h} = 2^{40}$ full key tests (see Subsection 2.2.3), since all key byte candidates are equiprobable. However, DPA outputs a ranking of candidates for each key byte, $\mathcal{K}_j = (\kappa_j^1, \ldots, \kappa_j^{256})$ as shown in Subsection 2.2.5. So if this DPA information is available in the collision attack, the adversary can test only the most probable candidates of key byte chains. If the number of surviving chain candidates is reduced from 2^8 to 2^4 for each chain, the offline computational complexity will be just $2^{4h} = 2^{20}$ AES runs. Equivalently, this reduces the online measurement complexity by keeping the budget of offline computational complexity at the level of 2^{40}. We use this fact to address Problem 1.

2.3.2 Attack flow

All collision techniques of this work are studied at the example of the U.S. encryption standard AES, which is a highly relevant target, and experimentally verified on a wide-spread 8-bit platform. For the sake of simplicity, we will deal with AES-128 in this work, though having in mind that the collision techniques are actually much more generally applicable.

Let the AES-128 implementation have a 16-byte key fixed for the entire attack and leak a key-dependent side-channel parameter (e.g. power consumption or electromagnetic radiation).

A collision attack consists of an online stage, signal processing stage, and key-recovery stage. Its procedure is outlined in Algorithm 1 and is explained here:

- In the *online stage* (steps 1-2 of Algorithm 1), N chosen 16-byte plaintexts P^i are sent to the attacked device implementing AES (**ChooseInputs**). The side-channel traces T^i (e.g. power consumption or electromagnetic radiation) are acquired by the measurement equipment (**AcquireTraces**) for these plaintexts. Each trace T^i contains 16 subtraces, one for each S-box:

  $$T^i = \{\tau_j^i\}_{j=1}^{16}.$$

  That is, T_i is a set of 16 individual traces τ_j^i for each of the 16 S-box instances in the first AES round. The trace τ_j^i is a real-valued vector of length l, $\tau_j^i \in \mathbb{R}^l$, thus, containing l samples.

  In our attacks, we will send γ randomly drawn 16-byte plaintexts to the AES encryption, each repeated t times, which yields $N = \gamma \cdot t$.

- In the *signal processing stage* (steps 3-6 of Algorithm 1), collisions are detected in the target traces T^i (**DetectCollisions**) and the divide-and-conquer attack is applied to sort the key-byte candidates in each of the 16 byte positions (**SortKeyByte**). Before applying the signal processing, the traces corresponding to each of γ unique plaintexts are averaged t times to decrease noise. The output of the signal processing stage is the set of detected collisions $\mathcal{C}$ containing 4-tuples $(p_{j_1}^{i_1}, p_{j_2}^{i_2}, j_1, j_2)$ and 16 sorted lists $\mathcal{K}$ of 256 key byte candidates for each of the 16 byte positions. Depending on the measurement setup and implementation, one might choose to perform trace decimation and denoising in this stage.

- In the *key-recovery stage* (step 7 of Algorithm 1), an AES key candidate K' is computed using the list of detected collisions $\mathcal{C}$ and sorted candidates $\mathcal{K}$ (**RecoverKey** detailed in Algorithm 2). Note that **RecoverKey** can return either the right 16-byte key, a wrong 16-byte key or an empty set of keys $\emptyset$, if no key candidate has passed the final key testing. By π we denote the success probability of Algorithm 1 which is the probability that **RecoverKey** returns the right key.

Algorithm 1: Collision attack based on linear collisions combined with a divide-and-conquer test for AES-128

```
$\mathcal{P} = (P^1, \ldots, P^N) \leftarrow$ **ChooseInputs()**
$\mathcal{T} = (T^1, \ldots, T^N) \leftarrow$ **AcquireTraces**$(\mathcal{P})$
{each trace $T^i$ contains 16 subtraces, one for each S-box}
$\mathcal{C} \leftarrow$ **DetectCollisions**$(\mathcal{P}, \mathcal{T})$
{each collision in $\mathcal{C}$ is a four-tuple $(p_{j_1}^{i_1}, p_{j_2}^{i_2}, j_1, j_2)$, see (2.1)}
for each $k_j$ of 16 key bytes do
    $\mathcal{K}_j = (\kappa_j^1, \ldots, \kappa_j^{256}) \leftarrow$ **SortKeyByte**$(j, \mathcal{P}, \mathcal{T})$
end for
{now $\mathcal{K} = (\mathcal{K}_1, \ldots, \mathcal{K}_{16})$ contains 16 sorted lists of key byte candidates of length 256 each}
$K' \leftarrow$ **RecoverKey**$(\mathcal{C}, \mathcal{K})$
return $K'$ as a key candidate
```

Algorithm 2: Key-recovery **RecoverKey** based on linear collisions and sorted key-byte candidates from a divide-and-conquer test for AES-128

```
Input: Collisions $\mathcal{C}$ and sorted key-byte candidates $\mathcal{K}$
build $h$ chains $\zeta_1, \ldots, \zeta_h$ from collisions $\mathcal{C}$
{a chain $\zeta_i$ of length $n_i$ is an $n$-tuple $(j_1, \ldots, j_{n_i})$, see (2.4)}
for each $\zeta_i$ of $h$ chains do
    $\mathcal{G}_i \leftarrow \{0, \ldots, 2^8 - 1\}$, i.e. all $2^8$ chain guesses
    for each chain guess $g \in \{0, \ldots, 2^8 - 1\}$ do
        if not **TestChain**$(\zeta_i, g, \mathcal{K}, \mathcal{C})$ then
            remove chain guess $g$, $\mathcal{G}_i \leftarrow \mathcal{G}_i \backslash \{g\}$
        end if
    end for
    {now $\mathcal{G}_i$ only contains survived guesses for chain $\zeta_i$}
end for
unite chain guesses to full key guesses, $\mathcal{G} \leftarrow \cup_{i=1}^{h} \mathcal{G}_i$
{$\mathcal{G}$ contains full key guesses that survived chain filtration}
$K' \leftarrow$ **TestKeysWithAES**$(\mathcal{G})$
return $K'$ as a key candidate
```

2.3.3 Combined key recovery

The procedure of the combined key recovery is provided in Algorithm 2 and mainly relies on the test of each chain **TestChain** introduced and analyzed in Subsection 2.3.4. This is the major advantage of our approach compared to the conventional collision attacks where it is not possible to test the correctness of each chain separately and steps 4-8 of Algorithm 2 are missing. As opposed to that, the availability of divide-and-conquer information in the combined key recovery allows to test for each chain separately which can provide a significant efficiency gain. This can be reflected in the increased success probability of the attack given some measurement complexity or in the reduced measurement complexity given some success probability, thus, delivering a better solution to Problem 1. We will theoretically compute the success probability for a combined attack in Section 2.4 and practically evaluate it in Section 2.6.

2.3.4 Test of chain

TestChain has as input:

- Chain ζ of length n consisting of key byte indices $j_1, \ldots, j_n$.
- Guess g of the chain. Without loss of generality, we assume that it is the first byte j_1 of the chain and $k_{j_1} = g$.
- The list of (linear) collisions $\mathcal{C}$ to be able to compute all the other $n-1$ key bytes in the chain from k_{j_1}.
- The lists of sorted key-byte candidates $\mathcal{K}$ coming from a divide-and-conquer test (e.g. DPA). Only lists $\mathcal{K}_\ell$ with $\ell \in \{j_1, \ldots, j_n\}$ are needed for the chain to be tested. Each $\mathcal{K}_\ell$ is a sorted list of 256 candidates for the key byte k_ℓ.

The output of **TestChain** is *true*, if the chain ζ_i has passed the test, or *false* otherwise.

The idea of the test of chain is to filter out those guesses of chains that are less probable to be compatible with the key information obtained from a divide-and-conquer test.

In each list $\mathcal{K}_\ell$, we will consider values among the top m positions. These are the most probable candidates for the key byte k_ℓ as suggested by the divide-and-conquer test. We superpose the guess of the chain, computed from the byte guess g for k_{j_1}, and n corresponding sorted lists $\mathcal{K}_\ell$ of length 256 bytes each, see Figure 2.3.

Now the test of chain can be described as follows:

- The guess of chain is accepted if all key bytes of the chain are among the top m candidates, each in the corresponding list $\mathcal{K}_\ell$.
- The guess of chain is rejected if at least one key byte of the chain falls outside the m top candidates in its corresponding list $\mathcal{K}_\ell$.

The threshold m of the test of chain has an impact on the attack complexity and success probability. We detail on this in the next Section.

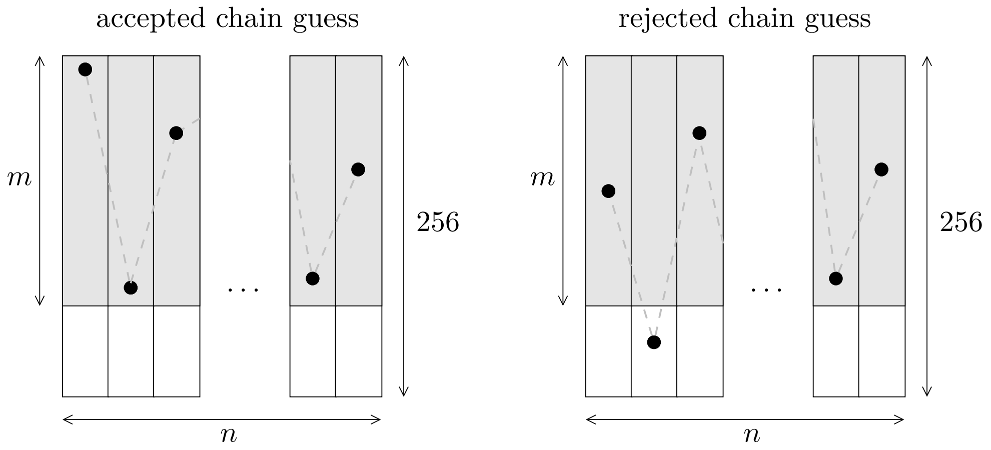


Figure 2.3: Test for chain ζ: the guess of chain is rejected if at least one key byte falls outside the most probable m byte values as suggested by the divide-and-conquer test

2.4 Attack complexity and success probability

According to the three stages of a collision attack outlined in Subsection 2.3.2, its complexity is defined by three parameters (see Algorithm 1):

- C_{online} is the number of inputs to AES for which measurements have to be performed in the online phase (**AcquireTraces**).
- $C_{\text{processing}}$ is the computational complexity of signal processing on side-channel traces needed to detect collisions (**DetectCollisions**) and sort key-byte candidates within the divide-and-conquer attack (**SortKeyByte**).
- C_{recovery} is the computational complexity of **RecoverKey** (Algorithm 2), that is, the number of operations needed to solve the resulting systems of linear or nonlinear equations and to identify the most probable solution.

For collision attacks in this work, we bound C_{recovery} by 2^{40} computations of AES which can be performed within several hours on a PC. Given this restriction on C_{recovery}, the online complexity $C_{\text{online}} = N = t \cdot \gamma$ becomes the major limiting factor of collision attacks, since $C_{\text{processing}}$, mainly determined by γ, will be negligible for our choices of γ.

The success of an attack is often of probabilistic nature and the success probability π of the attack has to be considered along with C_{online}. In this work, our main goal is to reduce C_{online} and increase π, given the above admissible upper bound on C_{recovery}.

2.4.1 Success probability of a combined attack

The threshold m of the test of chain described in Subsection 2.3.4 has to be chosen in a way that the probability to filter out the correct key is small. This probability depends on the distribution of the position for the correct key byte of the divide-and-conquer test used.

Let α be the probability that the correct key byte is among the top m candidates in the sorted divide-and-conquer list. (In terms of the unified framework [152], α is exactly the m-order success rate for a single key byte recovery.) Under the assumption that all chain tests are independent, the probability for the full correct key to survive after passing the tests with all h chains can be computed as

$$\Pr[\text{correct key survives after } h \text{ chains}] = \prod_{i=1}^{h} \alpha^{n_i} = \alpha^{\sum_{i=1}^{h} n_i} = \alpha^{16} ,$$

since the sum of all chain lengths is $\sum_{i=1}^{h} n_i = 16$. Moreover, if all collisions have been detected correctly (i.e. collision detection yielded no false positives), this determines the success probability of the full combined collision attack

$$\pi = \alpha^{16}, \tag{2.5}$$

which is in fact equivalent to the success probability of the divide-and-conquer attack.

As a practical example, our experiments with DPA attacks against an AES implementation on the 8-bit ATmega16 μC show that the chances for the correct key byte guess to be among the top m candidates in the sorted DPA list are quite good already for small values of m (which are preferred to have low C_{recovery}, as we will detail in the next Subsection) and small values of N, i.e. low C_{online}. Figure 2.4 depicts the dependency of α upon m for different numbers of inputs N in our DPA attack.

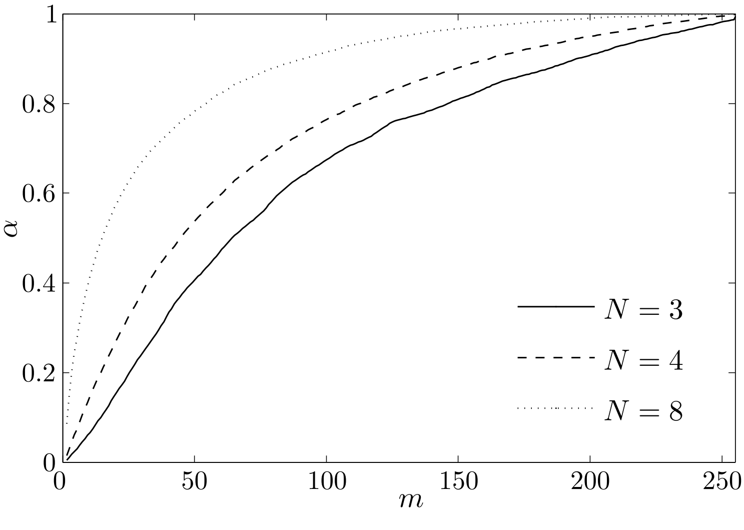


Figure 2.4: Empirical dependency of α upon m for different numbers of traces N in a DPA attack

2.4.2 Complexity of key recovery for a combined attack

Without the test of chain, the complexity of the collision attack is 2^{8h} AES computations, since each chain suggests 2^8 candidates for the respective subset of key bytes and there are h disjunct chains. In the combined approach, we effectively test m candidates for each chain. Moreover, we filter our improbable chain candidates separately for each chain. This results in a lower number of full 16-byte key candidates to be tested with AES at the end which determines C_{recovery}. Since the chain evaluation is much less complex than the testing of a full 16-byte candidate with the full AES, we can win in the total attack complexity significantly. Here we estimate C_{recovery} given m and α.

The expected number of wrong chain guesses to be tested in the test of chain can be computed as

$$(1-\alpha)m + \alpha(m-1) = m - \alpha \, .$$

The probability for a wrong chain guess to survive one element of chain can be derived as

$$\alpha\left(\frac{m-1}{255}\right) + (1-\alpha)\left(\frac{m}{255}\right) = \frac{m-\alpha}{255} \, ,$$

since a wrong chain guess results in wrong key byte candidates suggested along the entire chain.

The expected number of correct chain guesses to be tested in the test of chain is α. The probability for a correct chain guess to survive one element of chain is α.

Then the expected number of chain candidates to survive the test of chain ζ_i can be estimated as

$$\eta_i = (m-\alpha)\left(\frac{m-\alpha}{255}\right)^{n_i-1} + \alpha \cdot \alpha^{n_i-1} \, .$$

Assuming the independence of all chain tests, we obtain an estimation of the key-recovery complexity for the combined attack:

$$C_{\text{recovery}} \approx \prod_{i=1}^{h} \max\left(1, \eta_i\right) \, , \tag{2.6}$$

where the maximum is taken since one has to test at least one candidate for each chain in practice.

2.5 Collision detection

In this Section, we propose improved techniques of collision detection by dimension reduction specially tailored for the Euclidean metric. The practical evaluation of the dimension reduction techniques in a full (combined) collision attack will be given in Section 2.6.

2.5.1 Dimension reduction in side-channel attacks

Dimension reduction is the selection of samples from side-channel traces, usually with the purpose of improving the efficiency of an attack. In a typical setting, the clock frequencies of electronic devices are in the range of at least several MHz. Therefore, the side-channel trace acquired by the digital oscilloscope at an appropriate sampling rate of at least ten million samples per second will contain thousands, if not millions, of samples. In the presence of noise, when many traces are required, this makes the trace processing a determining part of the attack complexity and may sometimes even render the attack infeasible. On the other hand, it has long been known that only few samples in a trace would exhibit the leakage, the others being redundant. An example is the dimension reduction for a DPA attack (trace compression) [101], where we can limit ourselves to points at clock cycle maxima or to points exhibiting the highest variation across the traces corresponding to different input values. Note that while DPA is still feasible without dimension reduction, attacks employing multivariate methods (e.g. template-like and MIA attacks) are infeasible without an appropriate point selection.

Besides the reduced signal processing complexity, another effect of dimension reduction is the potential increase in the attack success rate. The points removed from the side-channel traces would normally carry more noise than the informative signal, while the opposite applies to the selected points. Therefore, dimension reduction would lead to the overall increase in the signal-to-noise ratio (SNR). In most cases, this will lower the number of measurements required to reach a given success rate. Again, DPA attacks are a good example: in our experiments, selection of cycle maxima or points with the largest variation across the power traces reduces the trace count for the key-byte recovery.

We would like to stress that dimension reduction is *not* a full-scale profiling; it indeed requires some additional knowledge about the implementation, but this knowledge is much less than one would normally impose in traditional profiled attacks to build templates. As opposed to that, in its essence, dimension reduction can be seen as knowledge about the *time moments* when certain computations are carried out, not *values* being processed.

Collision attacks can benefit a lot from the dimension reduction. First, in the following, we show how to select the points from the traces to improve collision detection. We utilize Euclidean distance for trace comparison to distinguish between collisions and non-collisions. We also deal with this in the information-theoretic sense by introducing a metric for comparing our dimension reduction techniques. Second, collision detection demands for higher sampling rates (as compared to DPA or other attacks working in the Hamming weight/distance leakage model) [28] to capture subtle differences between the traces, so reduction in the number of samples in a trace is desirable to decrease $C_{\text{processing}}$, which is quadratic in the number of samples and traces.

We start with a statistical model of Euclidean distance that we first introduced in [28] and provide here for completeness to support our intuition behind the choice of dimension reduction techniques in the sequel.

2.5.2 Statistical model for Euclidean distance

Given two traces $\tau_1 = (\tau_{1,1}, \dots, \tau_{1,l}) \in \mathbb{R}^l$ and $\tau_2 = (\tau_{2,1}, \dots, \tau_{2,l}) \in \mathbb{R}^l$, we assume that each point $\tau_{i,j}$ can be statistically described as $\tau_{i,j} = s_{i,j} + r_{i,j}$, where $s_{i,j}$ is signal constant (without noise) for the given time point i as well as some fixed input to the S-box, and $r_{i,j}$ is Gaussian noise due to univariate normal distribution[1] with mean 0 and some variance σ^2 remaining the same for all time instances in our rather rough model. Let τ_1 and τ_2 correspond to some S-box inputs a_1 and a_2.

If $a_1 = a_2$, the corresponding deterministic signals are equal (that is, $s_{1,j} = s_{2,j}$ for all j's) and one has:

$$E(\tau_1, \tau_2)_{a_1=a_2} = \sum_{j=1}^{l} (\tau_{1,j} - \tau_{2,j})^2 = \sum_{j=1}^{l} \xi_j^2 = 2\sigma^2 \sum_{j=1}^{l} \eta_j^2,$$

where $\xi_j = r_{1,j} - r_{2,j}$, $\xi_j \sim \mathcal{N}\left(0, 2\sigma^2\right)$ and $\eta_j \sim \mathcal{N}\left(0, 1\right)$. That is, statistic $E(\tau_1, \tau_2)_{a_1=a_2}$ follows the chi-square distribution with l degrees of freedom up to the coefficient $2\sigma^2$. As the chi-square distribution is approximated by normal distribution for high degrees of freedom, one has the following

Proposition 1. *Statistic*

$$E(\tau_1, \tau_2)_{a_1=a_2} = \sum_{j=1}^{l} (\tau_{1,j} - \tau_{2,j})^2$$

for $\tau_i = (\tau_{i,1}, \dots, \tau_{i,l}) \in \mathbb{R}^l$ *with* $\tau_{i,j} \sim \mathcal{N}\left(s_{i,j}, \sigma^2\right)$ *can be approximated by normal distribution* $\mathcal{N}(2\sigma^2 l, 8\sigma^4 l)$ *for sufficiently large* l*'s.*

Alternatively, if $a_1 \neq a_2$, one has

$$E(\tau_1, \tau_2)_{a_1\neq a_2} = \sum_{j=1}^{l} (\tau_{1,j} - \tau_{2,j})^2 = \sum_{j=1}^{l} \left(\delta_j^{(1,2)} + \xi_j\right)^2 = 2\sigma^2 \sum_{j=1}^{l} \nu_j^2 ,$$

where

$$\delta_j^{(1,2)} = s_{1,j} - s_{2,j}, \quad \xi_j = r_{1,j} - r_{2,j},$$

$$\xi_j \sim \mathcal{N}\left(0, 2\sigma^2\right) \text{ and } \nu_j \sim \mathcal{N}\left(\delta_j^{(1,2)}/\sqrt{2}\sigma, 1\right) .$$

That is, statistic $E(\tau_1, \tau_2)_{a_1\neq a_2}$ follows the *noncentral* chi-square distribution with l degrees of freedom and $\lambda = \sum_{j=1}^{l} \left(\delta_j^{(1,2)}/\sqrt{2}\sigma\right)^2$ up to the coefficient $2\sigma^2$. Again, we have an approximation using

Proposition 2. *Statistic*

$$E(\tau_1, \tau_2)_{a_1\neq a_2} = \sum_{j=1}^{l} (\tau_{1,j} - \tau_{2,j})^2$$

---

[1] The real measured power consumption is often due to the generic multivariate normal distribution. However, almost all entries of the corresponding covariance matrix are close to zero. Thus, the model with independent multivariate normal distribution seems to be quite realistic.

for $\tau_i = (\tau_{i,1}, \dots, \tau_{i,l}) \in \mathbb{R}^l$ *with* $\tau_{i,j} \sim \mathcal{N}\left(s_{i,j}, \sigma^2\right)$ *can be approximated by normal distribution* $\mathcal{N}\left(2\sigma^2(l+\lambda), 8\sigma^4(l+2\lambda)\right)$ *with* $\lambda = \sum_{j=1}^{l}\left(\delta_j^{(1,2)}/\sqrt{2}\sigma\right)^2$ *for sufficiently large* l*'s.*

2.5.3 Dimension reduction with signal variance (SV)

Signal variance is known [101] to be an adequate indicator of the points leaking information in DPA. We will refer to signal variance as SV. More formally, given a fixed level of noise, the points of traces with higher values of $\mathrm{var}(s_{i,j})$ correspond to higher values of SNR [101]. An estimation of variance $\mathrm{var}(s_{i,j})$ is computed for each point j of the trace over input values $a_i \in \mathbb{F}_{2^8}$.

In the sequel, we use SV as a reference dimension reduction technique and show that for collision detection more efficient criteria exist.

2.5.4 Dimension reduction with signal difference (SD)

In the comparison of the two traces with the Euclidean distance, we try to distinguish between the collisions and non-collisions, i.e. between the distributions $E(\tau_1, \tau_2)_{a_1 \neq a_2}$ and $E(\tau_1, \tau_2)_{a_1 = a_2}$. As described above these statistics approximately follow normal distribution for large numbers of trace points. To efficiently distinguish between these two statistics it is crucial to decrease their variances while keeping the difference of their means high. For this purpose, to better distinguish between the collisions and non-collisions, we proposed to discard [28] points of traces with small minimal contribution to the difference of means[2].

To illustrate this method of dimension reduction, we assume for the moment that $\delta_j^{(1,2)} = 0$ for $j > l/2$ and $\delta_j^{(1,2)} \neq 0$ for $j \leq l/2$ with l even, that is, the second half of the trace does not contain any data dependent information. Then we can discard the second halves of the both traces τ_1 and τ_2 in the comparison with the Euclidean distance and compute two related statistics on the rest of the points:

$$E'(\tau_1, \tau_2)_{a_1 = a_2} = \sum_{j=1}^{l/2} (\tau_{1,j} - \tau_{2,j})^2,$$

$$E'(\tau_1, \tau_2)_{a_1 \neq a_2} = \sum_{j=1}^{l/2} (\tau_{1,j} - \tau_{2,j})^2.$$

This will adjust the means and variances of the approximating normal distributions: $\mathcal{N}\left(\sigma^2 l, 4\sigma^4 l\right)$ and $\mathcal{N}\left(2\sigma^2(l/2+\lambda), 8\sigma^4(l/2+2\lambda)\right)$, respectively. Note that the difference of means remains unaffected and equal to $2\sigma^2\lambda$. At the same time both variances are reduced, one of them by factor 2, which allows one to distinguish between these two distributions more efficiently and, thus, to detect collisions more reliably.

---

[2] A more general way is weighting the points by their contribution to the difference of means and using weighted Euclidean distance as a metric, however our experiments have shown that point selection, being an extreme case of point weighting, is much more efficient.

Definition of SD. More generally speaking, for AES we have to reliably distinguish between inputs in each (a_{i_1}, a_{i_2}) of the $\binom{256}{2}$ pairs of byte values, $a_{i_1}, a_{i_2} \in \mathbb{F}_{2^8}$. Thus, the most informative points j of the traces are those with maximal minimums of $|\delta_j^{(i_1,i_2)}|$ over all pairs of different inputs, that is, points j with maximal values of

$$\min_{a_{i_1} \neq a_{i_2}} |\delta_j^{(i_1,i_2)}| \tag{2.7}$$

with $\delta_j^{(i_1,i_2)} = s_{i_1,j} - s_{i_2,j}$, where the value of signal $s_{i,j}$ is estimated by trace averaging in practice. In other words, we first fix a trace sample j and then go over all pairs of non-equal inputs. For each pair, we compute the difference between signals. Among those differences, we take the minimum. Trace samples with maximal minimums are expected to contribute more to collision detection.
We will denote this point selection criterion as *SD (signal difference)* for brevity. For each point of a side-channel trace, the criterion gives its weight relevant for collision detection: the higher the weight, the stronger the contribution of the point to the comparison with Euclidean distance. So only points with maximal weights are selected for collision detection.

Comparison to SV. We estimated the values of SD for all time instances j of the two (most leaky) cycles of the S-box look-up in our reference AES implementation (i.e. for all points of corresponding side-channel traces) and compared this to the signal variance SV in the same time points, $\mathrm{var}(s_{i,j})$, see Figure 2.5(a). The figure shows the normalized values of the two criteria (weights) for each point of a side-channel trace of the S-box lookup. One can clearly see that values of each criterion for different points of a trace differ significantly. So one can select points with the largest weights of a selected criterion, that is, perform dimension reduction of side-channel traces. Figure 2.5(a) reveals that the point selection based on SD is more fine-grain than that based on SV. In Section 2.5.7 we will compare the effect of dimension reduction using different criteria with an information-theoretic metric.

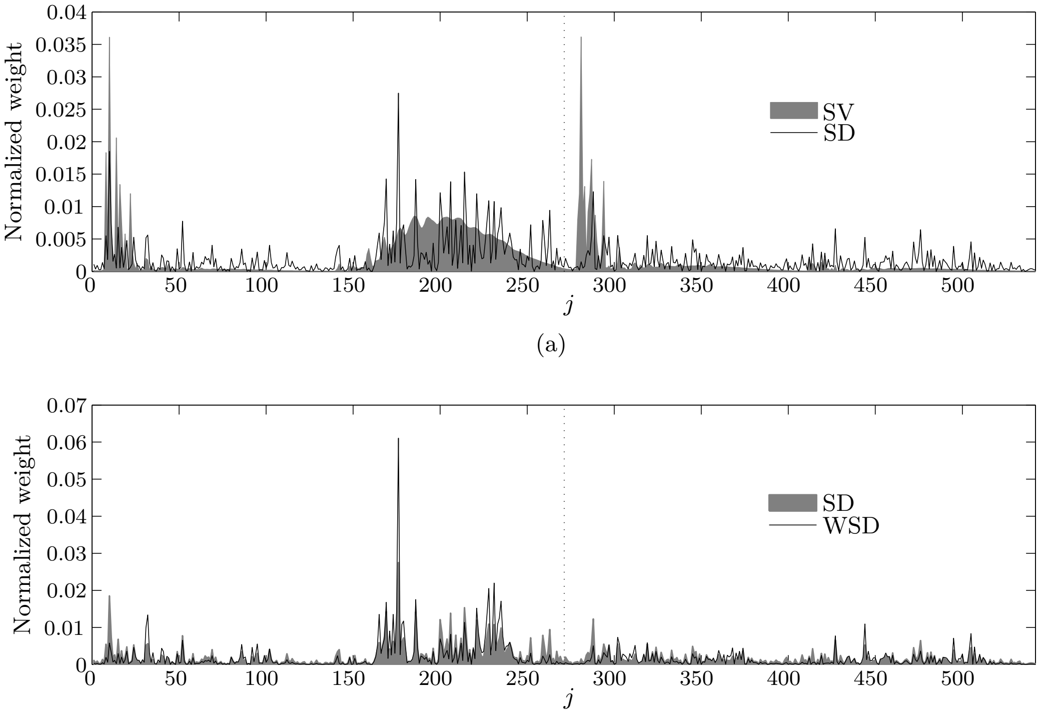
2.5.5 Dimension reduction with weighted signal difference (WSD)

Definition of WSD. To further amplify the selection of points for collision detection, when calculating the contribution of the point j to the Euclidean distance one can consider not only (2.7) but also noise variances in this point. Thus, a more efficient criterion, that we call *WSD (weighted signal difference)*, can be defined choosing points that maximize

$$\min_{a_{i_1} \neq a_{i_2}} \frac{|\delta_j^{(i_1,i_2)}|}{\mathrm{var}(r_{i_1,j}) + \mathrm{var}(r_{i_2,j})}$$

with $\mathrm{var}(r_{i,j})$ being the variance of the noise for input i in trace sample j. In practice, the variance is estimated using several traces for a single input. The intuition behind this additional weighting of points by the inverse of the noise variance value is to exclude points that contribute more noise to Euclidean distance.

Comparison to SD. In Figure 2.5(b), we compare WSD to SD for the same two clock cycles of our reference implementation. One can see the differences:



(a)

(b)

Figure 2.5: Different dimension reduction criteria for each point j of a side-channel trace for an AES S-box lookup: (a) signal difference (SD) versus signal variance (SV); (b) signal difference (SD) versus signal difference weighted with noise variance (WSD)

weighting by the noise variance increases the contribution of some points while decreasing the contribution of the others (WSD) compared to the pure signal difference (SD). This difference will be well captured by the information-theoretic metric that we will define below and use to compare the techniques in a sound way.

2.5.6 λ-divergence as an information-theoretic metric

As mentioned above, the goal of collision detection is to efficiently distinguish between collisions and non-collisions, that is, between the distribution of the Euclidean distance for a pair of equal and non-equal inputs. Here we propose an information-theoretic measure of difference between these distributions.

Let $\Pr[X]$ be the probability distribution over all possible secret values to be recovered using side-channel information. In the case of collision attacks, X is a set of two elements: $X = \{\text{collision}, \text{non-collision}\}$. Further let $\mathcal{L} = \Pr[L]$ be the probability distribution of side-channel leakage as measured by a collision detection method. For instance, L can be the set of all possible values of the Euclidean distance between side-channel traces for pairs of inputs to the S-box. Correspondingly, let $\Pr[L \mid X = x]$ be the probability distribution of leakage, taken separately for collisions and non-collisions, depending on X. We will denote the probability distributions $\mathcal{L}_\mathrm{C} = \Pr[L \mid X = \text{collision}]$ and $\mathcal{L}_\mathrm{N} = \Pr[L \mid X = \text{non-collision}]$. (Note that in Section 2.5.2 we have described these distributions in the independent multivariate Gaussian model.)

Our metric uses the notion of Kullback-Leibler divergence between two distributions $\mathcal{A}$ and $\mathcal{B}$, $D_\mathrm{KL}(\mathcal{A}\|\mathcal{B})$. For discrete distributions, it is defined as

$$D_\mathrm{KL}(\mathcal{A}\|\mathcal{B}) = \sum_i \mathcal{A}(i) \log_2 \frac{\mathcal{A}(i)}{\mathcal{B}(i)} \,. \tag{2.8}$$

Note that D_KL is not commutative and $D_\mathrm{KL}(\mathcal{A}\|\mathcal{B}) \neq D_\mathrm{KL}(\mathcal{B}\|\mathcal{A})$.

To compare collision detection methods, we use the λ-divergence between leakage distributions for collisions and non-collisions which is defined as

$$\begin{aligned} D_\lambda(\mathcal{L}_\mathrm{C}\|\mathcal{L}_\mathrm{N}) = \lambda D_\mathrm{KL}(\mathcal{L}_\mathrm{C}\|\lambda\mathcal{L}_\mathrm{C} + (1-\lambda)\mathcal{L}_\mathrm{N}) +& \\ +(1-\lambda) D_\mathrm{KL}(\mathcal{L}_\mathrm{N}\|\lambda\mathcal{L}_\mathrm{C} + (1-\lambda)\mathcal{L}_\mathrm{N}) \,,& \end{aligned} \tag{2.9}$$

where λ is the a priori probability of a collision, in other words, $\lambda = \Pr[X = \text{collision}]$. Note that we have then

$$\lambda\mathcal{L}_\mathrm{C} + (1-\lambda)\mathcal{L}_\mathrm{N} = \mathcal{L} \,.$$

For the 8-bit S-box of the AES $\lambda = 1/256$.

It is well known that mutual information can be expressed in terms of the Kullback-Leibler divergence. Similarly, it can be shown that the λ-divergence metric as introduced above and the mutual information metric [152] when applied to collision detection instead of template attacks are equivalent.

Proposition 3. *$I(X, L) = D_\lambda(\mathcal{L}_\mathrm{C}\|\mathcal{L}_\mathrm{N})$.*

Proof. The left-hand side of the above equality is $I(X, L) = H(L) - H(L \mid X)$ by definition. One can transform the right-hand side using definitions (2.8) and (2.9), and knowing that $\lambda = \Pr[X = \text{collision}]$ obtain the left-hand side:

$$\begin{aligned} D_\lambda(\mathcal{L}_\mathrm{C}\|\mathcal{L}_\mathrm{N}) &= \lambda D_\mathrm{KL}(\mathcal{L}_\mathrm{C}\|\mathcal{L}) + (1-\lambda) D_\mathrm{KL}(\mathcal{L}_\mathrm{N}\|\mathcal{L}) = \\ &= \lambda \sum_i \mathcal{L}_\mathrm{C}(i) \log_2 \frac{\mathcal{L}_\mathrm{C}(i)}{\mathcal{L}(i)} + (1-\lambda) \sum_i \mathcal{L}_\mathrm{N}(i) \log_2 \frac{\mathcal{L}_\mathrm{N}(i)}{\mathcal{L}(i)} = \\ &= \left[-\lambda \sum_i \mathcal{L}_\mathrm{C}(i) \log_2 \mathcal{L}(i) - (1-\lambda) \sum_i \mathcal{L}_\mathrm{N}(i) \log_2 \mathcal{L}(i) \right] - \\ &- \left[-\lambda \sum_i \mathcal{L}_\mathrm{C}(i) \log_2 \mathcal{L}_\mathrm{C}(i) - (1-\lambda) \sum_i \mathcal{L}_\mathrm{N}(i) \log_2 \mathcal{L}_\mathrm{N}(i) \right] = \\ &= H(L) - H(L \mid X). \end{aligned}$$

□

So the information-theoretic metric of the unified framework [152] applies well to the collision detection procedure.

The metric is interpreted in the sense that its higher values (i.e. higher mutual information $I(X, L)$) mean better distinguishing between the distributions and therefore better collision detection. In the following, we use the λ-divergence to compare our collision detection techniques to each other and to the existing techniques.

2.5.7 Comparison of dimension reduction techniques

For the comparison of collision detection techniques using the λ-divergence, one has to know the distributions $\mathcal{L}_\mathrm{C}$ and $\mathcal{L}_\mathrm{N}$. The only way to obtain these distributions is to estimate them empirically. The same problem of estimating the leakage distribution arises in MIA attacks and several distribution estimation methods have been reported to be used [121]. We have opted for the histogram method, that is, we obtain the histograms for $\mathcal{L}_\mathrm{C}$ and $\mathcal{L}_\mathrm{N}$ from the samplings of Euclidean distance using the side-channel traces from our reference implementation for equal and non-equal inputs, respectively. Then we compute $D_\lambda(\mathcal{L}_\mathrm{C} \| \mathcal{L}_\mathrm{N})$ following (2.9) in a straightforward way, setting $\lambda = 1/256$.

Figure 2.6 shows an example of distributions $\mathcal{L}_\mathrm{C}$ and $\mathcal{L}_\mathrm{N}$ from our experiments. More precisely, the histogram for $\mathcal{L}_\mathrm{C}$ describes the distribution of the Euclidean distance between traces corresponding to pairs of colliding inputs, whereas the histogram for $\mathcal{L}_\mathrm{N}$ presents the distribution of the Euclidean distance between the traces corresponding to pairs of non-colliding inputs. The distributions in this example are visually different.

We have evaluated our two new dimension reduction techniques SD and WSD, the technique SV commonly used in DPA, and detection without dimension reduction as a reference point. We tried averaging t traces to capture the effect of noise reduction. Figure 2.7 presents the experimental results for the considered techniques. The figure demonstrates the gain of collision-related key information brought by one comparison. The horizontal axis shows the data complexity, namely, the number t of trace averagings before comparison. The vertical axis shows information D_λ, measured in bits. Different curves correspond to different dimension reduction methods. Dimension reduction by variance, SV, which works well in DPA attacks, only moderately improves collision detection. Whereas both new methods, SD and WSD, lead to clearly more efficient collision detection, WSD being superior. In other words, they provide lower data complexity for the same amount of information delivered by one comparison. As expected, the information gain grows with the averaging, since the latter increases the SNR.

2.6 Practical evaluation

Here we show that the techniques we introduce in this work perform well in practice. We implement (linear) collision attacks (Section 2.2) following our combination framework with the DPA-driven test of chain (Section 2.3) and employing the new collision detection methods (Section 2.5) against the target implementation. We experimentally estimate the efficiency of these DPA-combined collision attacks.

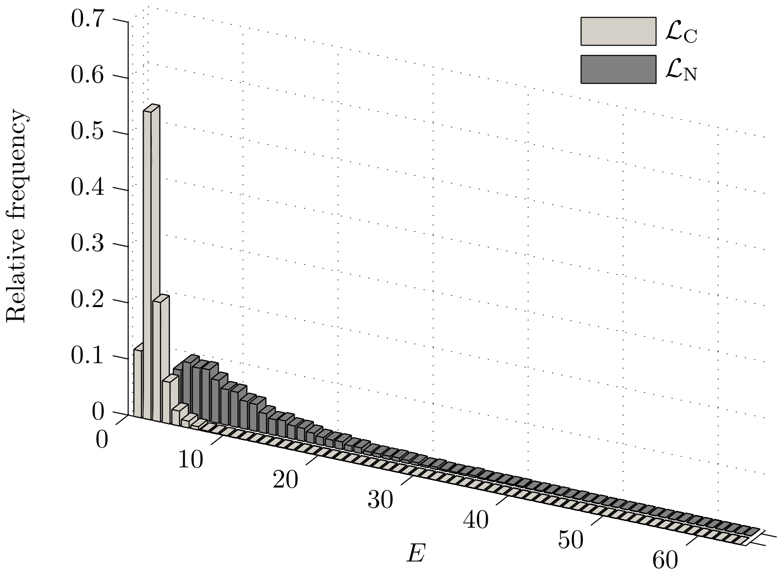


Figure 2.6: Empirical distributions for $\mathcal{L}_\mathrm{C}$ and $\mathcal{L}_\mathrm{N}$

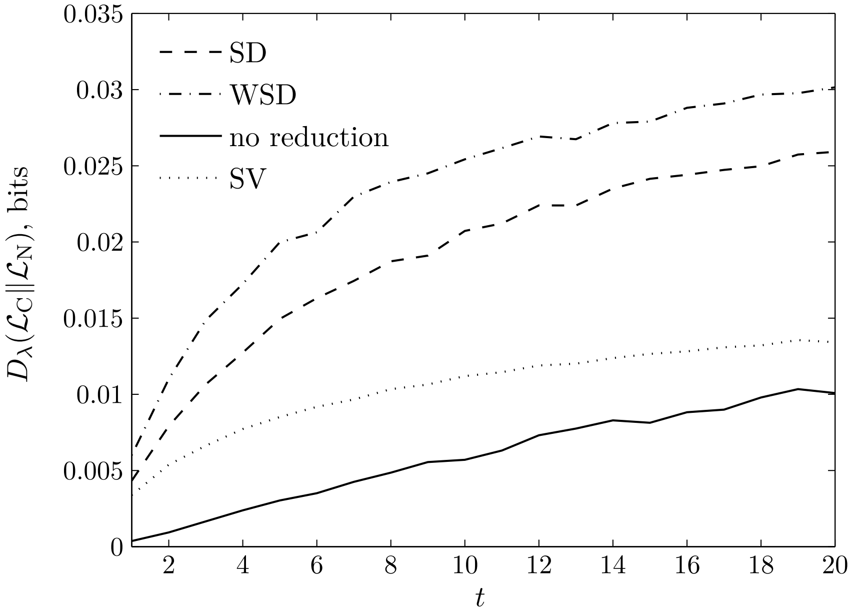


Figure 2.7: Effect of dimension reduction on collision detection measured with information-theoretic metric D_λ

2.6.1 AES implementation and measurement equipment

We performed our attacks for a typical AES-128 implementation on the Atmel ATmega16, a RISC μC from the 8-bit AVR family following a Harvard architecture. 8-bit AVR μC are widely used in embedded devices.

To run a collision attack, the attacker has to know when the AES S-boxes are executed. So we measured the power consumption of the table look-ups corresponding to the relevant S-box applications. These include instances in `SubBytes`, `ShiftRows`, and `MixColumns` operations. The typical AES operations involving a single state byte were implemented in the AVR assembly as follows.

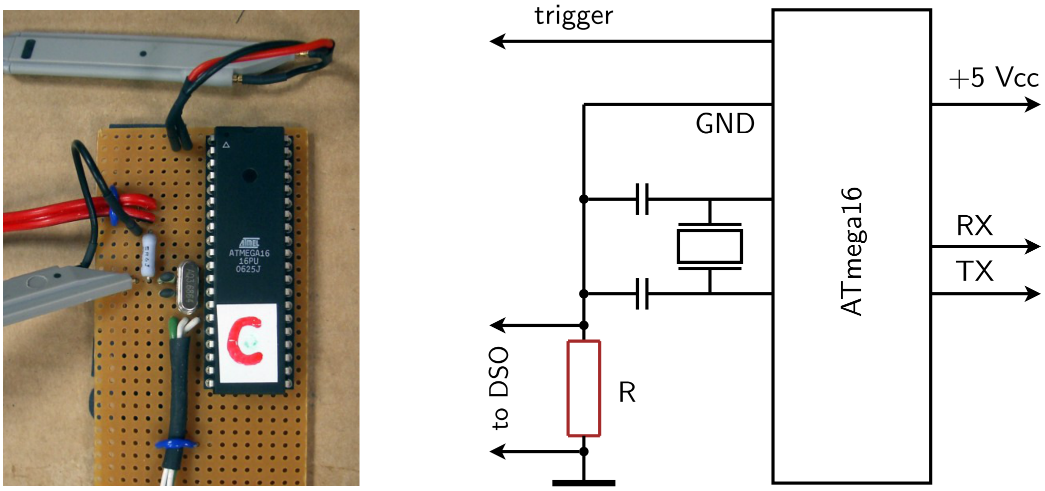
```
; SubBytes
    mov  ZL, R16   ; load input byte, 1 cycle
    lpm            ; perform S-box lookup, 3 cycles
    st Y, R0       ; store S-box output byte into SRAM, 2 cycles
; ShiftRows
    ld R1, Y       ; load state byte, 2 cycles
    st X, R2       ; store it to another location in SRAM, 2 cycles
; pre-MixColumns
    ld R2, X       ; load it again, 2 cycles
```

The S-box lookup table was stored in the program memory and accessed with the `LPM` instruction of the μC. SRAM was used to store the state byte. The trigger signal was raised by the implementation on a microcontroller's pin prior to execution of the above instructions.

The μC was clocked at 3.68 MHz and supplied with an operating voltage of 5 V from a standard laboratory power source. The variations of the power consumption were observed on a shunt resistor of 5.6 Ohm inserted into the ground line of the microcontroller. Our target board with the μC is shown in Figure 2.8. The measurements were performed with a LeCroy WaveRunner 104MXi DSO equipped with ZS1000 active probe. The DSO has 8-bit resolution and 1 GHz input bandwidth (with the specified probe). The acquisitions were performed at the maximum sampling rate of 10 GS/s without any input filters. A Faraday cage made of the copper foil was used, but the measurement setup was *not* noise-optimized in any other way. For example, we did not use optical decoupling in the serial line, while the level converter that is necessary in the case of the galvanic coupling is know to be a source of noise [144].

With the given shunt resistor and 160 mV peak-to-peak vertical resolution set on the DSO, variations of the consumed current as small as 110 μA could be observed, while the average power consumption of the microcontroller in the given configuration was about 12 mA. The DSO, the power source and the target device shared the common ground, also connected to the Faraday cage.

The DSO was controlled remotely via Gigabit Ethernet connection by the MATLAB script running on the host PC and using LeCroy's ActiveDSO ActiveX component. The same script was sending the inputs to the microcontroller over the serial line. The DSO was set to the sequence acquisition mode, which enabled to achieve the acquisition rate of 150 traces per second. Each trace consisted initially of 46000 samples.



(a) (b)

Figure 2.8: Our target board with ATmega16: photo (a) and schematics (b)

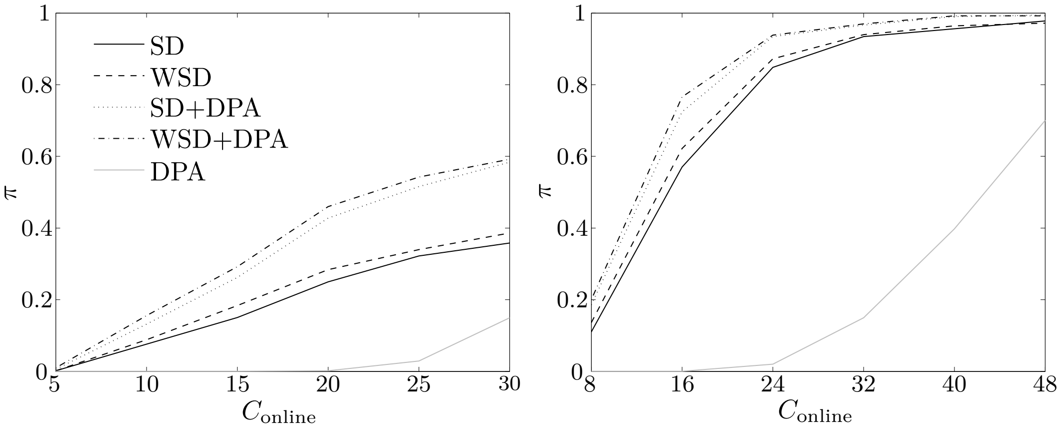
As in case of DPA [36], collision detection methods tend to be sensitive to the pre-processing of measured signals. To pre-process the traces, we proceed in two steps. First, the traces are decimated by applying a low-pass filter to the original traces and subsequently resampling them at a lower rate. Additionally to noise reduction, this was reported to weaken time jitter [144]. We decimated the traces 10 times, to 4600 samples. Second, the decimated traces are denoised by applying a wavelet decomposition at a certain level, thresholding the detail coefficients, and subsequent wavelet reconstruction. Wavelets of the Symlet family proposed by Daubechies that were used in [36] have shown to be most suitable for this operation in our experiments as well (first of all, the 'sym3' wavelet). Pre-processed traces were stored along with the corresponding input bytes.

Finally, we reduced the dimension of the pre-processed traces to 900 samples using our techniques SD or WSD. This number was chosen empirically by observing when the attack efficiency reached saturation. The component of DPA was implementing CPA in the Hamming weight model [31].

2.6.2 Practical combined attacks

Launching a series of 500 attacks for a given number of inputs γ and averagings t, we experimentally estimated the efficiency parameters defined in Section 2.4 of all attacks in question. Namely, besides the online complexity $C_{\text{online}} = t \cdot \gamma$, we obtained the success probability π by counting the number of successful attacks: we considered an attack successful if it was possible to recover the correct full AES key with $C_{\text{recovery}} \leq 2^{40}$.

Instead of using a fixed or adaptive (like in [28]) threshold for the value of Euclidean distance E in collision detection, we follow another approach that allows to improve the attack efficiency. We consider a list of *collision candidates* consisting



(a) $\gamma = 5$ (b) $\gamma = 8$

Figure 2.9: Attack success rate π in practice against the measurement complexity C_{online} for different number of inputs γ

of all $(16 \cdot \gamma)^2/2 - 8 \cdot \gamma$ S-box instance pairs sorted by E ascending. Taking c top candidates out of this list, starting from $c = 1$, we determine the number of chains h and their lengths n_i, $i = 1, \ldots, h$. In case of the collision-only attack, $C_{\text{recovery}} = 2^{8h}$ and we can have h at most 5 to stay within the admissible bound of 2^{40}. If h is higher, we take one more collision (increment c). Once we have enough collisions to attain $h = 5$, we perform the key recovery and check if the correct full key is among the recovered candidates. A similar approach can be used in case of the DPA-combined collision attack.

We experimentally characterized 4 specific attacks employing our techniques: collisions using SD, collisions using WSD, collisions using SD combined with DPA, and collisions using WSD combined with DPA. Comparison of these 4 attacks in terms of the success rate π of the full key recovery for different number of distinct inputs γ against the total measurement complexity $C_{\text{online}} = t \cdot \gamma$ is presented in Figure 2.9. As a reference, we also plot the success rate of the DPA-only attack with $N = C_{\text{online}}$ inputs (note that in a DPA attack $t = 1$) with C_{recovery} also bounded by about 2^{40}, i.e. when about 2^{40} most probable AES key candidates as suggested by DPA are tested.

One can see that the combination of collision attacks with DPA clearly outperforms both collision-only and DPA-only attacks. The dimension reduction technique WSD outperforms SD, thus, conforming to the information-theoretic comparison in Section 2.5.7.

2.7 Conclusions and open problems

In this Chapter, we presented combined divide-and-conquer and collision attacks using side-channel leakage against implementations of cryptographic algorithms. Our experimental results suggest that combined attacks exploit more information in the side-channel scenario than their stand-alone components. Below we present a relation of our attacks to the existing ones and outline some open problems.

Unlike collision attacks, template attacks require, in addition to a proper dimension reduction, detailed knowledge of the implementation for profiling. This appears to be a much weaker attack model than the one we use. However, the evaluation of template-combined collision attacks using our framework is still an open problem. This combination can possibly reduce the cost of the profiling stage. Another line of future research, initiated in [25], is using profiling to improve collision detection.

As collision and template attacks, MIA [12] can work with the leakage models where DPA attacks do not work. However, as demonstrated in [121], MIA tends to be significantly less efficient than DPA in terms of the required number of traces for implementations exhibiting a simple (linear) leakage model, even in the presence of strong noise. It is another open problem to evaluate MIA-combined collision attacks using our framework. As in the case of template attacks, we expect this combination to result in a reduced complexity.

While DPA, MIA and template techniques have been naturally incorporated by the unified framework for comparing side-channel attacks [152], such analytical techniques as collision attacks, considered here, and algebraic attacks [126], cannot be reasonably captured by the unified framework directly. We consider it an important open problem to come up with a development of the unified framework both practically and generically applicable to analytic attacks. However, in this work we successfully applied metrics similar to those of [152] to study some local properties of collision attacks.

From an information-theoretic perspective, each comparison of two traces with the purpose of collision detection, should yield in our attacks up to 0.03 bit key-related information (see Figure 2.7). However, not all of it is used for key recovery afterwards, where only collisions result in equations. At the same time, detected non-collisions also carry useful information ignored by the current techniques. Their usage seems to be technically problematic, since each non-collision would add an equation of a high degree to the system of equations to be solved. We leave this as another open problem.

To prevent attacks we described in this Chapter, the countermeasures commonly used against side-channel attacks, namely masking and hiding [101], can be applied. We note that, unlike stand-alone collision attacks, DPA-combined collision attacks cannot naturally overpass certain masking schemes since masking will disable the DPA component. Overpassing masking might be possible in case higher-order DPA [101] or template attacks are used as a divide-and-conquer component of a combined attack. Practicability of such combinations against a masked implementation is an open question.

Chapter 3

Side-channel trace driven cache-collision attacks

In this Chapter we elaborate on the attacks that exploit cache events, which are visible in some side channel, to derive a secret key used in an embedded software implementation of AES. We focus on making trace driven cache collision attacks tolerant to errors in cache event detection that can take place in a noisy measurement environment. Our basic known plaintext attack can recover the full 128-bit AES secret key with approximately 30 side-channel measurements, with the offline working time of several seconds on a common PC. This is comparable to classical DPA; additionally, our attacks are able to overcome certain boolean masking variants. We also improve the attack presented in ACISP 2006 [58].

We perform practical explorations of cache event detection in the electromagnetic leakage of a 32-bit ARM μC. Finally, we show that previous univariate models for estimating attack complexity are not good, and present the multivariate model which is easy to simulate.

This is a joint work with Jean-François Gallais and Michael Tunstall, published in [60], and with Jean-François Gallais, published in [59].

Contents

3.1 Introduction

Fetching data from the random access memory or non-volatile memory in embedded microprocessors can take a significant number of clock cycles and a processor is unable to perform any further instructions while it waits. The use of cache memory aims to decrease the cost of memory accesses. Cache memory is a memory held within the core of the microprocessor that can be accessed rapidly. When data is accessed the line of data holding this address is moved to the cache, where the amount of data moved is dictated by the architecture of the cache. This is based on the assumption that when a certain address is accessed it is likely that the data around this address is also likely to be accessed in the near future.

It has been noted that the power consumption of a microprocessor is dependent on the instruction being executed and on any data being manipulated [88, 31]. An attacker can, therefore, observe where functions, and sequences of functions, occur in a power consumption trace. This could, potentially, allow an attacker to derive information on cryptographic keys if an observed sequence is affected by the value of the key. It has also been observed that the electromagnetic field around a microprocessor also has this property [61, 122].

3.1.1 Cache-collision attacks

Cache mechanism of the microprocessors has been known to be able to leak key-dependent information and therefore to be exploitable in side-channel attacks for about 10 years [78, 115]. Since then, many cache-based attacks were reported; they fall into three different types. *Time-driven attacks* like [15] exploit the dependence of the execution time of an algorithm on the cache accesses. In *access-driven attacks* presented in [109, 114], an attacker learns which cache lines were accessed during the execution by pre-loading the cache with the chosen data (which requires running a spy process on the device in parallel to the target algorithm). Finally, in *trace-driven attacks* first suggested in [16] an adversary derives information from individual cache events from the side-channel trace of an execution, such as registered power consumption or electromagnetic emanations. Trace-driven attacks pose a particular threat to embedded devices since the latter are exposed to a high risk of power or electromagnetic (EM) analysis, as opposed to desktop and server implementations that are a usual target in access- and time-driven cache attacks.

Like collision [141, 24] and algebraic attacks [126], trace-driven cache collision attacks recover the key from the relations yielded by the side-channel leakage. In this, they are different from differential power analysis (DPA) [88] which is a divide-and-conquer attack known to be very practical. Our motivation here is to make trace-driven cache collision attacks comparable, if not superior, to DPA in terms of both online (number of measurements) and offline (recovering the key from the measurements) complexity.

3.1.2 Our contributions

In this work we consider the effect of a cache on an instantiation of AES. Given the above observation, cache accesses should be visible in the power consumption or electromagnetic emanations. The location of these cache accesses during the computation of AES has been shown to reveal information on the secret key used [16, 58].

We start with presenting a significant improvement of the adaptive chosen plaintext trace-driven attack of [58] with a new adaptive algorithm for choosing plaintexts enabling the recovery of the 60 bits of the key with the expected number of measurement being 14.5 instead of 127 for the original attack (Section 3.3). Then we present a known-plaintext attack requiring only 30 traces and a remaining exhaustive search in 10 hypotheses to recover the full 128-bit AES secret key (Section 3.4). This attack follows the same approach as in [2] and [30], however we treat the case of conventional 256-byte S-box lookup tables (since such would be used in a very constrained or masked implementation), and precisely describe the full key recovery process, including the second round part of the attack. While being comparable to DPA in terms of complexity, our attacks are able to overcome certain masking techniques.

Our main contribution is in making the attacks tolerant to uncertainties in cache event detection. Our practical explorations show that cache events corresponding to AES S-box lookups are well distinguishable in the power consumption and EM radiation of a microcontroller (Section 3.5). However, side-channel measurements may be noisy in a real-life scenario, making it hard to distinguish cache events reliably. We show (Section 3.6) that even for significant probabilities of detection uncertainties, the error-tolerant versions of our attacks still require a feasible number of measurements to succeed. Our attacks can also deal with the condition when the cache already holds a part of the S-box lookup table prior to the algorithm execution, in a manner described in [30].

Finally, we scrutinize the theoretical model of the attacks and show that while a univariate model can be used to roughly estimate the attack measurement complexity, the real attack follows a multivariate model (Section 3.7). Our practical results match the theoretical estimations.

3.1.3 Notation

Throughout this Chapter, we denote

- the most and the least significant nibbles of a byte b with $\widehat{b}$ and $\check{b}$ correspondingly;
- the input of the `SubByte` function in the first AES round as x_i, equal to $p_i \oplus k_i$, where p_i and k_i respectively represent the plaintext byte and the corresponding key byte, $0 \le i \le 15$;
- addition and multiplication over GF(2^8) by $\oplus$ and $\bullet$ respectively.

We index the bytes row-wise and *not* column-wise as in the AES specification [171, 49], i.e. in our notation p_0, p_1, p_2, p_3 is the first row of a 16-byte plaintext (we assume that in an embedded software AES implementation S-box lookups are performed row-wise, so indexing bytes in the order of S-box computation simplifies the description of our algorithms).

3.2 Generalities and previous work

In this Section, we present an overview of caching, our assumptions about the cache mechanism, related work and relation to side-channel collision attacks.

3.2.1 Caching and performance

The gap between the increased speed at which modern microprocessors treat data and the comparatively slow latencies required to fetch the data from the Non-Volatile Memories to the registers raise performance issues. To reduce the "distance" between the CPU and the NVM, i.e. the number of wasted clock cycles for which the CPU has to wait for the data, the solution is to keep them quickly accessible in a faster memory. Faster, however, typically means more expensive, hence this choice affects the size of available fast storage memory, the so-called *cache memory.* Examples of embedded devices with cache memory are the microprocessors of the widespread ARM9 family and of the subsequent ARM families [7].

Concretely, modern microprocessors typically come with a SRAM cache memory. When a byte of data must be paged in during the computation, the processor first looks for in the cache. If present in the cache, this results in a **cache hit**, the data is brought to the registers within a single clock cycle without stalling the pipeline. If not present in the cache, this results in the **cache miss**, and the desired data fetched from Non-Volatile Memory (NVM), and the entire line containing the desired data is loaded into the cache. As suggested by the different technologies used in the cache and main memory, a cache miss typically takes more clock cycles and consumes more energy than a cache hit.

3.2.2 Cache-based attacks: related work

Following the pioneering articles of Kelsey et al. [78] and Page [115], several notorious attacks have been published involving the cache mechanism and targeting AES. Cache-based attacks fall into three different types. *Time-driven attacks* exploit the dependence of the execution time of an algorithm on the cache accesses. Bernstein described a simple cache-timing attack leading to a complete key recovery on a remote server [15]. In *access-driven attacks* presented in [109, 114], an attacker learns which cache lines were accessed during the execution by pre-loading the cache with the chosen data.

Here, we elaborate on *trace-driven attacks*. In this type of cache attacks an adversary derives information from individual cache events from the side-channel trace of an execution, such as registered power consumption or electromagnetic emanations. Trace-driven attacks pose a particular threat to embedded devices since the latter are exposed to a high risk of power or electromagnetic analysis, as opposed to desktop and server implementations that are a usual target in access- and time-driven cache attacks (however, a cache-timing attack on embedded AES implementation was presented recently in [26]).

Previous work on trace-driven attacks was described in [16, 91, 2, 58, 167]. However, most of these works target an optimized AES implementation that uses large lookup tables, as described in the original Rijndael proposal [49]. Here we focus on a conventional 256-byte lookup table since this would often be the choice in a constrained device, e.g. in a smart card, for the reasons outlined above in the Introduction. Also, previous works did not tackle the unreliable cache event detection, at most considering the setting when a lookup table is partially preloaded into the cache [30]. In [16, 91, 2, 30], the effect of cache organization, and in particular the cache line size, on the attack was considered. Here we develop our attacks assuming the cache line size is 16 bytes, but they can be easily adapted to other sizes. Another popular cache line size is 32 bytes; in this case our attack will be more complex (however, the dependency of the attack complexity on the cache organization is not straightforward, as detailed in [30]).

Also, previous work on trace-driven cache attacks rarely included experiments with real embedded devices and assumed a noise-free environment. Simulations of the side-channel leakage of the cache were performed in [16]. A novel trace-driven cache attack against CLEFIA [124] exploiting differential cryptanalysis methods has been carried out in practice using the power consumption measurements on the PowerPC processor of a Xilinx Virtex-II FPGA.

3.2.3 Relation to side-channel collision attacks

We also note that there is a similarity between cache attacks and side-channel collision attacks [141, 24], as already observed in [91], hence the name *cache-collision* attacks. Here we outline the commons and differences between the two attacks to better explain the nature of the subject.

As opposed to DPA and other divide-and-conquer attacks, both collision attacks and cache-collision attacks do not recover the individual key chunks. Instead, they impose restrictions on the key through a set of equations in the key chunks and thus reduce the key search space, belonging to the class of analytic attacks. Both attacks get the equations involving key bytes from the collisions in the values of the sensitive intermediate variables. Collisions are detectable through side-channel leakage. So both attacks feature two distinct phases: a) detection of collisions in side-channel traces, leading to a set of equations in the subkey chunks, and b) exploitation of collisions by solving (simplifying) the system of equations and thus obtaining a set of key candidates.

At the phase of collision detection, the similarity between the attacks is in the loosened side-channel leakage model. In both attacks, one does *not* need to know exactly *how* does the leakage behave, it suffices to know that it is different for different values of the target variable and the same for the equal values. The difference however is that in collision attacks, collisions are detected by explicit comparison of side-channel traces using some measure of distance, like the sum of the squared differences of corresponding trace points. Whereas in cache-collision attacks, the comparison is implicitly performed by the cache access mechanism of the device itself and the side-channel trace shows already the result of the comparison. So it suffices to distinguish a cache miss from a cache hit in a side-channel trace. But, again, the exact behaviour of the leakage in each of the cases (hit/miss) is not important. In both cases, the issue of reliable detection of collisions (cf. cache events) arises when it comes to the real measurements “in the wild”. The approaches to dealing with detection errors are similar, as described further in Section 3.6.

At the phase of collision exploitation, the attacks are distinct in the structure of the systems of equations. In a cache-collision attack, one looks for collisions between S-box inputs and the contents of the memory cache when the look-up occurs. This is the place where the two important differences from collision attacks arise, namely:

1. the cache may contain multiple entries, which means that a single cache event gives rise either to several possible collisions of which one and the only one is true (in the case of a hit), or to several non-collisions that hold simultaneously (in the case of a miss);

2. the cache entries are normally the entire memory *lines* corresponding to the higher order part of the lookup address, which means that cache events (hits/misses) are related to collisions in the higher order bits of the sensitive variable; no information about the lower order bits is revealed.

Because of this, cache-collisions result in more complex systems of equations compared to the ones arising in collision attacks.

3.2.4 Our assumptions about cache mechanism

Here we present our assumptions about the cache mechanism and the implementation of the AES lookup table that will be used in our attacks. In general, we follow the description made in [30] and assumptions of [58].

We assume that the AES implementation uses lookup tables of 256 entries. Let b be the size of a table entry in bytes. In case of a standard S-box implementation $b = 1$ (unless the 8-bit entries are stored as words of native length for the platform), in case of T-tables used in optimized implementations [49] $b = 4$. Let l be the cache line size in bytes. In modern embedded microcontrollers common sizes are $l = 16$ and $l = 32$. Then, we have $\delta = l/b$ entries per cache line and $m = 256 \cdot b/l$ cache blocks per lookup table (note that $\delta \cdot m = 256$). The value of δ (or, equivalently, m) has effect on the attack complexity (described in [30]) since cache events are determined by equalities/inequalities of $8 - \log_2 \delta$ higher order bits of the inputs to the lookups. We assume that the lookup table is aligned with the cache. Recent work [167] has shown that the attack is also possible when lookup tables are misaligned.

We present our attacks for the case $b = 1$, $l = 16$. The attacks are adaptable to other cache and lookup table configurations. We carry out theoretical analysis (Section 3.7) in general for different cache line sizes, i.e. for different values of m.

We assume that in an embedded software AES implementation S-box lookups are performed row-wise (and therefore our row-wise notation simplifies the description of the attack algorithms). The adversary is dealing with a sequence of *observed* cache events, misses or hits, occurred in the first two rounds of the implementation. We call this sequence a *cache trace*. A cache trace can be recovered from a side-channel trace, as we show in Section 3.6. Since there may be uncertainties in distinguishing a miss from a hit, we also introduce an additional type of observed cache event: the uncertain event.

Like the attacks of [30], our attack do not necessarily require the cache to be clean of lookup table entries prior to each run of the implementation, but can be more efficient under the clean cache assumption.

3.3 Improved chosen plaintext attack

In this Section we recall the attack presented in [58]. We then show how to improve the first round part of this attack to achieve a significant reduction in the number of chosen plaintexts for the recovery of the 60 bits of the key. It will appear however that for the full key recovery the chosen plaintext strategy is inefficient since choosing plaintexts in the second round stage cannot be done efficiently, so we will describe an efficient known plaintext attack in Section 3.4.

3.3.1 Adaptive chosen plaintext attack of ACISP'06

The trace-driven cache-collision attack presented in [58] uses an adaptive chosen plaintext strategy, i.e. each plaintext is chosen according to the result of the analysis done beforehand. The attack follows the assumptions we made in Section 3.2.4.

The first round part of the attack works as follows. An attacker first makes the device encrypt a plaintext (we can assume all its bytes are zero without loosing the generality). Since the cache is clean prior to the AES execution, during the initial S-box lookup of the first round the entire line indexed by the upper nibble of x_0 is loaded to the cache, inducing a first cache miss. The subsequent lookup, indexed by x_1, will be a cache hit with probability $\frac{1}{16}$, as there is one chance over 16 that the values $S(x_0)$ and $S(x_1)$ belong to the same line. If a cache hit occurs, then we have

$$\widehat{k_0 \oplus p_0} = \widehat{k_1 \oplus p_1} \, .$$

By rearranging the terms in the equation, we obtain

$$\widehat{k_0 \oplus k_1} = \widehat{p_0 \oplus p_1} \, .$$

An attacker can search among the 16 possible values for the upper nibble of p_1 (submitting further traces to the device) and find the one inducing a cache hit within an expected number of $\sum_{i=1}^{16} \frac{i}{16} = 8.5$ acquisitions. Once the correct value for the second lookup is found, she can reiterate the process for the 14 other lookups. She will end up with the trace MHH...H and thus with the actual values for $\widehat{k_0 \oplus k_1}, \widehat{k_0 \oplus k_2}, \ldots, \widehat{k_0 \oplus k_{15}}$. This gives a system of 15 linear equations in 16 upper nibbles of the key bytes. This system reduces the key search space by 60 bits. Algorithm 3 presents this adaptive strategy.

Algorithm 3: Adaptive Chosen Plaintext Trace-Driven Cache-Collision Attack [58]

```
Input: AES
Plaintext ← (0, 0, 0, ..., 0)
$i \leftarrow 1$
while $i < 16$ do
    CacheTrace ← AES(Plaintext)
    if CacheTrace[$i$] == Miss then
        $\widehat{\text{Plaintext}}[i]$++
    else
        $i$++
    end if
end while
Output: Plaintext=($p_0, p_1, p_2, \ldots, p_{15}$)
```

The expected number of plaintexts (traces) required for this algorithm to finish is 15 × 8.5 = 127.5. Below we show how to improve this figure,

3.3.2 Improved strategy

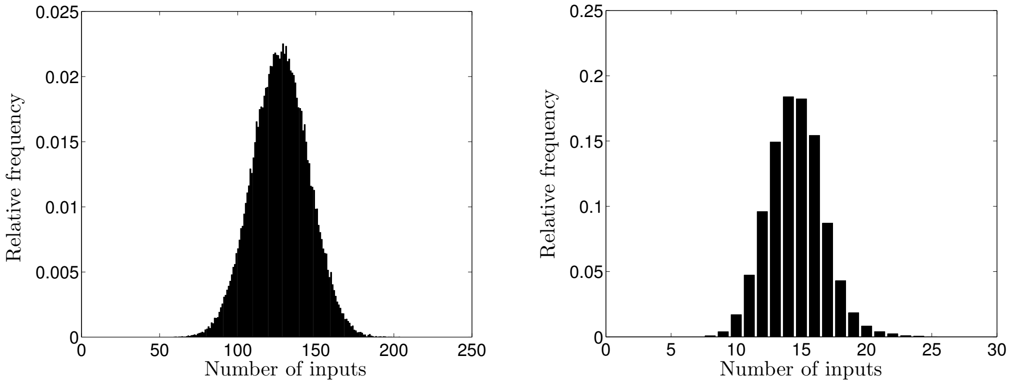
The attack strategy describe above does not optimally exploit the adaptive scenario. Namely, we observe that in Algorithm 3 ignored are the events located in the cache trace to the right of the current event (for which the plaintext nibble is being chosen). Below we show that by observing these events we can induce more constraints on the choice of the subsequent plaintexts drastically reduce the total number of plaintexts required to achieve the desired trace MHH…H.

We make use of the fact that a miss at position i, $0 < i < 16$, indicates that $\widehat{p_j \oplus k_j} \neq \widehat{p_i \oplus k_i}$ for all j such that $0 \leq j < i$ and event at position j is a miss. This means that any plaintext with the particular difference $\widehat{p_j \oplus p_i}$ between the nibbles in positions j and i will not lead to the desired trace MHH…H. So the plaintexts with this difference can be omitted from the subsequent queries. On the other hand, if there is a hit at position i, the plaintext nibble in this position may already be the one which we are searching for, i.e. satisfying $\widehat{p_0 \oplus k_0} = \widehat{p_i \oplus k_i}$. So we cannot do better than keeping it for the next query, changing it only if the event in position i becomes a miss in subsequent queries.

The formal description of our improved strategy is given in Algorithm 4. The algorithm terminates with a plaintext $(p_0, p_1, \ldots, p_{15})$ yielding the desired cache trace MHH…H and thus reducing the key search space by 60 bits. In the algorithm, SelectNextPlaintext is the routine choosing the next plaintext satisfying the given constraints. Namely, in every miss position j (except for the initial miss in position zero), this routine chooses a plaintext nibble satisfying all the constraints with j preceding nibbles, or, in case there is no such plaintext nibble, just the subsequent value of the nibble.

Algorithm 4: Improved Chosen Plaintext Trace-Driven Cache-Collision Attack

```
Input: AES
Plaintext ← $\{0\}_{16}$
Constraints ← $\{\emptyset\}_{16,16}$
$i \leftarrow 1$
while $i < 16$ do
    Plaintext ← SelectNextPlaintext($i$, Plaintext, Constraints)
    CacheTrace ← AES(Plaintext)
    for $j$ from $i$ to 15 do
        if CacheTrace[$j$] == Miss then
            for $k$ from 0 to $j - 1$ do
                if CacheTrace[$k$] == Miss then
                    Constraints[j][k] ⟵ Constraints[$j$][$k$] ∪ $\widehat{\text{Plaintext}}[j] \oplus \widehat{\text{Plaintext}}[k]$
                end if
            end for
        end if
    end for
    while (CacheTrace[$i$] == Hit) AND ($i < 16$) do
        $i$++
    end while
end while
Output: Plaintext=$\{p_l\}_{l=0}^{15}$
```



(a) Original chosen plaintext attack [58]

(b) Improved chosen plaintext attack

Figure 3.1: Distribution of the number of plaintexts required to obtain a 60-bit reduction of the key search space in a chosen plaintext attack

We have simulated this attack with 10^5 random keys both for the original Algorithm 3 describing the strategy of [58] and our improved Algorithm 4. The results are shown in Figure 3.1. The improvement is drastic: on average 14.5 plaintexts for our improved attack to obtain a 60-bit reduction against 127.5 for the original attack. These figures are for the case of absolutely reliable cache event detection. In Sect. 3.6 we will show that this improved algorithm is in fact inherently tolerant to uncertainties in cache event detection.

3.3.3 Drawback of the chosen plaintext strategy

To recover the remaining 68 bits of the key, the analysis of the second round lookups is performed in [58]. It also exploits chosen plaintext strategy, continuing to look for plaintexts leading to hits in the first two second round lookups. This strategy is quite inefficient, requiring 1280 traces on average to succeed with the recovery with the exhaustive search over the remaining 24 (and not 32 as stated in [58]) unknown bits. The inefficiency is due to picking the plaintexts from the pool of unused one at random. Unlike the first round stage, this cannot be improved: one cannot introduce an efficient way of inducing constraints on the plaintexts due to the non-linearity of the equations emerging from the second round lookups. So our improvement has an insignificant effect on the full attack measurement complexity. Therefore, we had to come up with a known plaintext attack which we will describe in the next Section.

3.4 Known plaintext attack

Here we present a trace-driven attack that works in the known plaintext scenario. That is, an attacker in possession of the target device implementing AES can submit plaintexts (or ciphertexts, since the attack will work for decryption as well) to it and observe the corresponding cache traces for the first AES round and 2 to 4 first lookups of the second AES round. Our attack strategy is similar to those already described in [30] and [58]. We describe the exact algorithm of the attack, including the second round stage. Our description is for the case of a common S-box lookup table of 256 bytes and cache line size of 16 bytes. We will show in Section 3.6 that our algorithm can be made tolerant to uncertainties in cache event detection.

The attack is composed of an online phase and an offline phase. In the online phase, the attacker makes the device encrypt a number of plaintexts and observes the side-channel leakage that reveals cache traces. In the offline phase, the cache traces are used to recover the secret key used by the device. The offline phase has two stages. The first stage deals with the first round cache events and leads to the recovery of 60 key bits. The second round stage recovers the remaining the bits (or part of them enabling exhaustive search for the remaining unknown bits) using the second round cache events.

3.4.1 Analysis of the first round

In this stage, we analyze the 16 cache events occurring in the first AES round. Our inputs are N plaintexts and the corresponding cache traces that are obtained from N acquisitions. In a q-th acquisition, a plaintext $P^{(q)}$ is a 16-byte array and a cache trace $(CT)^{(q)}$ is the array of the cache accesses observed in the first round of AES, while encrypting $P^{(q)}$ under the unknown key $K = (k_0, k_1, \dots, k_{15})$. The result of the analysis will be a set of linear equations in the high nibbles of k_i, $i \in \{0, 15\}$ that will decreases the entropy of the key search space by 60 bits.

We recall that cache events observed in a side-channel trace allow an attacker to determine whether at a certain lookup the S-box input belongs to a previously loaded line of the lookup table or not. A cache **miss** occurring at the i-th lookup can be algebraically expressed as the following inequation:

$$\forall j \in \Gamma,\ \widehat{k_i \oplus p_i} \neq \widehat{k_j \oplus p_j}$$

where Γ denotes the set of indices where previously occurred a cache miss. In a similar way a cache **hit** at the i-th lookup can be described with:

$$\exists! j \in \Gamma,\ \widehat{k_i \oplus p_i} = \widehat{k_j \oplus p_j}$$

Using the relation $\widehat{k_i \oplus k_j} = \widehat{k_i \oplus k_0} \oplus \widehat{k_0 \oplus k_j}$, the terms above can be rearranged as follows:

$$\forall j \in \Gamma,\ \widehat{k_i \oplus k_0} \neq \widehat{p_i \oplus p_j} \oplus \widehat{k_j \oplus k_0}$$

when the i-th cache event is a miss, and

$$\exists! j \in \Gamma,\ \widehat{k_i \oplus k_0} = \widehat{p_i \oplus p_j} \oplus \widehat{k_j \oplus k_0}$$

when it is a hit.

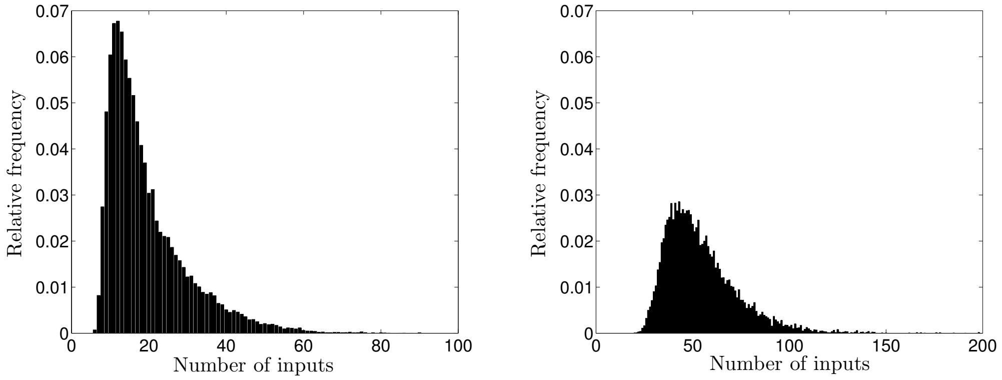
We aim to recover $\widehat{k_0 \oplus k_i}$, $1 \leq i \leq 15$, for which we define the inital set of possible values: $\boldsymbol{\kappa}_i = \{0, \dots, 15\}$. If we assume to know the values of $(\widehat{k_0 \oplus k_j})_{1 \leq j \leq i-1}$, the above equations and inequations resulting from cache events gradually reduce the number of possible values for $\widehat{k_0 \oplus k_i}$ from 16 to 1, i ranging from 1 to 15, allowing a recursive recovery of the values for all $\widehat{k_i \oplus k_0}$. Hence we analyze the traces lookup by lookup, instead of trace by trace. Finally, the high nibbles of the key bytes form a system of 15 linearly independent equations, reducing the entropy of the key down to $128 - 4 \times 15 = 68$ bits. The formal description of the first round analysis is given by Algorithm 5.

Algorithm 5: Known plaintext analysis of the first round

```
Input: $(P^{(q)}, CT^{(q)}),\ q \in [1, N]$
 1: $\boldsymbol{\kappa}_i \leftarrow \{0, \ldots, 15\},\ 1 \leq i \leq 15$
 2: for $i \leftarrow 1$ to 15 do
 3:   $q \leftarrow 0$
 4:   while $|\boldsymbol{\kappa}_i| > 1$ do
 5:     $q \leftarrow q + 1$
 6:     $\boldsymbol{\kappa}' \leftarrow \emptyset$
 7:     for $j \leftarrow 0$ to $i - 1$ do
 8:       if $CT_j^{(q)} = \text{Miss}$ then
 9:         $\boldsymbol{\kappa}' \leftarrow \boldsymbol{\kappa}' \cup \left\{\widehat{p_i^{(q)} \oplus p_j^{(q)}} \oplus \boldsymbol{\kappa}_j\right\}$
10:       end if
11:     end for
12:     if $CT_i^{(q)} = \text{Miss}$ then
13:       $\boldsymbol{\kappa}_i \leftarrow \boldsymbol{\kappa}_i \setminus \boldsymbol{\kappa}'$
14:     else
15:       $\boldsymbol{\kappa}_i \leftarrow \boldsymbol{\kappa}_i \cap \boldsymbol{\kappa}'$
16:     end if
17:   end while
18: end for
Output: $\boldsymbol{\kappa}_i,\ i \in [1, 15]$
```

To estimate the number of traces required to complete the first round analysis, we ran the implementation of Algorithm 5 in 10^5 simulated attacks, each with a random key. Figure 3.2(a) presents the results of this simulation. On average 19.43 acquisitions are required to reduce the entropy of the key to 68 bits. We stress that this is less than for the original *chosen plaintext* attack of [58] that we recalled here in Section 3.3.1.

Our strategy also works if we do not take into account the information available from the cache hits, that is, if $\boldsymbol{\kappa}_i$ is not modified when a cache hit is observed in a cache trace at the i-th lookup (line 15 of Algorithm 5). The resulting analysis is less efficient than the one of the misses and the hits, as depicted in the plotted distribution of the number of required inputs (Figure 3.2(b)). The average number of inputs required to perform a 60-bit reduction of the entropy of the key is in this case 54.19. However, it was shown in [30] using only misses enables the analysis



(a) Misses and hits

(b) Misses only

Figure 3.2: Distribution of the required number of plaintexts to obtain a 60-bit reduction of the key search space in the known plaintext attack

in the case the cache already contains S-box lookup table lines prior to encryption. We will recall this in Section 3.6.4.

3.4.2 Analysis of the second round

After the first round analysis we are left with 68 unknown key bits. In this Section we show how to recover these remaining bits by analyzing the cache events occurring during the second AES encryption round. Our approach in general resembles that of [58], however we exploit the cache events much more efficiently. Similar approach was briefly sketched in [2], they did not present the analysis for the number of traces required, whereas we perform theoretical analysis in Sect. 3.7. We assume that the round keys are pre-computed and pre-stored, thus no access to the S-box lookup table occurs between the encryption rounds[1].

The second round analysis re-uses the known inputs from the first round analysis (we assume that a 2-round cache trace is acquired for each input) and in most cases require some additional known inputs. It covers 4 lookups and finishes with a small exhaustive search, namely:

1. from the first lookup, recover $\widehat{k}_0$, $\check{k}_0$, $\check{k}_5$, $\check{k}_7$, $\check{k}_{10}$, $\check{k}_{15}$, 24 bits in total;
2. from the second lookup, recover $\check{k}_1$, $\check{k}_6$, $\check{k}_{11}$, $\check{k}_{12}$, 16 bits in total;
3. from the third lookup, recover $(\check{k}_2, \check{k}_8, \check{k}_{13})$, 12 bits in total;
4. from the fourth lookup, recover $(\check{k}_3, \check{k}_4, \check{k}_9, \check{k}_{14})$, 16 bits in total;

---

[1] Meanwhile, the strategy presented here would be straightforward to adapt to an AES implementation with an on-the-fly key schedule, and a similar strategy can be applied using `xtimes` operation of AES `MixColumns` transform in case the former is implemented as a lookup table (see [58] for using `xtimes` in a chosen plaintext attack).

5. at this point we can have at most 10 full key candidates, the correct one can be found by checking each against a known plaintext-ciphertext pair

The final multiplicity of full key candidates is due to the fact that analysis of some lookups may result in several candidates that are indistinguishable with any number of inputs. Note that one can launch exhaustive search already after the analysis of the second lookup, since at that point one is left with 2^{28} unknown key bits, so along with the possible multiplicity of candidates this leads to at most 2^{30} full key candidates to be tested. This is already practical, however the analysis of the subsequent lookups has much less computational complexity while requiring almost the same measurement complexity.

We describe our algorithm in detail for the first lookup of the second round, and briefly for the subsequent lookups since the technique is the same.

First lookup of the second round

The first lookup is indexed by

$$y_0 = 2 \bullet s(x_0) \oplus 3 \bullet s(x_5) \oplus s(x_{10}) \oplus s(x_{15}) \oplus s(k_7) \oplus k_0 \oplus 1.$$

The fact that this lookup is a miss, i.e. for a cache trace of the form M**…*|M…, leads to the following system of inequations:

$$\begin{cases} \widehat{y_0} \neq \widehat{x}_{j_1} \\ \dots \\ \widehat{y_0} \neq \widehat{x}_{j_L} \end{cases}, \quad j_1, \dots, j_L \in \Gamma,$$

where Γ is set of indices of misses observed in the 16 previous lookups (of the first round), $|\Gamma| = L$. After rearranging the terms the system becomes

$$2 \bullet s(x_0) \oplus 3 \bullet s(x_5\widehat{) \oplus s(x}_{10}) \oplus s(x_{15}) \oplus s(k_7) \neq \begin{cases} \widehat{\delta}_{j_1} \\ \dots \\ \widehat{\delta}_{j_L} \end{cases}, \quad j_1, \dots, j_L \in \Gamma, \tag{3.1}$$

where $\widehat{\delta}_j$ are some known values depending on the plaintext bytes and the XOR differences of key byte nibbles recovered in the first part of the analysis. Similarly, in case the first lookup is a hit, i.e. for a cache trace of the form M**…*|H…, we have

$$\left\langle \begin{array}{lcl} \widehat{y_0} & = & \widehat{x}_{j_1} \\ \dots & & \\ \widehat{y_0} & = & \widehat{x}_{j_L} \end{array} \right. , \quad j_1, \dots, j_L \in \Gamma$$

which after rearrangement becomes

$$2 \bullet s(x_0) \oplus 3 \bullet s(x_5\widehat{) \oplus s(x}_{10}) \oplus s(x_{15}) \oplus s(k_7) = \left\langle \begin{array}{l} \widehat{\delta}_{j_1} \\ \dots \\ \widehat{\delta}_{j_L} \end{array} \right. , \quad j_1, \dots, j_L \in \Gamma. \tag{3.2}$$

We have only 24 unknown bits in the left part of (3.1) or (3.2) since from the first round analysis we know the high nibble of the XOR difference between any two bytes of the key. Solving (3.1) or (3.2) for a single trace by exhaustive search over 2^{24} candidates for these bits will leave us with some fraction of these candidates. The next trace will result in a different set of equations either of the form (3.1) or of the form (3.2), and we can further reduce the amount of candidates.

After several traces, we will remain with the key bytes k_0, k_5, k_{10}, k_{15} completely recovered. However, up to 4 candidates will remain for k_7, since for a given input high nibble of the `SubByte` table, 1, 2, 3 or 4 input low nibbles may yield the same output high nibble. Hence having just $\widehat{s(k_7)}$ in the (in)equation it is not possible to distinguish between several possibilities for $\check{k}_7$. In the analysis of the subsequent lookups one has to consider this multiplicity, so in the end of the analysis one may end up with several full key candidates.

Second lookup of the second round

Once done with the analysis of the first lookup, we proceed to the second lookup of the second round. This lookup is indexed by

$$y_1 = 2 \bullet s(x_1) \oplus 3 \bullet s(x_6) \oplus s(x_{11}) \oplus s(x_{12}) \oplus s(k_7) \oplus k_0 \oplus k_1 \oplus 1$$

Similarly, the fact that this lookup is a miss leads to the following system of inequations (after rearranging the terms):

$$2 \bullet s(x_1) \oplus 3 \bullet \widehat{s(x_6) \oplus} s(x_{11}) \oplus s(x_{12}) \neq \begin{cases} \widehat{\delta}_{j_1} \\ \dots \\ \widehat{\delta}_{j_R} \end{cases}, \quad j_1, \dots, j_R \in \Gamma, \tag{3.3}$$

and if it is a hit, to the set of equations

$$2 \bullet s(x_1) \oplus 3 \bullet \widehat{s(x_6) \oplus} s(x_{11}) \oplus s(x_{12}) = \left\langle \begin{array}{l} \widehat{\delta}_{j_1} \\ \dots \\ \widehat{\delta}_{j_R} \end{array} \right., \quad j_1, \dots, j_R \in \Gamma, \tag{3.4}$$

where Γ is set of indices of misses observed in the 17 previous lookups (i.e. in the first round and in the first lookup of the second round), $|\Gamma| = R$, and $\widehat{\delta}_j$ are some known values depending on the plaintext bytes and the previously recovered nibbles of key bytes. The difference from the first lookup analysis is that one of the (in)equations in (3.3) or (3.4) may emerge from $\widehat{y}_1 \neq \widehat{y}_0$ (if the first lookup is a miss). From the analysis of the first lookup we already know y_0 and thus can consider this inequation here.

We have only 16 unknown bits in the left part of (3.3) or (3.4), namely the nibbles $\check{k}_1$, $\check{k}_6$, $\check{k}_{11}$ and $\check{k}_{12}$, the rest having been recovered in the previous steps. Solving the equations for several traces like in the analysis of the first lookup, we will get a single candidate for these unknown nibbles (for each of the candidates from the first lookup analysis, if they were multiple).

After the analysis of the first and second lookups of the second round the remaining unknown key chunks are $\breve{k}_2$, $\breve{k}_3$, $\breve{k}_4$, $\breve{k}_8$, $\breve{k}_9$, $\breve{k}_{13}$, $\breve{k}_{14}$. They comprise 28 bits. Considering up to 4 possible candidates for k_7, the number of full key candidates at this point is $2^{3}0$. So an exhaustive search can be already launched. But further analyzing the third and the fourth lookups of the second round leads to a faster attack, since a single evaluation of equation like 3.3 is much faster than the single AES run, and we will have to perform at most about 2^{16} evaluations.

Third and fourth lookups

The third lookup is indexed by

$$y_2 = 2 \bullet s(x_2) \oplus 3 \bullet s(x_7) \oplus s(x_8) \oplus s(x_{13}) \oplus s(k_7) \oplus k_0 \oplus k_1 \oplus k_2 \oplus 1$$

where the nibbles $(\breve{k}_2, \breve{k}_8, \breve{k}_{13})$, thus 12 bits, are unknown. Like for the first 2 lookups, an adversary can run through the 2^{12} possibilities for these nibbles and retain only the candidate that satisfies all the (in)equations and inequations derived from the previous lookups and plaintexts. These (in)equations may involve $\widehat{y}_0$ and $\widehat{y}_1$, which are known from the previous steps.

The fourth lookup is indexed by

$$y_3 = 2 \bullet s(x_3) \oplus 3 \bullet s(x_4) \oplus s(x_9) \oplus s(x_{14}) \oplus s(k_7) \oplus k_0 \oplus k_1 \oplus k_2 \oplus k_3 \oplus 1$$

where the unknown nibbles $(\breve{k}_3, \breve{k}_4, \breve{k}_9, \breve{k}_{14})$, 16 bits in total, are determined in the same manner.

Like the fist round analysis,the second round analysis can exploit only misses to be robust to the pre-loaded cache condition.

3.4.3 Attack complexity

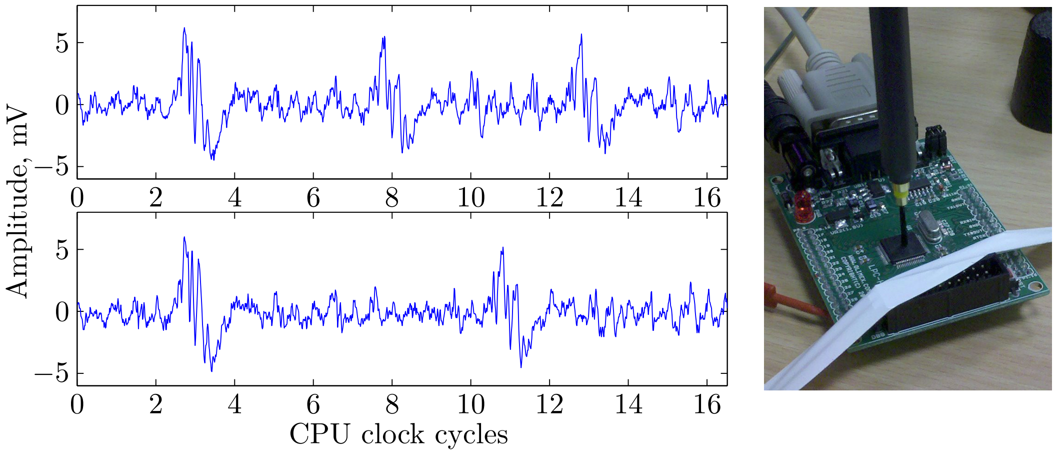
For the first round part, the computational complexity is negligible: it runs in seconds on a PC. The average measurement complexity of this part was already presented in Section 3.4.1: about 19 traces in case both hits and misses are used.

For the second round part, we have to perform at most $O(2^{24})$ evaluations of equations like 3.3. The final exhaustive search is negligible. This part runs in less than a minute on a PC. We mounted simulated attacks to determine average measurement complexity; it is about 29 traces (re-using the traces from the first round part) in case both hits and misses are used.

We will present more detailed measurement complexity data after introducing error-tolerant attack versions for a real-life noisy environment in Section 3.6. We will also present the theoretical model for determining the measurement complexity (Section 3.7), which is matched by our practical results.

3.5 Cache events in side-channel leakage

Here we describe our experiments with a μC and show that cache events can be distinguished in the EM leakage with a very simple measurement setup.



(a)

(b)

Figure 3.3: (a) EM traces of an ARM7 μC with distinguishable cache event sequences: miss-miss-miss (top) versus miss-hit-miss (bottom); (b) the μC with the passive EM probe

We have experimented with Olimex LPC-H2124 development board [112] (Figure 3.3(b)) carrying NXP LPC2124 [111], an ARM7 μC. Though ARM7 family devices do not normally have cache, this particular μC features a Memory Acceleration Module (MAM). The MAM is a single-line cache that increases the efficiency of accesses to the onboard flash memory. In Figure 3.3(a) we present the EM traces acquired while the μC with MAM enabled was performing a series of lookups in the AES S-box table that was stored in the flash memory. The acquisition was performed with Langer RF-B 0.3-3 H-field probe, Langer PA 203 20 dB pre-amplifier and LeCroy WaveMaster 104MXi oscilloscope. The CPU clock frequency of the μC was 59 MHz, the sampling rate of the oscilloscope was set to 5 GS/s. The probe was fixed by a lab stand with the probe tip touching the surface of the μC package; the precise position of the probe was determined experimentally.

The top trace shows a sequence of 3 cache misses, whereas the bottom trace shows a miss-hit-miss sequence. Cache misses can be seen as distinguished peaks. Note also the timing differences: Figure 3.3(a) suggests that the cache hit takes 2 CPU clock cycles less than the cache miss. We would like to stress that the traces were acquired *without any averaging in an unshielded setup* depicted in Figure 3.3(b).

Since MAM is a single-line cache, the full attack we describe in this work cannot be implemented with this μC. Still our experiments are a sound example of cache event leakage for an ARM μC, which has not been considered in earlier works except for [124], exploiting cache event leakage in the power consumption of a PowerPC processor within a Xilinx Virtex-II FPGA to mount an attack against CLEFIA.

3.6 Error-tolerant attack

In this Section, we will describe the full attack version resistant to uncertainties in cache event detection. We begin with describing a general approach to dealing with detection uncertainties, then explain how to make our attack algorithms deal with uncertainties and finally present the resulting attack measurement complexities for different uncertainty levels.

3.6.1 General approach to distinguishing cache events

Although our experiments have shown a very clear and strong leakage of cache events in EM traces, in general measurements can be contaminated with noise which can disturb the extraction of the cache sequence from the side-channel trace. Here we outline a way to deal with the noisy measurements in the case of cache event detection, that will allow us to make our attacks work even in a noisy setting.

We assume that we can measure some statistic in a side-channel trace like the height of the peak in the cycles corresponding to the table look-up, the value of the statistic being larger in case of a cache miss and smaller in case of a cache hit. We have shown above that this is a sound assumption which holds in practice. We further assume that the statistic for hits and misses will follow the distributions that are close to normal (due to the noise that is usually Gaussian). Distinguishing between hits and misses is then a task of distinguishing between the two distributions.

A simple distinguishing solution would be in fixing a single threshold for the value of the statistic, like in practical collision detection of [28]. This will result in an unavoidable trade-off between Type I and Type II errors. However, in our algorithms both taking a miss for a hit and a hit for a miss will often lead to the incorrect key recovery. Therefore, our approach is in fixing 2 thresholds t_H and t_M, $t_H \leq t_M$. In this setting, we distinguish between three types of events.

1. If the statistic is smaller than t_H we consider the event to be a hit.
2. If the statistic is larger than t_M, we consider the event to be a miss.
3. If the value of the statistic falls between the thresholds, we consider the event to be “uncertain”.

We assume that the thresholds t_H and t_M are chosen such that it is highly unlikely for a miss to be misinterpreted as a hit and vice versa. We denote the probability of the “uncertain event” by p, which we also refer to as error probability. Below we show how the additional “uncertain” category helps in making our algorithms resistant to errors when p is non-zero. Obviously, our error-tolerant attacks require more traces in order to succeed in the presence of errors.

3.6.2 Error-tolerant chosen plaintext attack

To make our attack of Sect. 3.3 error-tolerant, we keep the plaintext nibbles unchanged if the event is “uncertain” since we cannot do anything better than wait for another trace. This means that for the cache events following the currently analyzed cache event, the uncertainties are treated just as hits. For the currently analyzed cache event, where a hit leads to the desired equation and thus to proceeding to the next position, in case of an "uncertain" event we keep the current plaintext nibble and proceed with the next trace. In fact, Algorithm 4 does not need to be changed: it automatically implements the described strategy when the cache trace includes 3 event types: Miss, Hit, Uncertain.

We performed 10^4 simulated attacks with random keys in the presence of detection errors. The results show that our Algorithm 4 tolerates errors very well. For the error probability 0.2 it requires 22.6 traces on average, and for the the error probability 0.5 – 47.2 traces on average. Note that this is better than for the original adaptive algorithm of [58] without detection errors.

3.6.3 Error-tolerant known plaintext attack

Along with the cache hit H and the cache miss M, the **uncertain** cache event U integrates in our analysis. We show in detail how to adapt first and second round analysis. Results based on simulated attacks are provided in Section 3.6.5.

Adaptation of the first round

We consider the lookup i of the first round, eventually with uncertain events occurred before it. For this lookup, there are 3 possibilities.

1. If CT_i is a cache **hit**, then the eventual uncertain previous events of the trace are considered as misses, such that when shrinking the set $\boldsymbol{\kappa}_i$ to the set of values $\widehat{k_0 \oplus k_j}$ for $CT_j = M$, the correct value of $\widehat{k_0 \oplus k_i}$ is still included in $\boldsymbol{\kappa}_i$. Hence, the latter is reduced to the set $\{\widehat{k_0 \oplus k_j} \mid j \in \Gamma \cup \Upsilon\}$, where Γ and Υ denote the sets of indices where previously occurred respectively a cache miss and an uncertain event.

2. If CT_i is a cache **miss**, then the uncertain previous events of the trace are considered as hits, such that when evicting from $\boldsymbol{\kappa}_i$ the values $\widehat{k_0 \oplus k_j}$ for $CT_j = M$, the correct value of $\widehat{k_0 \oplus k_i}$ is not evicted. Hence, $\boldsymbol{\kappa}_i$ is defined as $\boldsymbol{\kappa}_i \setminus \{\widehat{k_0 \oplus k_j} \mid j \in \Gamma\}$.

3. If CT_i is an **uncertain** event, no action is performed on $\boldsymbol{\kappa}_i$.

This strategy is explicitly written in Algorithm 6.

Adaptation of the second round

The second round adapts itself to uncertain events similarly. Considering lookup i in the second round, $i = 0, 1, 2, 3$:

1. if CT_i is a cache **hit**, for every possible vector of the unknown nibbles involved in y_i, $\widehat{y}_i$ must belong to $\{k_0 \oplus k_{0,j} \oplus p_j \mid j \in \Gamma_1 \cup \Upsilon_1\} \cup \{\widehat{y}_j \mid j \in \Gamma_2 \cup \Upsilon_2\}$, where

Algorithm 6: Known plaintext analysis of the first round with uncertain cache events

```
Input: $(P^{(q)}, CT^{(q)}),\ q \in [1, N]$
$\boldsymbol{\kappa}_i \leftarrow \{0, \ldots, 15\},\ 1 \leq i \leq 15$
for $i \leftarrow 1$ to 15 do
    $q \leftarrow 0$
    while $|\boldsymbol{\kappa}_i| > 1$ do
        $q \leftarrow q + 1$
        $\boldsymbol{\kappa}', \boldsymbol{\kappa}^* \leftarrow \emptyset$
        for $j \leftarrow 0$ to $i - 1$ do
            if $CT_j^{(q)} = \text{Miss}$ then
                $\boldsymbol{\kappa}' \leftarrow \boldsymbol{\kappa}' \cup \left\{\widehat{p_i^{(q)} \oplus p_j^{(q)}} \oplus \boldsymbol{\kappa}_j\right\}$
            else if $CT_j^{(q)} = \text{Uncertain}$ then
                $\boldsymbol{\kappa}^* \leftarrow \boldsymbol{\kappa}^* \cup \left\{\widehat{p_i^{(q)} \oplus p_j^{(q)}} \oplus \boldsymbol{\kappa}_j\right\}$
            end if
        end for
        if $CT_i^{(q)} = \text{Miss}$ then
            $\boldsymbol{\kappa}_i \leftarrow \boldsymbol{\kappa}_i \setminus \boldsymbol{\kappa}'$
        else if $CT_i^{(q)} = \text{Hit}$ then
            $\boldsymbol{\kappa}_i \leftarrow \boldsymbol{\kappa}_i \cap (\boldsymbol{\kappa}' \cup \boldsymbol{\kappa}^*)$
        end if
    end while
end for
Output: $\boldsymbol{\kappa}_i,\ i \in [1, 15]$
```

Γ_r and Υ_r denote the sets of indices where previously occurred respectively a cache miss and an uncertain event in the round r, $r = 1, 2$.

2. if CT_i is a cache **miss**, for every possible vector of the unknown nibbles involved in y_i, $\widehat{y}_i$ must not belong to $\{k_0 \oplus k_{0,j} \oplus p_j \mid j \in \Gamma_1\} \cup \{\widehat{y}_j \mid j \in \Gamma_2\}$.

3. if CT_i is an **uncertain** event, no action is performed on the set of possible vectors of the unknown nibbles involved in y_i.

3.6.4 Partially preloaded cache

Our attacks can tolerate the setting when the cache already contains some S-box lines at the beginning of the first AES round in the way described in [30]. If the lookup table is partially loaded in the cache prior to the encryption, the cache trace will result in having more hits than one could have got with a clean cache.

Our adaptive Algorithm 4 straightforwardly tolerates this setting because it exploits only misses, and a partially preloaded cache means that some misses will not be observed. This does not lead to an incorrect key recovery since we will not exclude the correct key hypotheses from our set but only leave some additional incorrect key hypotheses.

In the case of our known plaintext attack, the claims from Sect. 3.4.1 when a hit occurs at the i-th lookup are no longer true: there does not necessarily exist an index $j \in \Gamma$ such that $\widehat{x}_i = \widehat{x}_j$. If one applies Algorithm 5 (or Algorithm 6) to such inputs, when $CT_i = \text{Hit}$, the set $\boldsymbol{\kappa}_i$ will be intersected with a set $\boldsymbol{\kappa}'$ possibly not containing the correct value for $\widehat{k_i \oplus k_0}$, thus evicting the latter from $\boldsymbol{\kappa}_i$. However, when $CT_i = \text{Miss}$, one can subtract $\boldsymbol{\kappa}'$ from $\boldsymbol{\kappa}_i$ because the former contains only incorrect values for $\widehat{k_0 \oplus k_i}$, although $\boldsymbol{\kappa}'$ may be smaller than if the cache did not contain any lines prior to the encryption.

This suggests that in order to avoid a failure in the key recovery, Algorithm 5 should be adapted to perform an action on $\boldsymbol{\kappa}_i$ only when misses occur, as mentioned in Section 3.4.1. The analysis of the second round can exploit misses only in the same manner to tolerate the partially preloaded cache. In case a noisy environment is combined with a partially pre-loaded cache, our solutions described in Sect. 3.6.3 and here are perfectly compatible, though requiring a higher number of inputs. The results showing the average measurement complexity for several cases of preloaded cache are below.

3.6.5 Practical results

We have conducted simulated attacks against AES-128 with different values for the probability p of uncertainty for a cache event and the number of preloaded lines in the cache for 10^4 random keys in every case. The results are depicted in Figure 3.4.

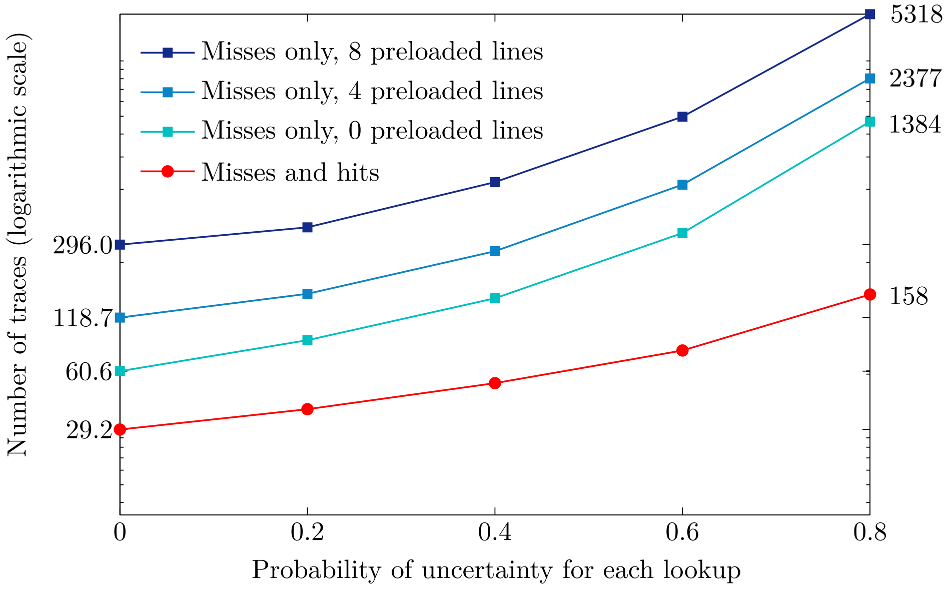


Figure 3.4: Average number of traces required for the full AES key recovery

One can see that our algorithm tolerates errors well: in case error probability is 0.8, one would need about 160 measurements to preform the full key recovery in case it is known that the cache is clean prior to each run of the algorithm. If the cache is not clean from the lookup table lines, then the numbers of traces (required for the misses-only analysis) are of course higher, but the attack is still feasible.

3.7 Theoretical model

In this Section we deal with theoretically estimating the number of traces required for the attack. In [2], a theoretical model was already presented to estimate the number of traces required for the analysis of the 15 cache events in the first round. This model however did not take into consideration the dependency between the cache events and assumed an error-free cache event detection. Here we first describe the sound model to obtain the number of traces required for the error-tolerant analysis of each of the cache events *individually*, both for the first and second round. This model takes detection errors into account. Then we show that this model, being univariate, still does not estimate the number of traces *for the full attack* precisely due to the statistical dependency between the cache events. The distribution in the multivariate model is however too complex to be expressed theoretically, so we present some illustrations based on experimental data.

3.7.1 Univariate model for the error-tolerant attack

The model we develop here will provide an expected number of traces $\mathbf{E}N_i$ required for the analysis of an i-th lookup, $1 \leq i \leq 15$ in the first round, $16 \leq i \leq 19$ in the second round (i.e. the enumeration of the cache events continues in the second round, to simplify writing the formulae), for a given error probability p, for the general case of m cache lines per S-box lookup table. We note that we will implicitly assume that the inputs to the second round lookups are statistically independent of the inputs to the first round lookups. Strictly speaking, this is not true. However, the statistical dependency in this case is not significant and so can be omitted for practical reasons; this is verified by the empirical results that we obtain running attack simulations.

We start with obtaining the expectation $\mathbf{E}R_i$ for the fraction of candidates R_i remaining after analyzing lookup i of a single cache trace. This expectation is expressed as

$$\mathbf{E}R_i = \sum_{s=1}^{i} \Pr(T_i = s) \cdot R_i^{(s)}, \quad i \geq 1 \tag{3.5}$$

where T_k is the number of lookup table lines in cache (i.e. $|\Gamma|$) after k lookups, and $R_i^{(s)}$ is the fraction of the key candidates remaining after analysis of the i-th lookup of a single cache trace when the number of lines previously loaded into cache is s. Note that (3.5) works for the second round lookups as well.

The distribution $\Pr(T_k = s)$ is a classical allocation problem [89]. We have

$$\Pr(T_k = s) = \binom{m}{m-s} \left(\frac{s}{m}\right)^k \Pr^0_k(s) \;, \tag{3.6}$$

where

$$\Pr^0_k(s) = \sum_{l=0}^{s} \binom{s}{l} (-1)^l \left(1 - \frac{l}{s}\right)^k .$$

Note that this distribution describes the process within the device and not the attacker's observations and therefore does not depend on the error probability p.

The fraction $R_i^{(s)}$ is expressed as the sum of the products of the conditional probabilities of the three possible cache event observations (miss, hit, uncertain) and corresponding remaining fractions $R_{i,M}^{(s)}$, $R_{i,H}^{(s)}$, $R_{i,U}^{(s)}$) (so, strictly speaking, this fraction is the expected value under a fixed s):

$$\begin{aligned} R_i^{(s)} &= \Pr(CT_i = M \mid T_i = s) \cdot R_{i,M}^{(s)} \\ &+ \Pr(CT_i = H \mid T_i = s) \cdot R_{i,H}^{(s)} \\ &+ \Pr(CT_i = U) \cdot R_{i,U}^{(s)} \;. \end{aligned} \tag{3.7}$$

Now, the three probabilities are

$$\begin{aligned} \Pr(CT_i = M \mid T_i = s) &= \tfrac{(1-p)(m-s)}{m} \;, \\ \Pr(CT_i = H \mid T_i = s) &= \tfrac{(1-p)s}{m} \;, \\ \Pr(CT_i = U) &= p \;. \end{aligned}$$

Finally, recalling the error-tolerant attack description in Section 3.6.3, the fractions for the cases of a miss, a hit and an uncertain event are

$$\begin{aligned} R_{i,M}^{(s)} &= \tfrac{m-(1-p)s}{m} \;, \\ R_{i,H}^{(s)} &= \tfrac{s+p(i-s)}{m} \;, \\ R_{i,U}^{(s)} &= 1 \;. \end{aligned}$$

Now, from equations (3.5), (3.6) and (3.7) we can obtain $\mathbf{E}R_i$ for $1 \le i \le 19$, i.e. both for the first and second round round lookups.

Knowing $\mathbf{E}R_i$, we can estimate the expected number of traces $\mathbf{E}N_i$ required for the analysis of an i-th lookup. We recall from Section 3.4.2 that in the analysis of each lookup in the first round ($1 \le i \le 15$), we want to reduce the number of candidates for the corresponding XOR difference of the key nibbles from m to 1. The traces are statistically independent if the inputs are independent (which is the assumption of our known plaintext attack), and each trace leaves us with a fraction $\mathbf{E}R_i$ of the remaining candidates, so we have

$$\begin{aligned} m \cdot (\mathbf{E}R_i)^{N_i} &\le 1 \;, \\ \mathbf{E}N_i &\approx -\log_{\mathbf{E}R_i} m \;. \end{aligned}$$

We obtain $\mathbf{E}R_i$ and estimate $\mathbf{E}N_i$ for all the lookups of the first round being analyzed assuming $m = 16$. The values for the case there are no errors in detection, i.e. $p = 0$, are shown in Table 3.1. Note that these values are larger than the ones obtained in [2] (denoted there by $R^k_{expected}$, k being i in our terms) since in our model we have correctly considered the dependency of the analysis of an i-th lookup on the events in the previous i lookups (recall that the enumeration of lookups starts from 0).

Table 3.1: Expected ratios of the remaining candidates and expected numbers of traces for the first round lookups, $m = 16$, $p = 0$.

i	1	2	3	4	5
$\mathbf{E}R_i$	0.882813	0.787598	0.711151	0.650698	0.603836
$\mathbf{E}N_i$	22.24	11.61	8.13	6.45	5.50

i	6	7	8	9	10
$\mathbf{E}R_i$	0.568490	0.542866	0.525418	0.514812	0.509903
$\mathbf{E}N_i$	4.91	4.54	4.31	4.18	4.12

i	11	12	13	14	15
$\mathbf{E}R_i$	0.509705	0.513373	0.520184	0.529519	0.540853
$\mathbf{E}N_i$	4.11	4.16	4.24	4.36	4.51

In Figure 3.5, we compare the theoretical estimates for the first round events with the empirical results that we obtained in attack simulations for the cases $p = 0$, $p = 0.25$ and $p = 0.5$. One can see that our model captures the behaviour of the attack and the effect of the errors.

Similarly, for the lookups of the second round ($16 \leq i \leq 19$) the numbers of traces can be estimated from the inequality

$$z_i \cdot (\mathbf{E}R_i)^{N_i} \leq 1 \,,$$

where z_i is the initial number of candidates in the system of (in)equations for the lookup i. From Section 3.4.2 we recall that $z_{16} = 2^{24}$, $z_{17} = 2^{16}$, $z_{18} = 2^{12}$ and $z_{19} = 2^{16}$. The theoretical estimates of $\mathbf{E}R_i$ and $\mathbf{E}N_i$ for the second round, for the case $m = 16$, $p = 0$, are given in Table 3.2. The numbers of traces observed in our experiments are correspondingly 26.86, 19.47, 15.02 and 20.36, thus perfectly matching the theoretical figures.

The model is also easily adjustable for the misses-only analysis in the case of partially preloaded cache.

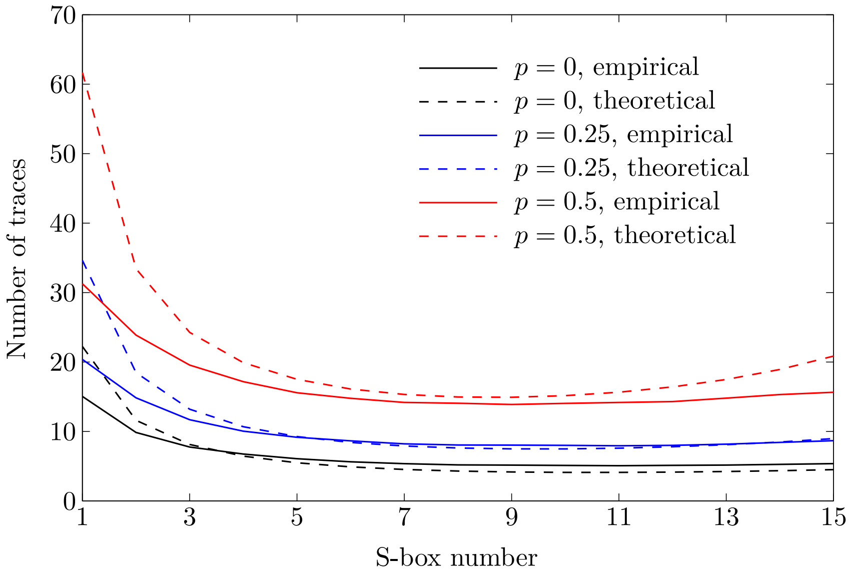


Figure 3.5: Expected number of traces for the lookups of the first round. Theoretical and empirical figures for different error probabilities p are shown.

Table 3.2: Theoretical expected ratios of the remaining candidates and expected numbers of traces for the second round lookups

Theoretical estimates of $\mathbf{E}R_i$ and $\mathbf{E}N_i$ for the second round lookups, $m = 16$,

	i	16	17	18	19
$p = 0.$	$\mathbf{E}R_i$	0.553737	0.567792	0.582699	0.598187
	$\mathbf{E}N_i$	28.15	19.59	15.40	21.58

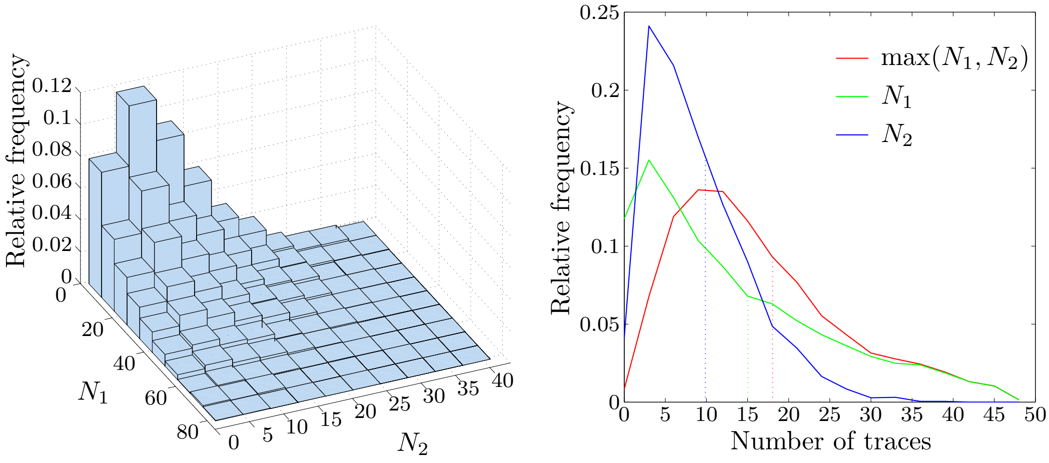
3.7.2 The multivariate model

The full attack measurement complexity is determined by the maximum number of traces required for the analysis of each lookup. If the cache events were statistically independent, the expectation for the maximum number of traces $\mathbf{E}(\max_i N_i)$ would be equal to the maximal expected number of traces among each of the lookups $\max_i(\mathbf{E}N_i)$, so one could use the model described above.

However, the cache events are dependent, therefore

$$\mathbf{E}(\max_i N_i) \neq \max_i(\mathbf{E}N_i) ,$$

and so the univariate model is not applicable for the estimation of the full attack complexity. This is well shown by the results of attack simulations: for example, in case $p = 0$ for the first round in the univariate model we observed $\max_i(\mathbf{E}N_i) = \mathbf{E}N_1 = 15$, whereas $\mathbf{E}(\max_i N_i) = 19$. The same holds for the second round, though not as explicit: the the observed average maximum number of traces for the *full*



(a) (b)

Figure 3.6: (a): empirical bivariate distribution for (N_1, N_2); (b): empirical univariate distributions for N_1, N_2, and for $\max(N_1, N_2)$, dashed lines showing the corresponding means.

analysis of the second round is 29, whereas the observed maximal number of traces for an *individual* lookup is 27.

So, to estimate the number of traces required for the full attack, one has to consider the distribution of $\max_i N_i$ for the case of statistically dependent random variables N_i, which requires the multivariate distribution for $(N_1, N_2, \ldots, N_{19})$. This multivariate distribution is hard to express analytically, therefore it is easier to simulate it carrying out the attack and sampling the values of $\max_i N_i$. This is what we do to obtain the results presented in Section 3.6.5.

To better illustrate the behaviour of the attack, in Figure 3.6(a) we present the empirical bivariate distribution for the case of the first round lookups 1 and 2. In Figure 3.6(b) we show the distribution for $\max(N_1, N_2)$ against the independent distributions for N_1 and N_2. The mean of the former is 18.04. One can see that it is greater than the maximum of the means for N_1 and N_2 and is actually quite close to the number of traces required for the analysis of all the 15 lookups in the first round.

Nevertheless, the univariate model provides an estimate for the lower bound for the number of traces required for the full attack, so it can be still applied when one requires this bound.

3.8 Conclusion

In this Chapter we described side-channel analysis that can be applied to implementations of AES on embedded devices featuring a cache mechanism. We improved the adaptive chosen plaintext attack described in [58] and presented a known plaintext attack that recovers a 128-bit AES secret key with approximately 30 measurements.

Our experiments with an ARM μC showed that cache events can be detected in the EM leakage. We have presented the versions of our attacks that can tolerate uncertainties in cache event detection from a side-channel trace that may occur in a noisy environment, as well as the setting when the cache is not clean of the S-box lookup table lines prior to the AES execution.

The attack measurement complexity for low error probabilities (which are quite realistic looking at our practical explorations in Section 3.5) is comparable to that of the DPA. The offline complexity is negligible: recovering the full key from a set of cache traces takes less than a minute on a standard PC. At the same time, cache-collision attacks are resistant to Boolean masking in the case where all S-boxes share the same random mask, as detailed in [58]. When such a masking scheme is used, our attacks will outperform higher order DPA attacks that typically require thousands of traces.

The countermeasures against trace-driven cache-collision attacks have been discussed in the previous works on the subject [16, 91, 58, 30] and are similar to the countermeasures against cache attacks in general [116]. They include pre-fetching the lookup table into the cache prior to encryption and shuffling the order of table lookup computations.

Chapter 4

Analysis of the AVR XMEGA cryptographic engine

AVR XMEGA is a recent general-purpose 8-bit μC from Atmel featuring symmetric crypto engines. In this Chapter we analyze the resistance of XMEGA crypto engines to side-channel attacks. We reveal the relatively strong side-channel leakage of the AES engine that enables full 128-bit AES secret key recovery in a matter of several minutes with a measurement setup cost about 1000 USD. 3000 power consumption traces are sufficient for our attack. Our analysis was performed without knowing the details of the crypto engine internals; quite the contrary, it reveals some details about the implementation. We sketch other feasible side-channel attacks on XMEGA and suggest the countermeasures that can raise the complexity of the attacks but not fully prevent them.

This is an independent work of the author that was published in [85].

Contents

4.1 Introduction

XMEGA [168] is the recent 8-bit RISC microcontroller (μC) in the Atmel's AVR family. To our knowledge, it is the first general-purpose μC available "over-the-counter" with AES [171] and DES [172] acceleration. Others μC's with cryptographic acceleration are either "prescription-only" secure versions (like Atmel's AT9xSC series) that are relatively hard to obtain for experiments, or dedicated wireless chips (for example, Ember EM250 or Jennic JN5121 ZigBee SoC's, or Chip-Con CC2420 RF transceiver used in MICAz motes). The AES engine of XMEGA provides 128-bit encryption and can maintain a throughput of about 10 Mbps. Apart from the cryptographic features, XMEGA boasts a broad set of peripherals and functionalities including DMA controller, event system for CPU- and DMA-independent inter-peripheral communication, advanced clocking and energy saving options. Along with the low power consumption and an attractive price of less than 10 USD apiece, this makes XMEGA a promising device for a wide range of embedded applications.

Motivation *De jure*, XMEGA is not claimed by Atmel to be a microcontroller for secure applications. *De facto* however, the presence of the cryptographic features suggests the contrary. A scenario where an embedded device implementing cryptography is subject to implementation attacks, in particular, attacks exploiting side-channel leakage of the device, is very realistic.

The typical scenario of a side-channel attack is when the secret key is stored within the device. This secret key is used by the device to perform cryptographic operations but never exposed through the communication channels of the device. The attacker can try to deduce the secret key by analyzing physical parameters of the device during an execution like timing of operations, power consumption and electromagnetic radiation, since these side channels may leak information about the internal variables processed by the device. side-channel attacks are posing a real threat for unprotected devices since they can be mounted by an attacker with a modest budget.

The XMEGA symmetric crypto engines can be of course used in non-keyed constructions like block-cipher based hash functions. In this scenario side-channel attacks are irrelevant. But (a) there are many more keyed scenarii and (b) there are no widespread AES and DES-based hash functions that could be used. Also, AES in XMEGA is implemented as an atomic full encryption/decryption, which means that hash functions that are based on AES building blocks (like some of SHA-3 candidates) will not benefit from this acceleration.

Related Work To date, we are not aware of any works studying XMEGA implementation security issues. In [134] and [135], XMEGA crypto engines were utilized for acceleration of AES- and DES-based hash functions. Same application was mentioned in [133]. In [52], XMEGA DES engine was used for the fast implementation of a cryptographic pseudo random number generator.

Side-channel attacks on hardware implementations of AES were performed in several works; see [113], [10] and [155] for examples.

Our Contribution The work presented here is very practical. We do not develop new attacks but apply the more or less standard attack to the implementation without any detailed knowledge of the internals of this implementation. We look at XMEGA cryptographic features from the side-channel attacker's point of view and highlight their weaknesses in the side-channel scenario. Then, experimentally we reveal the relatively strong side-channel leakage of the XMEGA AES engine. Exploiting this leakage in a practical power analysis attack, we recover the full 128-bit AES secret key with only 3000 acquisitions and post-processing time of several minutes. Acquisition can be performed with quite modest (around 1000 USD) measurement setup without any noise reduction measures.

With our attack we can reveal some details about the implementation. Namely, we discover that the AES implementation is non-parallelized. We suggest the possible countermeasures that can be used to partially hinder the attacks.

Semi-invasive and invasive attacks like fault analysis are beyond the scope of this work. However, we believe that faults can be induced quite efficiently on this μC.

4.2 XMEGA cryptographic features

We begin with a description of XMEGA cryptographic functionality, following the preliminary manual [168].

4.2.1 DES instruction

DES is implemented in XMEGA as an instruction set extension. A CPU instruction `DES K` executes a single DES round and operates with the state and the key loaded in the main register file. The round number `K` is passed to the instruction as a parameter. Direction is controlled by the half-carry flag of the μC's status register. Encryption/decryption is done as follows:

1. the 64-bit plaintext/ciphertext is placed in CPU registers `R0-R7`;
2. the full 64-bit key (including the parity bits) is placed in CPU registers `R8-R15`;
3. the CPU half-carry flag is set to 0 for encryption or to 1 for decryption;
4. DES instruction is executed 16 times with increasing `K` (`DES 0`, `DES 1`, ..., `DES 15`);

The 64-bit result (cihertext/plaintext) will be in the registers `R0-R7`. Intermediate states in `R0-R7` differ from the FIPS standard [172] since the initial permutation and the inverse initial permutations are performed each iteration [170].

The instruction takes 1 CPU clock cycle. Another CPU clock cycle is added if the instruction is followed by a non-DES instruction. Thus, the full DES execution (not counting data and key loading/unloading) takes 17 clock cycles.

4.2.2 AES peripheral

XMEGA implements AES-128 [171], a variant of AES with 128-bit secret key and 10 rounds. Unlike DES, AES engine in XMEGA is interfaced through 5 dedicated registers in the I/O address space. Encryption/decryption is performed as follows:

1. set encryption direction and other parameters (see below) in AES CTRL register, enable/disable AES interrupts in AES INTCTRL register;
2. the 16 key bytes are sent one-by-one to the AES key memory through the AES KEY register;
3. the 16 plaintext/ciphertext bytes are sent one-by-one to the AES state memory through the AES STATE register;
4. the start bit in the control register is set to start AES execution (this step can be omitted if the auto start flag was set in the status register).

When the execution is finished, the interrupt flag in the AES STATUS register is set and AES interrupt is optionally generated. The result can be read out byte-by-byte from the AES STATE register. If the XOR flag in the AES CTRL register is set, the data loaded in the AES STATE register is XOR'ed with the current AES state. This allows to implement CBC and other modes of operation efficiently.

After the execution the AES key memory will contain the last round subkey, so for another encryption the key should be reloaded. For decryption, the last round subkey should be loaded into the key register. It can be obtained either by running AES key expansion in software or by running a single AES engine encryption with a dummy state and the original key.

AES execution takes 375 peripheral[1] clock cycles not counting data and key loading/unloading (*cf.* about 3 thousand CPU cycles for a software implementation [119]). The CPU can execute any code in parallel. Combined with the DMA transfer (that relieves the CPU from data and key loading/unloading) and event system triggering at the end of AES execution, this definitely makes AES implementation in XMEGA quite efficient and attractive for embedded developers: when operated at its maximum clock frequency of 32 MHz, XMEGA can maintain AES-128 encrypted communication bandwidth of about 10 Mbps.

In the following Sections we will show that crypto engines of XMEGA are susceptible to efficient side-channel attacks.

---

[1] The peripheral clock frequency is the same as of the CPU clock; this is not the fast peripheral clock used by some of the XMEGA modules.

4.3 Security model and potential weaknesses

We consider a simple security model when an attacker aims at recovering the secret key stored within the device. An attacker knows inputs or corresponding encrypted or decrypted outputs (known plaintext or ciphertext scenario) of the cryptographic module and can observe physical parameters of the device while it performs cryptographic operations. This model is quite realistic when there is a chance for an embedded device to fall for a while into the hands of a technically prepared malefactor.

Resistance of XMEGA to implementation attacks was not claimed by Atmel: our research does not result in any statements regarding implementation security either in XMEGA documentation [168] or in other publicly available sources. However a scenario where an embedded device implementing cryptography is subject to implementation attacks is quite probable. One may therefore think that some countermeasures that are declared for AT9xSC secure series of μC's [169] were implemented in XMEGA as well.

In case there are no countermeasures implemented, we see the following potential weaknesses of XMEGA in the model defined above:

- AES and DES: the crypto engines themselves can exhibit side-channel leakage;
- AES and DES: direct manipulation of the secret key bytes by the AVR core is required, in particular transferring the secret key bytes over the memory buses;
- DES: intermediate state is stored in the main register file;
- AES: key expansion for decryption can be done manually in software.

Therefore, the threats for AES and DES modules are as follows:

- side-channel attacks on the secret key bytes manipulated by the AVR core;
- side-channel attacks on the cryptographic engine execution.

In the next Section, we show that the AES engine exhibits significant side-channel leakage (which suggests that no *sound* countermeasures were foreseen in XMEGA) and present an efficient practical full key recovery side-channel attack on the AES engine.

4.4 Practical attack

Here we present our practical side-channel attack on the XMEGA AES engine, recovering full key in a matter of several minutes of acquisition and post-processing. Our attack is based on the standard Differential Power Analysis (DPA) technique, but the attack flow for the full key recovery is specific to our particular case. We refer the reader to the book [101] for the basics of the differential power analysis. We would like to stress that we discovered the attack without any prior knowledge about the internals of the AES implementation in XMEGA.

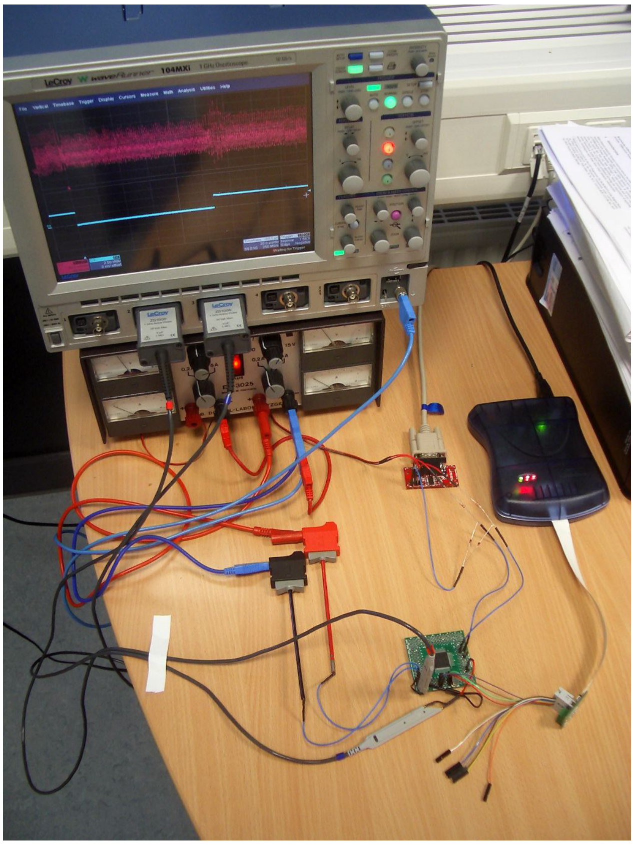


Figure 4.1: Measurement setup around ATXMEGA128A1

4.4.1 Measurement setup

We experimented with an ATXMEGA128A1-AU μC [168]. The μC was clocked at 3.6864 MHz (this is both CPU and peripheral clock frequency) using a quartz crystal and was provided with 3.6 V from a standard laboratory power supply. Connection to the host PC was performed via serial interface through the ADM202E level converter.

A 50 Ohm shunt resistor was inserted in the ground line of the μC for measuring the power consumption. Measurements were performed with the LeCroy WaveRunner 104MXi DSO equipped with ZS1000 active probes and connected to the host PC via a 100 Mbit Ethernet link. One of the pins of the μC was dedicated to the trigger signal. The setup did not include any specific measures for noise reduction. Figure 4.1 depicts the setup.

To verify the setup, we have first mounted a CPA attack in the Hamming weight leakage model [31] on an unprotected software implementation of AES [119] on XMEGA. Then we compared its results with those obtained for the same software implementation on the ATmega16 μC in the same measurement conditions. The results are consistent: for the software AES-128 implementation, around 50 acquisitions (*i.e.* random plaintexts) are required in our setup for the successful full key recovery both on ATmega16 and ATXMEGA128A1.

The acquisition parameters in the attack on the software implementation were as follows:

- analog low-pass filtering of the signal with 20 MHz cut-off frequency;
- sampling rate 100 MS/s;
- post-processing: power consumption curves were compressed to 1 sample per μC's CPU clock cycle by selecting cycle maxima.

We decided to start attacking the hardware AES implementation on XMEGA keeping these acquisition parameters.

4.4.2 Implementation details

We have implemented AES-128 encryption utilizing the AES crypto engine of the XMEGA. The encryption key was stored in the the program flash memory and loaded into the AES module. Then the plaintext bytes were obtained from the host PC via the serial line and loaded into the AES module. Then the encryption was started and the core entered a loop waiting for encryption to finish. When encryption was finished, the ciphertext was read from the AES module and returned to the host PC over the serial line.

We have acquired power consumption traces of the full encryption (375 clock cycles) for 10000 random plaintexts with the same acquisition parameters as for the attack on the software implementation.

4.4.3 The attack

Here we will describe our attack on the XMEGA AES peripheral. The attack consists of a basic step which is a more or less standard CPA attack recovering one key byte, and the algorithm for the full key recovery that is simple but specific to our particular case.

Basic attack

First, we have tried exactly the same CPA attack in the Hamming weight model as for the software implementation described above. No significant correlation was observed with 10000 traces for most of the bytes[2]. Knowing that hardware implementations typically follow the Hamming distance model (see [113] and [10] for examples of attacks on ASIC implementations of AES), we tried several combinations of various internal bytes that did not show any sensible correlation until we came to the following target value in the attack:

$$b_i = (p_i \oplus k_i) \oplus (p_{i+1} \oplus k_{i+1}), \quad 1 \leq i \leq 15 \ ,$$

---

[2] only bytes 2 and 12 exhibited significant leakage

where p_i and k_i are corresponding plaintext and key bytes counted in a standard way (column-wise). So by taking the Hamming weight of b_i, $HW(b_i)$, we get the Hamming distance between the successive bytes of the state after the first `AddRoundKey` transform of AES.

A straightforward approach would require to compute $2^8 \times 2^8 = 2^{16}$ correlation traces for all possible pairs of the two key bytes k_i and k_{i+1} involved. To reduce the number of correlation traces, we have assumed byte k_i to be known (like in [10]) and calculated correlation traces for 256 guesses of k_{i+1} (below we will show that this approach leads to a very efficient full key recovery attack). Since we targeted the operations in the 1st round of AES, we decided to consider only first 80 cycles of encryption and this turned out to be sufficient. Figure 4.2 shows the correlation coefficient ρ against clock cycles of AES encryption for the pairs of bytes k_1, k_2 and k_4, k_5. Correlation traces for the guesses corresponding to the correct 1st-2nd and 4th-5th subkey bytes correspondingly are highlighted.

We can clearly see up to three distinct (in time) correlation peaks. The first peak is for the key byte k_i (the one we assume to be known): the correlation trace highlighted in black corresponds to $k_{i+1} = k_i$ and and exhibits the maximal correlation in the region of the peak. The 2nd peak reveals the second subkey byte k_{i+1} (the one we are guessing): the correlation trace highlighted in red corresponds to the correct guess of k_{i+1} and shows maximal correlation among all traces in the region of the peak. By checking all 15 possible byte pairs we discovered that the 3-rd peak also revealing the correct guess of k_{i+1} is present for byte pairs 1-2 to 3-4 and 9-10 to 11-12, as shown in Figure 4.2(a). This 3-rd peak can be used to increase the reliability of the attack.

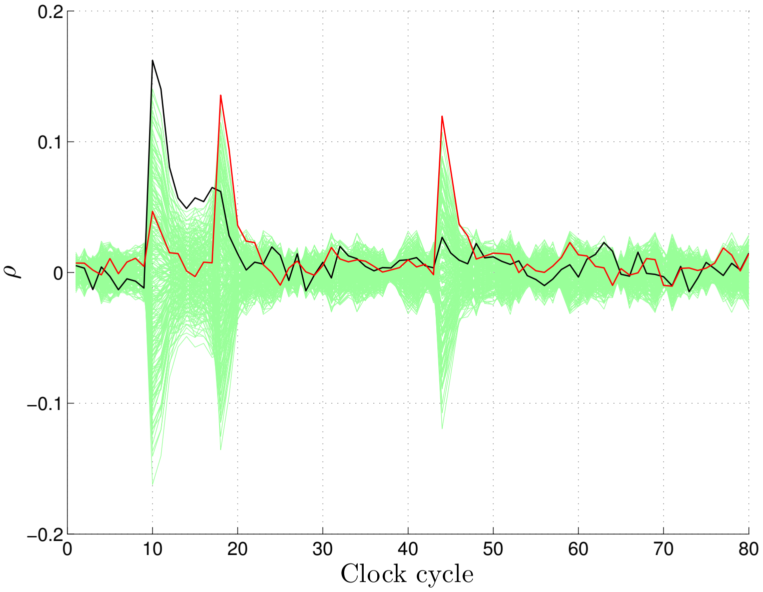
Such distinct pattern emerges for all $i = 1, 2, \ldots, 15$, that is, for all successive pairs of adjacent state bytes after the initial key addition. The evolution of the correlation coefficients in the region of the 2-nd peak with the increase of the number of available power consumption traces is shown in Figure 4.3. We can see that 2-3 thousand traces is already sufficient to recover the correct (relative to the 1st byte) guess of the 2nd key byte. For other byte pairs these figures are the same.

Full key recovery

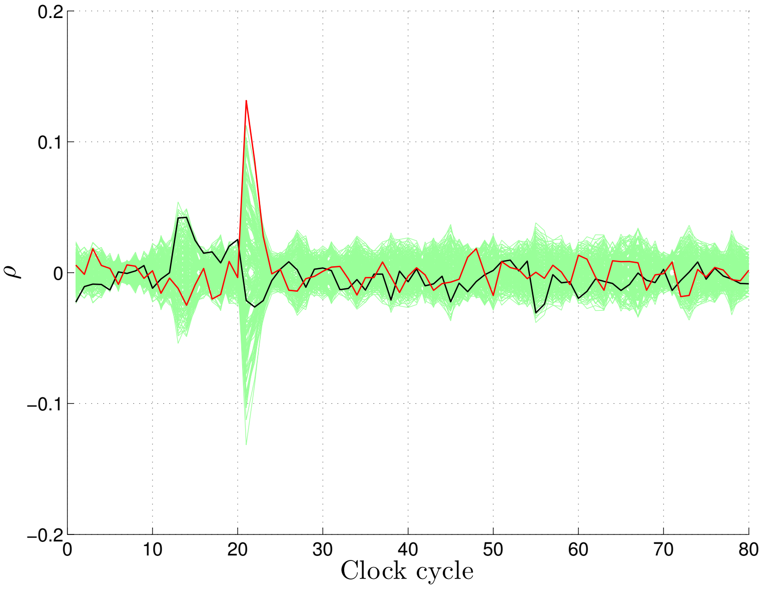
Above we have presented the basic CPA attack recovering the single key byte under the assumption that the previous key byte is known. Here we present the algorithm for the full 128-bit key based on this attack. The algorithm is straightforward:

1. guess the first key byte k_1 (256 guesses in total);

2. determine k_2 with the basic CPA attack as described above: with our guess of k_1 calculate the correlation traces for all 256 guesses of k_2 and obtain the correct guess by choosing the trace that exhibits the maximal correlation value in the region of the second peak; note that the peaks can be identified quite easily since we know the trace corresponding to the guess of k_1 and it shows its maximum in the region of the first peak;



(a)



(b)

Figure 4.2: Results of CPA in HD model on XMEGA AES engine, for byte pairs 1-2 (a) and 4-5 (b)

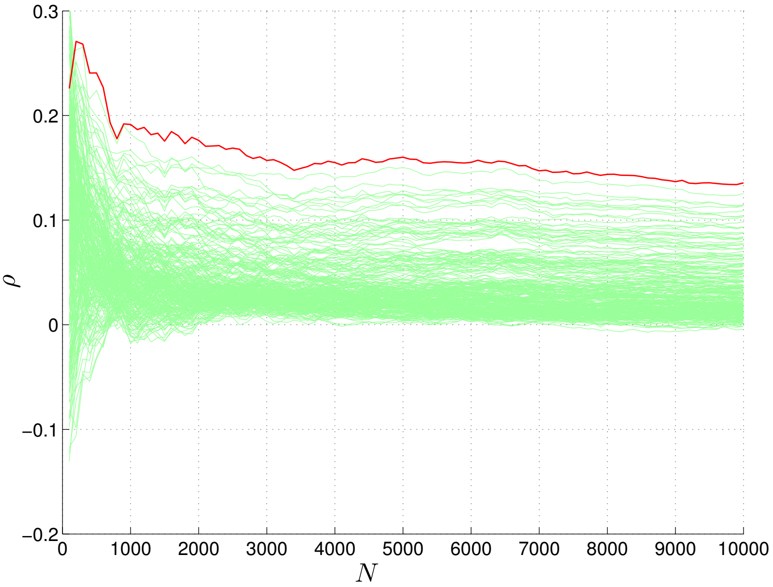


Figure 4.3: Evolution of the correlation coefficient in the region of the 2nd CPA peak against the number of traces, for bytes 1 and 2. Correlation trace for the correct 2nd subkey byte guess is highlighted.

3. determine $k_3, k_4, \ldots, k_{156}$ sequentially in the same way: knowing k_2, recover k_3 and so on;
4. having ended up with 256 full key candidates, find the correct one among them by an exhaustive search; note that this requires at least one known plaintext-ciphertext pair from the attacked device.

As already shown (see Fig. 4.3), the attack requires less than 3000 traces in our setup; with our acquisition rate of 15 traces per second this *on-line* phase takes about 3 minutes. Since compressed traces are used, the *off-line* stage of the attack (obtaining correlation curves, determining key candidates and running exhaustive search) takes about a minute. The relatively low sampling rate allows one to use an inexpensive DSO. Thus, the whole attack takes several minutes and requires 1000 USDs' worth of equipment.

4.4.4 Insight into AES hardware

Our attack was performed without any knowledge about the internals of the AES implementation. However, the results of the attack reveal some information about the AES implementation in XMEGA.

For example, in Figure 4.2 we can see that the correlation peaks for byte pair 4-5 occur later than the ones for the byte pair 1-2. The observation holds for all byte pairs. We conclude that operations are executed sequentially and not in parallel as one would expect from a hardware AES implementation. This is further supported by the fact that the similar behaviour of the correlation peaks were observed for all successive byte pairs, which suggests that all the 16 state bytes are processed sequentially.

Another example is the observed third correlation peak for some byte pairs that indicates re-use of these values, however we have not figured out the possible reason.

Refer to [37] for details on SCARE—side-channel analysis for reverse engineering. Such information can be in particular helpful for mounting fault attacks since it can indicate where to inject a fault.

4.5 Discussion and countermeasures

In the previous Section we have presented a practical attack on the XMEGA AES module recovering the full 128-bit key from 3000 acquisitions. While being almost 100 times larger than for an unprotected software implementation, this number is still so small that the whole attack takes several minutes.

We have not performed experimental investigation of the DES engine since we believe that AES engine will be used in most of XMEGA applications: XMEGA is highly suitable for wireless applications like ZigBee that imply AES usage. We suppose that DES engine does not include any side-channel attack countermeasures as well, though attacking it might require more traces due to the higher level of parallelism in DES implementation (17 cycles per full DES vs. 375 for full AES).

4.5.1 Attack practicability

Those familiar with embedded development might argue that in our attack we had several conditions that simplify it, namely:

1. perfect synchronization via the dedicated trigger signal in the beginning of encryption;
2. the CPU core running a deterministic loop during encryption;
3. the chip was clocked at only 3.7 MHz (and using a stable quartz oscillator) while it allows a maximum of 32 MHz.

Regarding synchronization, it can be often done in a real application by observing the transfer over the communication line that is being encrypted. What concerns the second condition, it is quite probable that even in a real application the CPU core will execute some deterministic sequence of operations while AES engine is running or will even be in the idle sleep mode (it is possible in XMEGA to stop the CPU clock while all peripherals are kept running).

With the higher clock frequency the amount of traces required for the successful attack should increase since the power consumption observations in consecutive clock cycles will be superposed due to the unchanging discharge time of the internal capacitances of the chip. However, for a low-power operation XMEGA can only be clocked at low frequencies (less than 16 MHz), so a relatively cheap DSO could still be used for acquisition.

4.5.2 Countermeasures

As a countermeasure against side-channel attacks both for AES and DES engines, we suggest deliberate aggravation of all the 3 conditions mentioned above.

Synchronization can be made difficult for an adversary by adding software random delays (see Chapter 6) prior and after execution, and for DES also between rounds. Using less stable internal oscillators of XMEGA as the clock source can be considered; however alone they seem to be still stable enough for acquisitions of several hundreds clock cycles. Finally, clocking at higher frequency will increase the number of traces required for the successful attack.

For AES, these can be combined with hiding in the amplitude domain by making the CPU core manipulate randomized data or even execute a randomized sequence of instructions while encryption/decryption is running, for example, running a software pseudo random number generator.

More efficient countermeasures are hardly possible for the crypto engines of XMEGA since the low granularity of the implementation (the entire execution for AES and a single round for DES) seems not to allow for any other software countermeasures like masking or shuffling, and sound hardware countermeasures are absent. However it is an interesting research question whether the round-level granularity of DES in XMEGA can be used to implement any efficient software countermeasure. See also [103] for an approach to protocol-level countermeasures against differential side-channel attacks that can be applicable in some cases.

Another weak link in our view is the direct manipulation of the secret key bytes by the CPU core that can be susceptible to template attacks (refer to Section 5.3 of [101]). These operations can be also protected by desynchronization. Unfortunately, shuffling of the AES key byte loading order cannot be used since key bytes should be loaded sequentially, so only random delays are applicable here.

We stress that the suggested countermeasures only raise the complexity of attacks while lowering performance, but do not render them completely impossible.

4.6 Conclusion

In this Chapter we have presented an efficient side-channel attack against the cryptographic module of AVR XMEGA, a recent general-purpose 8-bit microcontroller. Our practical side-channel attack against the AES engine recovers the full 128-bit secret key from 3000 power consumption traces that can be acquired with a very modest measurement setup. The attack results indicate that the AES implementation in XMEGA is not parallelized. We have pinpointed potential weaknesses of XMEGA allowing other efficient side-channel attacks. We have suggested the possible countermeasures that can be implemented to raise the complexity of attacks. However, these countermeasures do not prevent the attacks completely and decrease performance.

Therefore, we conclude that utmost care should be taken when using XMEGA in embedded applications dealing with security, *i.e.* implementing suggested countermeasures and using it only in those applications where the presented attacks are tolerated by the security level.

Our result demonstrates how easy it is to attack an unprotected device with more or less standard techniques even without knowing the details of the actual implementation of the cryptographic algorithm. Attacking a 10 USD device for 1000 USD may seem ridiculous but it is in fact not—attacking 100 devices repays the setup and a potential application may be worth much more.

Chapter 5

Fault attacks on RSA signatures with partially unknown messages

Fault attacks exploit hardware malfunctions to recover secrets from embedded electronic devices. In the late 90's, Boneh, DeMillo, and Lipton [29] introduced fault-based attacks on CRT-RSA. These attacks factor the signer's modulus when the message padding function is deterministic. However, the attack does not apply when the message is partially unknown, for example when it contains some randomness which is recovered only when verifying a *correct* signature.

In this Chapter we successfully extend RSA fault attacks to a large class of partially known message configurations. The new attacks rely on Coppersmith's algorithm for finding small roots of multivariate polynomial equations. We illustrate the approach by successfully attacking several randomized versions of the ISO/IEC 9796-2 encoding standard. Practical experiments show that a 2048-bit modulus can be factored in less than a minute given one faulty signature containing 160 random bits and an unknown 160-bit message digest.

This is a joint work with Jean-Sébastien Coron, Antoine Joux, David Naccache, and Pascal Paillier, published in [43].

Contents

5.1 Introduction

5.1.1 Background

RSA [132] is undoubtedly the most common digital signature scheme used in embedded security tokens. To sign a message m with RSA, the signer applies an encoding (padding) function μ to m, and then computes the signature $\sigma = \mu(m)^d \bmod N$. To verify the signature, the receiver checks that

$$\sigma^e = \mu(m) \mod N.$$

As shown by Boneh, DeMillo and Lipton [29] and others (*e.g.* [77]), RSA implementations can be vulnerable to fault attacks, especially when the Chinese remainder theorem (CRT) is used; in this case the device computes

$$\sigma_p = \mu(m)^d \mod p\ , \qquad \sigma_q = \mu(m)^d \mod q$$

and the signature σ is computed from σ_p and σ_q by Chinese remaindering.

Assuming that the attacker is able to induce a fault when σ_q is computed while keeping the computation of σ_p correct, one gets

$$\sigma_p = \mu(m)^d \mod p\ , \qquad \sigma_q \neq \mu(m)^d \mod q$$

and the resulting (faulty) signature σ satisfies

$$\sigma^e = \mu(m) \mod p\ , \qquad \sigma^e \neq \mu(m) \mod q\ .$$

Therefore, given one faulty σ, the attacker can factor N by computing

$$\gcd(\sigma^e - \mu(m) \bmod N, N) = p\ . \tag{5.1}$$

Boneh *et al.*'s fault attack is easily extended to any deterministic RSA encoding, *e.g.* the Full Domain Hash (FDH) [14] encoding where

$$\sigma = H(m)^d \mod N$$

and $H : \{0,1\}^* \mapsto \mathbb{Z}_N$ is a hash function. The attack is also applicable to probabilistic signature schemes where the randomizer used to generate the signature is sent along with the signature, *e.g.* as in the Probabilistic Full Domain Hash (PFDH) encoding [42] where the signature is $\sigma \| r$ with $\sigma = H(m \,\|\, r)^d \bmod N$. In that case, given the faulty value of σ and knowing r, the attacker can still factor N by computing

$$\gcd(\sigma^e - H(m \,\|\, r) \bmod N, N) = p.$$

5.1.2 Partially-known messages: the fault-attacker's deadlock

However, if the message is not entirely given to the opponent the attack is thwarted, *e.g.* this may occur when the signature has the form $\sigma = (m\|r)^d \bmod N$ where r is a random nonce. Here the verifier can recover r only after completing the verification process; however r can only be recovered when verifying a *correct* signature. Given a faulty signature, the attacker cannot retrieve r nor infer $(m\|r)$ which would be necessary to compute

$$\gcd(\sigma^e - (m\|r) \bmod N, N) = p.$$

In other words, the attacker faces an apparent deadlock: recovering the r used in the faulty signature σ seems to require that σ is a correctly verifiable signature. Yet, obviously, a correct signature does not factor N. These conflicting constraints cannot be conciliated unless r is short enough to be guessed by exhaustive search. Inducing faults in many signatures does not help either since different r values are used in successive signatures (even if m remains invariant). As a result, randomized RSA encoding schemes are usually considered to be inherently immune against fault attacks.

5.1.3 The new result

We overcome this apparent deadlock by showing how to extract *in some cases* the $\underline{\text{u}}$nknown $\underline{\text{m}}$essage $\underline{\text{p}}$art (UMP) involved in the generation of faulty RSA signatures. We develop several techniques that extend Boneh *et al.*'s attack to a large class of partially known message configurations. We nonetheless assume that certain conditions on the unknown parts of the encoded message are met; these conditions may depend on the encoding function itself and on the hash functions used. To illustrate our attacks, we have chosen to consider the ISO/IEC 9796-2 standard [75]. ISO/IEC 9796-2 is originally a deterministic encoding scheme often used in combination with message randomization (*e.g.* in EMV [53]). The encoded message has the form:

$$\mu(m) = \texttt{6A}_{\texttt{16}} \,\|\, m[1] \,\|\, H(m) \,\|\, \texttt{BC}_{\texttt{16}}$$

where $m = m[1] \,\|\, m[2]$ is split into two parts. We show that if the unknown part of $m[1]$ is not too large (*e.g.* less than 160 bits for a 2048-bit RSA modulus), then a single faulty signature allows to factor N as in [29][1]. The new method is based

---

[1] In our attack, it does not matter how large the unknown part of $m[2]$ is.

on a result by Herrmann and May [72] for finding small roots of linear equations modulo an unknown factor p of N; [72] is itself based on Coppersmith's technique [41] for finding small roots of polynomial equations using the LLL algorithm [94]. We also show how to extend our attack to multiple UMPs and to *scenarii* where more faulty signatures can be obtained from the device.

It is trivially seen that other *deterministic* signature encoding functions such as PKCS#1 v1.5 can be broken by the new attack *even when the message digest is unknown*. We elaborate on this in further detail at the end of this Chapter.

5.1.4 The ISO/IEC 9796-2 standard

ISO/IEC 9796-2 is an encoding standard allowing partial or total message recovery [75, 76]. The encoding can be used with hash functions $H(m)$ of diverse digest sizes k_h. For the sake of simplicity we assume that k_h, the size of m and the size of N (denoted k) are all multiples of 8. The ISO/IEC 9796-2 encoding of a message $m = m[1] \,\|\, m[2]$ is

$$\mu(m) = \mathtt{6A}_{16} \,\|\, m[1] \,\|\, H(m) \,\|\, \mathtt{BC}_{16}$$

where $m[1]$ consists of the $k - k_h - 16$ leftmost bits of m and $m[2]$ represents the remaining bits of m. Therefore the size of $\mu(m)$ is always $k - 1$ bits. Note that the original version of the standard recommended $128 \le k_h \le 160$ for partial message recovery (see [75], §5, note 4). In [46], Coron, Naccache and Stern introduced an attack against ISO/IEC 9796-2; the authors estimated that attacking $k_h = 128$ and $k_h = 160$ would require respectively 2^{54} and 2^{61} operations. After Coron *et al.*'s publication, ISO/IEC 9796-2 was amended and the current official requirement (see [76]) is now $k_h \ge 160$. In a recent work Coron, Naccache, Tibouchi and Weinmann successfully attack the currently valid version of ISO/IEC 9796-2 [47].

To illustrate our purpose, we consider a message $m = m[1] \,\|\, m[2]$ of the form

$$m[1] = \alpha \,\|\, r \,\|\, \alpha' \,, \qquad m[2] = \textsc{data}$$

where r is a message part unknown to the adversary, α and α' are strings known to the adversary and DATA is some known or unknown string[2]. The size of r is denoted k_r and the size of $m[1]$ is $k - k_h - 16$ as required in ISO/IEC 9796-2. The encoded message is then

$$\mu(m) = \mathtt{6A}_{16} \,\|\, \alpha \,\|\, r \,\|\, \alpha' \,\|\, H(\alpha \,\|\, r \,\|\, \alpha' \,\|\, \textsc{data}) \,\|\, \mathtt{BC}_{16} \tag{5.2}$$

Therefore the total number of unknown bits in $\mu(m)$ is $k_r + k_h$.

5.2 Fault attack on partially-known message ISO/IEC 9796-2

This Section extends [29] to signatures of partially known messages encoded as described previously. We assume that after injecting a fault the opponent is in

---

[2] The attack will work equally well in both cases.

possession of a faulty signature σ such that:

$$\sigma^e = \mu(m) \mod p\ , \qquad \sigma^e \neq \mu(m) \mod q\ . \tag{5.3}$$

From (5.2) we can write

$$\mu(m) = t + r \cdot 2^{n_r} + H(m) \cdot 2^8 \tag{5.4}$$

where t is a known value. Note that both r and $H(m)$ are unknown to the adversary. From (5.3) we obtain:

$$\sigma^e = t + r \cdot 2^{n_r} + H(m) \cdot 2^8 \mod p\ .$$

This shows that $(r, H(m))$ must be a solution of the equation

$$a + b \cdot x + c \cdot y = 0 \mod p \tag{5.5}$$

where $a := t - \sigma^e \bmod N$, $b := 2^{n_r}$ and $c := 2^8$ are known. Therefore we are left with solving equation (5.5) which is linear in the two variables x, y and admits a small root $(x_0, y_0) = (r, H(m))$. However the equation holds modulo an unknown divisor p of N and not modulo N. Such equations were already exploited by Herrmann and May [72] to factor an RSA modulus $N = pq$ when some blocks of p are known. Their method is based on Coppersmith's technique for finding small roots of polynomial equations [41]. Coppersmith's technique uses LLL to obtain two polynomials $h_1(x, y)$ and $h_2(x, y)$ such that

$$h_1(x_0, y_0) = h_2(x_0, y_0) = 0$$

holds over the integers. Then one computes the resultant between h_1 and h_2 to recover the common root (x_0, y_0). To that end, we must assume that h_1 and h_2 are algebraically independent. This *ad hoc* assumption makes the method heuristic; nonetheless it turns out to work quite well in practice. Then, given the root (x_0, y_0) one recovers the randomized encoded message $\mu(m)$ and factors N by GCD.

Theorem 1 (Herrmann-May [72])**.** *Let N be a sufficiently large composite integer with a divisor $p \geq N^\beta$. Let $f(x, y) = a + b \cdot x + c \cdot y \in \mathbb{Z}[x, y]$ be a bivariate linear polynomial. Assume that $f(x_0, y_0) = 0 \mod p$ for some (x_0, y_0) such that $|x_0| \leq N^\gamma$ and $|y_0| \leq N^\delta$. Then for any $\varepsilon > 0$, under the condition*

$$\gamma + \delta \leq 3\beta - 2 + 2(1 - \beta)^{3/2} - \varepsilon \tag{5.6}$$

one can find $h_1(x, y), h_2(x, y) \in \mathbb{Z}[x, y]$ such that $h_1(x_0, y_0) = h_2(x_0, y_0) = 0$ over $\mathbb{Z}$, in time polynomial in $\log N$ and ε^{-1}.

We only sketch the proof and refer the reader to [72] for more details. Assume that $b = 1$ in the polynomial f (otherwise multiply f by $b^{-1} \bmod N$) and consider the polynomial

$$f(x, y) = a + x + c \cdot y$$

We look for (x_0, y_0) such that $f(x_0, y_0) = 0 \bmod p$. The basic idea consists in generating a family $\mathcal{G}$ of polynomials admitting (x_0, y_0) as a root modulo p^t for some large enough integer t. Any linear combination of these polynomials will also be a polynomial admitting (x_0, y_0) as a root modulo p^t. We will use LLL to find such polynomials with small coefficients. To do so, we view any polynomial $h(x, y) = \sum h_{i,j} x^i y^j$ as the vector of coefficients $\left(h_{i,j} X^i Y^j\right)_{i,j}$ and denote by $\|h(xX, yY)\|$ this vector's Euclidean norm. Performing linear combinations on polynomials is equivalent to performing linear operations on their vectorial representation, so that applying LLL to the lattice spanned by the vectors in $\mathcal{G}$ will provide short vectors representing polynomials with root $(x_0, y_0) \bmod p^t$.

We now define the family $\mathcal{G}$ of polynomials as

$$g_{k,i}(x, y) := y^i \cdot f^k(x, y) \cdot N^{\max(t-k,0)}$$

for $0 \le k \le m$, $0 \le i \le m - k$ and integer parameters t and m. For all values of indices k, i, it holds that

$$g_{k,i}(x_0, y_0) = 0 \mod p^t.$$

We first sort the polynomials $g_{k,i}$ by increasing k values and then by increasing i values. Denoting $X = N^\gamma$ and $Y = N^\delta$, we write the coefficients of the polynomial $g_{k,i}(xX, yY)$ in the basis $x^{k'} \cdot y^{i'}$ for $0 \le k' \le m$ and $0 \le i' \le m - k'$. This results in the matrix of row vectors illustrated in Table 5.1. The matrix is lower triangular; we only represent the diagonal elements.

Let L be the corresponding lattice; L's dimension is

$$\omega = \dim(L) = \frac{m^2 + 3m + 2}{2} = \frac{(m+1)(m+2)}{2}$$

and we have

$$\det L = X^{s_x} Y^{s_y} N^{s_N}$$

where

$$s_x = s_y = \sum_{k=0}^{m} \sum_{i=0}^{m-k} i = \frac{m(m+1)(m+2)}{6}$$

and

$$s_N = \sum_{i=0}^{t} (m + 1 - i)(t - i) = \frac{t}{6}(t+1)(t - 3m - 4).$$

We now apply LLL to the lattice L to find two polynomials $h_1(x, y)$ and $h_2(x, y)$ with small coefficients.

Theorem 2 (LLL [94]). *Let L be a lattice spanned by $(u_1, \ldots, u_\omega)$. Given the vectors $(u_1, \ldots, u_\omega)$, the LLL algorithm finds in polynomial time two linearly independent vectors b_1, b_2 such that*

$$\|b_1\|, \|b_2\| \le 2^{\omega/4} (\det L)^{1/(\omega-1)} .$$

$g_{i,j}(xX,yY)$	1	$\cdots$	y^m	x	$\cdots$	xy^{m-1}	$\cdots$	x^t	$\cdots$	x^ty^{m-t}	$\cdots$	x^{m-1}	$x^{m-1}y$	x^m
$g_{0,0}$	N^t													
$\vdots$		$\ddots$												
$g_{0,m}$			Y^mN^t											
$g_{1,0}$				XN^{t-1}										
$\vdots$					$\ddots$									
$g_{1,m-1}$						$XY^{m-1}N^{t-1}$								
$\vdots$				$*$			$\ddots$							
$g_{t,0}$								X^t						
$\vdots$									$\ddots$					
$g_{t,m-t}$										X^tY^{m-t}				
$\vdots$											$\ddots$			
$g_{m-1,0}$												X^{m-1}		
$g_{m-1,1}$													$X^{m-1}Y$	
$g_{m,0}$														X^m

Table 5.1: Lattice of row vectors formed by the coefficients of the polynomials $g_{k,i}(xX,yY)$

Therefore using LLL we can get two polynomials $h_1(x,y)$ and $h_2(x,y)$ such that

$$\|h_1(xX,yY)\|, \|h_2(xX,yY)\| \leq 2^{\omega/4} \cdot (\det L)^{1/(\omega-1)} \, . \tag{5.7}$$

Using Howgrave-Graham's lemma (below), we can determine the required bound on the norms of h_1 and h_2 to ensure that (x_0,y_0) is a root of both h_1 and h_2 over the integers:

Lemma 1 (Howgrave-Graham [73]). *Assume that $h(x,y) \in \mathbb{Z}[x,y]$ is a sum of at most ω monomials and assume further that $h(x_0,y_0) = 0 \mod B$ where $|x_0| \leq X$ and $|y_0| \leq Y$ and $\|h(xX,yY)\| < B/\sqrt{\omega}$. Then $h(x_0,y_0) = 0$ holds over the integers.*

Proof. We have

$$\begin{aligned} |h(x_0,y_0)| &= \left|\sum h_{ij} x_0^i y_0^i\right| = \left|\sum h_{ij} X^i Y^j \left(\frac{x_0}{X}\right)^i \left(\frac{y_0}{Y}\right)^j\right| \\ &\leq \sum \left|h_{ij} X^i Y^j \left(\frac{x_0}{X}\right)^i \left(\frac{y_0}{Y}\right)^j\right| \leq \sum \left|h_{ij} X^i Y^j\right| \\ &\leq \sqrt{\omega}\|h(xX,yY)\| < B \end{aligned}$$

Since $h(x_0,y_0) = 0 \bmod B$, this implies that $h(x_0,y_0) = 0$ over the integers. ☐ ☐

We apply Lemma 1 with $B := p^t$. Using (5.7) this gives the condition:

$$2^{\omega/4} \cdot (\det L)^{1/(\omega-1)} \leq \frac{N^{\beta t}}{\sqrt{\omega}} \, . \tag{5.8}$$

[72] shows that by letting $t = \tau \cdot m$ with $\tau = 1 - \sqrt{1-\beta}$, we get the condition:

$$\gamma + \delta \leq 3\beta - 2 + 2(1-\beta)^{3/2} - \frac{3\beta(1+\sqrt{1-\beta})}{m}$$

Therefore we obtain as in [72] the following condition for m:

$$m \geq \frac{3\beta(1+\sqrt{1-\beta})}{\varepsilon} \, .$$

Since LLL runs in time polynomial in the lattice's dimension and coefficients, the running time is polynomial in $\log N$ and $1/\varepsilon$.

5.2.1 Discussion

For a balanced RSA modulus ($\beta = 1/2$) we get the condition:

$$\gamma + \delta \leq \frac{\sqrt{2}-1}{2} \cong 0.207 \tag{5.9}$$

This means that for a 1024-bit RSA modulus N, the total size of the unknowns x_0 and y_0 can be at most 212 bits. Applied to our context, this implies that for ISO/IEC 9796-2 with $k_h = 160$, the size of the UMP r can be as large as 52 bits. Section 5.3 reports practical experiments confirming this prediction. In Section 5.6.1 we provide a Python code for computing the bound on the size of the unknown values $(k_r + k_h)$ as a function of the modulus size.

In the next paragraph we extend the method to 1) several disjoint UMP blocks in the encoding function, 2) to two dissimilar faults (one modulo p and one modulo q) and 3) to two or more faults modulo the same prime factor.

5.2.2 Extension to several unknown blocks

Assume that the UMP used in ISO/IEC 9796-2 is split into n different blocks, namely

$$\mu(m) = \mathtt{6A}_{16} \parallel \alpha_1 \parallel r_1 \parallel \alpha_2 \parallel r_2 \parallel \cdots \parallel \alpha_n \parallel r_n \parallel \alpha_{n+1} \parallel H(m) \parallel \mathtt{BC}_{16} \tag{5.10}$$

where the UMPs $r_1, \ldots, r_n$ are all part of the message m. The α_i blocks are known. We use the extended result of Herrmann and May [72], allowing to (heuristically) find the solutions $(y_1, \ldots, y_n)$ of a linear equation modulo a factor p of N:

$$a_0 + \sum_{i=1}^{n} a_i \cdot x_i = 0 \mod p$$

with $p \geq N^\beta$ and $|y_i| \leq N^{\gamma_i}$, under the condition [72]

$$\sum_{i=1}^{n} \gamma_i \leq 1 - (1-\beta)^{\frac{n+1}{n}} - (n+1)(1 - \sqrt[n]{1-\beta})(1-\beta) \ .$$

For $\beta = 1/2$ and for a large number of blocks n, one gets the bound

$$\sum_{i=1}^{n} \gamma_i \leq \frac{1 - \ln 2}{2} \cong 0.153$$

This shows that if the total number of unknown bits plus the message digest is less than 15.3% of the size of N, then the UMPs can be fully recovered from the faulty signature and Boneh *et al.*'s attack will apply again. However the number of blocks cannot be too large because the attack's runtime increases exponentially with n.

5.2.3 Extension to two faults modulo different factors

Assume that we can get two faulty signatures, one incorrect modulo p and the other incorrect modulo q. This gives the two equations

$$\begin{array}{ccccccccccc} a_0 & + & b_0 & \cdot & x_0 & + & c_0 & \cdot & y_0 & = & 0 \mod p \\ a_1 & + & b_1 & \cdot & x_1 & + & c_1 & \cdot & y_1 & = & 0 \mod q \end{array}$$

with small unknowns x_0, y_0, x_1, y_1. By multiplying the two equations, we get the quadri-variate equation:

$$\begin{aligned} &a_0a_1 + a_0b_1 \cdot x_1 + b_0a_1 \cdot x_0 + a_0c_1 \cdot y_0 + c_0a_1 \cdot y_1 + \\ &\qquad + b_0b_1 \cdot x_0x_1 + b_0c_1 \cdot x_0y_1 + c_0b_1 \cdot y_0x_1 + c_0c_1 \cdot y_0y_1 = 0 \mod N \end{aligned}$$

We can linearize this equation (replacing products of unknowns with new variables) and obtain a new equation of the form:

$$\alpha_0 + \sum_{i=1}^{8} \alpha_i \cdot z_i = 0 \mod N$$

where the coefficients $\alpha_0, \ldots, \alpha_8$ are known and the unknowns $z_1, \ldots, z_8$ are small. Using LLL again, we can recover the z_i's (and then x_0, x_1, y_0, y_1) as long as the cumulated size of the z_i's is at most the size of N. This yields the condition

$$6 \cdot (k_r + k_h) \leq k$$

which, using the notation of Theorem 1, can be reformulated as

$$\gamma + \delta \leq \frac{1}{6} \cong 0.167 \; .$$

This remains weaker than condition (5.9). However the attack is significantly faster because it works over a lattice of constant dimension 9. Moreover, the 16.7% bound is likely to lend itself to further improvements using Coppersmith's technique instead of plain linearization.

5.2.4 Extension to several faults modulo the same factor

To exploit single faults, we have shown how to use lattice-based techniques to recover p given N and a bivariate linear equation $f(x, y)$ admitting a small root (x_0, y_0) modulo p. In this context, we have used Theorem 1 which is based on approximate GCD techniques from [74]. In the present Section we would like to generalize this to use ℓ different polynomials of the same form, each having a small root modulo p. More precisely, let ℓ be a fixed parameter and assume that as the result of ℓ successive faults, we are given ℓ different polynomials

$$f_u(x_u, y_u) = a_u + x_u + c_u y_u \tag{5.11}$$

where each polynomial f_u has a small root (ξ_u, ν_u) modulo p with $|\xi_u| \leq X$ and $|\nu_u| \leq Y$. Note that, as in the basic case, we re-normalized each polynomial f_u to ensure that the coefficient of x_u in f_u is equal to one. To avoid double subscripts, we hereafter use the Greek letters ξ and ν to represent the root values. We would like to use a lattice approach to construct new multivariate polynomials in the variables $(x_1, \cdots, x_\ell, y_1, \cdots, y_\ell)$ with the root $R = (\xi_1, \cdots, \xi_\ell, \nu_1, \cdots, \nu_\ell)$. To that end we fix two parameters m and t and build a lattice on a family of polynomials $\mathcal{G}$ of degree at most m with root R modulo $B = p^t$. This family is composed of all polynomials of the form

$$y_1^{i_1}\, y_2^{i_2}\, \cdots\, y_\ell^{i_\ell}\, f_1(x_1,y_1)^{j_1}\, f_2(x_2,y_2)^{j_2}\, \cdots\, f_\ell(x_\ell,y_\ell)^{j_\ell}\, N^{\max(t-j,0)}\ ,$$

where each i_u, j_u is non-negative, $i = \sum_{u=1}^{\ell} i_u$, $j = \sum_{u=1}^{\ell} j_u$ and $0 \le i + j \le m$. Once again, let L be the corresponding lattice. Its dimension ω is equal to the number of monomials of degree at most m in 2ℓ unknowns, i.e.

$$\omega = \binom{m+2\ell}{2\ell}\ .$$

Since we have a common upper bound X for all values $|\xi_u|$ and a common bound for all $|\nu_u|$ we can compute the lattice's determinant as

$$\det(L) = X^{s_x} Y^{s_y} N^{s_N}\ ,$$

where s_x is the sum of the exponents of all unknowns x_u in all occurring monomials, s_y is the sum of the exponents of the y_u and s_N is the sum of the exponents of N in all occurring polynomials. For obvious symmetry reasons, we have $s_x = s_y$ and noting that the number of polynomials of degree exactly d in ℓ unknowns is $\binom{d+\ell-1}{\ell-1}$ we find

$$s_x = s_y = \sum_{d=0}^{m} d \binom{d+\ell-1}{\ell-1}\binom{m-d+\ell}{\ell} = \frac{\ell(2\ell+m)!}{(2\ell+1)!(m-1)!}.$$

Likewise, summing on polynomials with a non-zero exponent v for N, where the sum of the j_u is $t - v$ we obtain

$$\begin{aligned} s_N &= \sum_{v=1}^{t} v \binom{t-v+\ell-1}{\ell-1}\binom{m-t+v+\ell}{\ell} \\ &= \binom{1+\ell+m-t}{\ell}\binom{-2+\ell+t}{\ell-1}\, {}_3F_2\left[\begin{matrix} 2 \\ 1-t \\ 2+\ell+m-t \end{matrix}\ ;\begin{matrix} 2-\ell-t \\ 2+m-t \end{matrix};1\right] \end{aligned}$$

where ${}_3F_2$ is the generalized hypergeometric function.

As usual, assuming that $p = N^\beta$ we can find a polynomial with the correct root over the integers under the condition of formula (5.8).

Concrete bounds:

Using the notation of Theorem 1, we compute effective bounds on $\gamma+\delta = \log(XY)/\log(N)$ from the logarithm of condition (5.8), dropping the terms $\sqrt{\omega}$ and $2^{\omega/4}$ which become negligible as N grows. For concrete values of N, bounds are slightly smaller. Dividing by $\log(N)$, we find

$$s_x \cdot (\gamma + \delta) + s_N \le \beta t \omega\ .$$

Thus, given k, t and m, we can achieve at best

$$\gamma + \delta \leq \frac{\beta t \omega - s_N}{s_x}.$$

We have computed the achievable values of $\gamma + \delta$ for $\beta = 1/2$, for various parameters and for lattice dimensions $10 \leq \omega \leq 1001$. Results are given in Table 5.5, Section 5.6.2.

Recovering the root:

With 2ℓ unknowns instead of two, applying usual heuristics and hoping that lattice reduction directly outputs 2ℓ algebraically independent polynomials with the prescribed root over the integers becomes a wishful hope. Luckily, a milder heuristic assumption suffices to make the attack work. The idea is to start with K equations instead of ℓ and iterate the lattice reduction attack for several subsets of ℓ equations chosen amongst the K available equations. Potentially, we can perform $\binom{K}{\ell}$ such lattice reductions. Clearly, since each equation involves a different subset of unknowns, they are all different. Note that this does not suffice to guarantee algebraic independence; in particular, if we generate more than K equations they cannot be algebraically independent. However, we only need to ascertain that the root R can be extracted from the available set of equations. This can be done, using Gröbner basis techniques, under the heuristic assumption that the set of equations spans a multivariate ideal of dimension zero i.e. that the number of solutions is finite.

Note that we need to choose reasonably small values of ℓ and K to be able to use this approach in practice. Indeed, the lattice that we consider should not become too large and, in addition, it should be possible to solve the resulting system of equations using either resultants or Buchberger's algorithm which means that neither the degree nor the number of unknowns should increase too much.

Asymptotic bounds:

Despite the fact that we cannot hope to run the multi-polynomial variant of our attack when parameters become too large, it is interesting to determine the theoretical limit of the achievable value of $\gamma + \delta$ as the number of faults ℓ increases. To that end, we assume as previously that $\beta = 1/2$, let $t = \tau m$ and replace ω, s_x and s_N by the following approximations:

$$\omega \cong \frac{m^{2\ell}}{(2\ell)!}\ , \quad s_x \cong \sum_{d=0}^{m} \frac{d^{\ell}\,(m-d)^{\ell}}{(\ell-1)!\,\ell!}\ , \quad s_N \cong \sum_{v=1}^{t} v\,\frac{(t-v)^{\ell-1}(m-t+v)^{\ell}}{(\ell-1)!\,\ell!}\ .$$

For small ℓ values we provide in Table 5.2 the corresponding bounds on $\gamma + \delta$. Although we do not provide further details here due to lack of space, one can show that the bound $\gamma + \delta$ tends to $1/2$ as the number of faults ℓ tends to infinity and that all $\gamma + \delta$ values are algebraic numbers.

ℓ	1	2	3	4	5	6	7	8	9	10
$\gamma+\delta$	0.207	0.293	0.332	0.356	0.371	0.383	0.391	0.399	0.405	0.410

Table 5.2: Bound for the relative size $\gamma + \delta$ of the unknowns as a function of the number of faults ℓ

5.3 Simulation results

Assuming that fault injection can be performed on unprotected devices (see Section 5.4), we simulated the attack. In the experiment we generated faulty signatures (using the factors p and q) and applied to them the attack's mathematical analysis developed in the previous Sections to factor N. We refer the reader to Section 5.4 for more on physical fault injection experiments.

5.3.1 Single-fault attack simulations

We first consider a single-UMP, single-fault attack when $H = \text{SHA-1}$ *i.e.* $k_h = 160$. Using the SAGE library LLL implementation, computations were executed on a 2 GHz Intel notebook.

k, bits	k_r, bits	m	t	ω	runtime	
1024	6	10	3	66	4	minutes
1024	13	13	4	105	51	minutes
1536	70	8	2	45	39	seconds
1536	90	10	3	66	9	minutes
2048	158	8	2	45	55	seconds

Table 5.3: Single fault, single UMP 160-bit digests ($k_h = 160$). LLL runtime for different parameter combinations.

Experimental results are summarized in Table 5.3. We see that for 1024-bit RSA, the randomizer size k_r must be quite small and the attack is less efficient than exhaustive search[3]. However for larger moduli, the attack becomes more efficient. Typically, using a single fault and a 158-bit UMP, a 2048-bit RSA modulus was factored in less than a minute.

5.3.2 Multiple-fault simulations

To test the practicality of the approach presented in Section 5.2.4, we have set $(\ell, t, m) = (3, 1, 3)$ *i.e.* three faulty signatures. This leads to a lattice of dimension 84 and a bound $\gamma + \delta \leq 0.204$. Experiments were carried out with 1024, 1536 and 2048 bit RSA moduli. This implementation also relied on the SAGE library [136]

---

[3] Exhausting a 13-bit randomizer took 0.13 seconds.

running on a single PC. Quite surprisingly, we observed a very large number of polynomials with the expected root over the integers. The test was run for three random instances corresponding to the parameters in Table 5.4.

k, bits	k_r, bits	runtime
1024	40	49 seconds
1536	150	74 seconds
2048	250	111 seconds

Table 5.4: Three faults, single UMP, 160-bit digests ($k_h = 160$). LLL runtime for different parameter combinations.

For each parameter set, the first 71 vectors in the reduced lattice have the expected root. In fact, it is even possible to solve the system of equations without resorting to Buchberger's algorithm. Instead, we use a much simpler strategy. We first consider the system modulo a prime p' above 2^{160} (or above 2^{250} in the 2048-bit experiment). With this system, linearization suffices to obtain through echelonization the polynomials $x_i - \xi_i$ and $y_i - \nu_i$. Since p' is larger than the bounds on the values, this yields the exact values of ξ_1, ξ_2, ξ_3, ν_1, ν_2 and ν_3. Once this is done, the factors of N are easily recovered by computing the GCD of N with any of the values $f_i(\xi_i, \nu_i)$.

Three faults turn-out to be more efficient than single-fault attacks (Table 5.4 *vs.* Table 5.3). In particular for a 1024-bit RSA modulus, the three-fault attack recovered a 40-bit UMP r in 49 seconds[4], whereas the single-fault attack only recovered a 13-bit UMP in 51 minutes.

5.4 Physical fault injection experiments

We performed fault injection on an unprotected device to demonstrate the entire attack flow. We obtain a faulty signature from a general-purpose 8-bit microcontroller running an RSA implementation and factor N using the mathematical attack of Section 5.2.

Our target device was an Atmel ATmega128 [8], a very popular RISC μC with an 8-bit AVR core. The μC was running an RSA-CRT implementation developed in C using the BigDigits multiple-precision arithmetic library [18]. The μC was clocked at 7.3728 MHz using a quartz crystal and powered from a 5V source.

We induced faults using voltage spikes (cf. to [138] and [82] for more information on such attacks on similar μCs). Namely, we caused brief power cut-offs (spikes) by grounding the chip's V_{cc} input for short time periods. Spikes were produced by an FPGA-based board counting the μC's clock transitions and generating the spike at a precise moment. The cut-off duration was variable with 10ns granularity and

---

[4] We estimate that exhaustive search on a 40-bit UMP would take roughly a year on the same single PC.

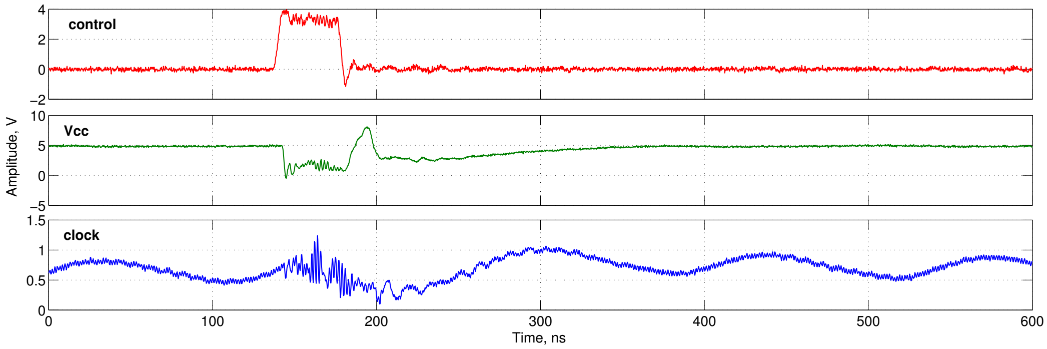


Figure 5.1: Spike captured with a DSO: control signal from FPGA, power supply cut-off, and induced glitch in the clock signal

the spike's temporal position could be fine-tuned with the same granularity. The fault was heuristically positioned to obtain the stable fault injection in one of the RSA-CRT branches (computing σ_p or σ_q). A 40ns spike is presented in Figure 5.1. Longer spike durations caused a μC's reset.

5.5 Conclusion

This Chapter introduced a new breed of partially-known message fault attacks against RSA signatures. These attacks allow to factor the modulus N given a single faulty signature. Although the attack is heuristic, it works well in practice and paradoxically becomes more efficient as the modulus size increases. As several faulty signatures are given longer UMPs and longer digests become vulnerable.

The new techniques are more generally applicable to any context where the signed messages are partially unknown, in which case we provide explicit size conditions for the fault attack to apply. This has a direct impact on other encoding functions, such as PKCS#1 v1.5 standard where a message m is encoded as

$$\mu(m) = \texttt{0001}_{16} \parallel \underbrace{\texttt{FF}_{16} \ldots \texttt{FF}_{16}}_{k_1 \text{ bytes}} \parallel \texttt{00}_{16} \parallel T \parallel H(m)$$

where T is a known sequence of bytes which encodes the identifier of the hash function and k_1 is a size parameter which is adjusted to make $\mu(m)$ have the same number of bytes than the modulus. With a single unknown bounded by N^δ the condition is $\delta < 0.25$. Therefore assuming a 2048-bit modulus and $H = $ SHA-512, we obtain that the modulus can be efficiently factored using a single faulty signature σ even when the signed message is *totally unknown*. This enables fault attacks in complex cryptographic *scenarii* where e.g. a smart-card and a terminal exchanging RSA signatures on encrypted messages.

5.6 Supplementary material

5.6.1 Python script for root size estimation

```
from math import log,sqrt

def LatticeExpo(t,m):
  sy=sx=sn=w=0
  for k in range(m+1):
    for i in range(m-k+1):
      j=max(t-k,0)
      sy+=i; sx+=k; sn+=j; w+=1
  return (sx,sy,sn,w)

def bound(t,m,n):
  (sx,sy,sn,w)=LatticeExpo(t,m)
  nxy=(w*(n*t*.5-.25*w-log(sqrt(w),2))-n*sn)/sx
  return nxy,w
```

5.6.2 Achievable bound on $\gamma + \delta$

We provide in Table 5.5 the achievable bound on $\gamma + \delta$, as a function of the number of faults ℓ and parameters (t, m).

5.6.3 Example of a faulty signature

Table 5.6 presents an example of a faulty 1536-bit RSA-CRT signature obtained during our fault injection experiments. Values known to the opponent are underlined.

Dimension ω	Bound $\gamma+\delta$	(ℓ,t,m)	Dimension ω	Bound $\gamma+\delta$	(ℓ,t,m)	Dimension ω	Bound $\gamma+\delta$	(ℓ,t,m)
10	0.100	$(1,1,3)$	15	0.125	$(1,1,4)$	21	0.129	$(1,1,5)$
28	0.143	$(1,2,6)$	36	0.155	$(1,2,7)$	45	0.158	$(1,2,8)$
55	0.161	$(1,3,9)$	66	0.168	$(1,3,10)$	78	0.171	$(1,3,11)$
91	0.172	$(1,3,12)$	105	0.176	$(1,4,13)$	120	0.179	$(1,4,14)$
120	0.179	$(1,4,15)$	153	0.181	$(1,5,16)$	171	0.183	$(1,5,17)$
190	0.184	$(1,5,18)$	210	0.184	$(1,5,19)$	231	0.186	$(1,6,20)$
253	0.187	$(1,6,21)$	276	0.188	$(1,6,22)$	300	0.190	$(1,7,23)$
325	0.190	$(1,7,24)$	351	0.190	$(1,7,25)$	378	0.190	$(1,8,26)$
406	0.192	$(1,8,27)$	435	0.192	$(1,8,28)$	465	0.192	$(1,9,29)$
496	0.193	$(1,9,30)$	528	0.194	$(1,9,31)$	561	0.194	$(1,9,32)$
595	0.194	$(1,10,33)$	630	0.195	$(1,10,34)$	666	0.195	$(1,10,35)$
703	0.195	$(1,10,36)$	741	0.196	$(1,11,37)$	780	0.196	$(1,11,38)$
820	0.196	$(1,11,39)$	861	0.196	$(1,12,40)$	903	0.197	$(1,12,41)$
946	0.197	$(1,12,42)$	990	0.197	$(1,12,43)$			
15	0.125	$(2,1,2)$	35	0.179	$(2,1,3)$	70	0.179	$(2,1,4)$
126	0.214	$(2,2,5)$	210	0.222	$(2,2,6)$	330	0.229	$(2,3,7)$
495	0.240	$(2,3,8)$	715	0.241	$(2,3,9)$	1001	0.248	$(2,4,10)$
28	0.167	$(3,1,2)$	84	0.204	$(3,1,3)$	210	0.222	$(3,2,4)$
462	0.247	$(3,2,5)$	924	0.247	$(3,2,6)$			
45	0.188	$(4,1,2)$	165	0.216	$(4,1,3)$	495	0.244	$(4,2,4)$
66	0.200	$(5,1,2)$	286	0.223	$(5,1,3)$	1001	0.258	$(5,2,4)$
91	0.208	$(6,1,2)$	455	0.228	$(6,1,3)$			
120	0.214	$(7,1,2)$	680	0.231	$(7,1,3)$			
153	0.219	$(8,1,2)$	969	0.234	$(8,1,3)$			
190	0.222	$(9,1,2)$	231	0.225	$(10,1,2)$	276	0.227	$(11,1,2)$
325	0.229	$(12,1,2)$	378	0.231	$(13,1,2)$	435	0.232	$(14,1,2)$
496	0.233	$(15,1,2)$	561	0.234	$(16,1,2)$	630	0.235	$(17,1,2)$
703	0.236	$(18,1,2)$	780	0.237	$(19,1,2)$	861	0.238	$(20,1,2)$
946	0.238	$(21,1,2)$						

Table 5.5: Achievable bounds for $\gamma+\delta=\log(XY)/\log(N)$

RSA PARAMETERS		
96	$p =$	`0x99c72d76722f217240dea9dead9339ed0f610e47166a6b995c5b4dd62532f0bf3c7a1875431f3b98af2b5fc284d13956e2f58a819ba9e5be28b300ae99a43c08e601cf6da250cd2902dde345b90632201cc2eebfe776151a53409d87aa1f27a5`
96	$q =$	`0xca60434275ea27e7eedba94fac9947986569b321a8784c1b1283f2b3c62746f98ca4fbe00609750e11c239ca222a52c540ababba6133504ec3610556e46ba6caa52f63dcb5f081ae58adfeb6ef41fc744528af66bfdb79a026631af8568fbcbd`
192	$N =$	`0x7990fcf78f986a4bd285a5f8c4ac98d546c98d95fea23f3034bfc306cc22d9c3423743ecbc346add601a94eff298f84f5476896c704fd364121c0d9d6873cebe3124339bd3298a98ce4da3d42efe3a863e25979613e7e357c17e248e928bf01f1abe69e6cd0c761cc54a21edc60466e5c49fa0abdf57fabc21ec41ca51f58d7b44f08666b408a0508c6e114efdbbc6de3c2d65d350d18720044d5410d1afbbb69dc0cf57da519aa7aec644b9f792e369cf6501359305b059475696a80d4870d1`
1	$e =$	`0x03`
96	$d =$	`0x510b534fb51046dd3703c3fb2dc865e384865e63ff16d4cacdd52caf32c1e6822c24d7f32822f1e8eabc634aa1bb5034e2f9b0f2f58a8ced616809139af7df2976182267e21bb1bb3433c28d74a97c597ec3ba640d45423a80fec309b707f5692464a61e98a21dd70e5fdf2a47e543958a8dea2cc04e2cafccb3562aef12392c528ba160f1ea9fc687aafa818f2ad1d6bb081fba37f8360cbad0deb237bfe5ecb70a68090160328ae226ec7e34dc788e48fb975f47cd6bbf33cc941ab311084b`
MESSAGE FORMATTING		
71	text $=$	`Fault Attacks on RSA Signatures with Partially Unknown Message Encoding`
81	$\alpha =$	`O for a Muse of fire, that would ascend the brightest heaven of invention, a king`
8	$r =$	`0xb43558c5e4aeb6e8`
81	$\alpha' =$	`dom for a stage, princes to act and monarchs to behold the swelling scene! Then s`
20	$H(\alpha \vert\vert r \vert\vert \alpha') =$	`0xb27dd515171666e807e48cc7d811d91eb824cb27`
192	$\mu(m) =$	`0x6a4f20666f722061204d757365206f6620666972652c207468617420776f756c6420617363656e64207468652062726967687465737420686561766656e206f6620696e76656e74696f6e2c2061206b696e67b43558c5e4aeb6e8646f6d20666f7220612073746167652c207072696e63657320746f2061637420616e64206d6f6e617263687320746f206265686f6c64207468652073776656c6c696e67207363656e6521205468656e2073b27dd515171666e807e48cc7d811d91eb824cb27bc`
FAULTY SIGNATURE (FAULT IN σ_p)		
96	$\sigma'_p =$	`0x23ebbf4274295ed52ec83f6d67da3f08ead22455b956d6e8f4eadf36624627616c632b3899a64e0e2e5bec2d66f66e67146974fb441365dcab1a7a33e4fe4868a9e8955fa2f2f0572c66333e6898e3e5ad9ed88665126d182ab3677b172733af`
96	$\sigma_q =$	`0x2e0505263d4e7eebbfd294f06ffb9cb00816730d30cc27a52918920d9bdcb9d808fc4cbddb5fae85dd40f50fef20aafbe73917e8daae4d7c9f96ac102bf64b80e4947f3509ed11a659c62a6e5f24ed611e7dffc9e77fbd1a42f41e946be7561e`
192	$\sigma' =$	`0x5defad0dc0082ec43c48860dbc3cf4160a66deb6c2304cd1323920941ad5a184bf92da34608e407042e8db9f3778b181f58f3fd3b6851ca0fd8fb975419c9f288d13f2f82a161be25a8fb27810f83a118f6a4f2f7e5cfc54582d88862ba3eec9a3ceb7330fe4dde64c3f99ef23a07a78180dd43d2c455e52c3bc55553d25e36cf8265eb4bf4c7b9198a8b89d622ffcd52ed1702988f40602064bae6a7d599884b8d23f5394d2b4aeecbd33d177f46baf71cdff70cafabfd0ac1ea2548c5ab5a6`
ADDITIONAL DATA: CORRECT SIGNATURE (σ_q SAME AS ABOVE)		
96	$\sigma_p =$	`0x8c0bd7487bd14111703e1c7d3d94cdbf6cc37f1c7f958d764e93285c3ef94c1e8dae46a4fe1615b194515c3f646bd01a358756dad5abdab19bf95b118ce98002fda41ae0cc583442cf7b50cea8740d1428c1451bb6d9daceb55d33dfb3363194`
192	$\sigma =$	`0x5a67520fb1a049a1b027905867280e66fadbdb2d375c05ea6678e77db836f99148134a673e009d4e09e02e68513228d986f580340f3b474012d28fb0f0aa221cd5a6485d5a17ec0993240b923c4167ebd959bac0c3194bade0de8baa3e3782da4981465e8d721709be01d70242cafdd6c7d031bb6ca32d9dececf4ad06fdeca85e5b6b6f63a4284241ffe5c9f8937499607ae67c3c5852663f39c48f8e69526d475af6214f36d09e66b17268e2c3b674c08248592164f45328721e3f75c9c0ac`

Table 5.6: Faulty 1536-bit RSA signature example ($H =$ SHA-1)

Chapter 6

Efficient generation of random delays in embedded software

Random delays are a countermeasure against a range of side-channel and fault attacks which is often implemented in embedded software to protect cryptographic implementations. In this Chapter we propose a new method for generation of random delays and a criterion for measuring the efficiency of a random delay countermeasure. We implement this new method along with the existing ones on an 8-bit platform and mount practical side-channel attacks against the implementations. We show that the new method is significantly more efficient and secure in practice than the previously published solutions.

This is a joint work with Jean-Sébastien Coron which was published in [44] and [45].

Contents

6.1 Introduction

Insertion of random delays in the execution flow of a cryptographic algorithm is a simple yet rather effective countermeasure against side-channel and fault attacks. To our knowledge, random delays are often used for protection of cryptographic implementations in embedded devices. It belongs to a group of *hiding* countermeasures, that introduce additional noise (either in time, amplitude or frequency domain) to the side-channel leakage while not eliminating the informative signal itself. This is in contrary to *masking* countermeasures, that eliminate dependency between the instantaneous side-channel leakage and the sensitive data processed by an implementation.

Hiding countermeasures increase complexity of attacks while not rendering them completely impossible. They are not treated in academia as extensively as masking but are of great importance in industry. A mixture of multiple hiding and masking countermeasures would often be used in a real-life protected implementation to raise the complexity of attacks above the foreseen capabilities of an adversary.

There are two connected problems that arise in this field. The first one is to develop efficient countermeasures, and the second one is how to measure the efficiency of the countermeasures. In this Chapter we tackle both tasks for the case of the random delays.

Our contribution. We propose a new algorithm for generating random delays in software which is significantly more efficient and secure than the existing methods. Our main idea is to generate random delays non-independently in order to introduce greater uncertainty for the attacker for the same mean of the cumulative delay. Our method allows to use more frequent shorter delays compared the previous methods. We thoroughly analyze our new method and present its improved method that allows for a wider choice of parameters and better security.

We also introduce a criterion for estimating the efficiency of random delays. This criterion shows how much uncertainty is introduced the sum of the delays for a given performance overhead. Our efficiency criterion is directly linked with the number of traces in a DPA attack and is information-theoretically sound.

We present and efficient implementation of our method for and 8-bit Atmel AVR paltform. We demonstrate by mounting practical side-channel attacks that our new method is indeed more efficient and secure than the existing solutions.

6.2 Random delays as a countermeasure

We begin with an overview of random delays in countering implementation attacks and figuring out assumptions about attacker's capabilities and the design criteria for the countermeasure. For another discussion on random delays in software we refer the reader to [156].

Most side-channel and fault attacks require an adversary to know precisely when the target operations occur in the execution flow. This enables her to synchronize multiple traces at the event of interest as in the case of DPA and to inject some disturbance into the computations at the right time as in the case of fault attacks. By introducing random delays into the execution flow the synchronization is broken, which increases the attack complexity. This can be done in hardware with the *Random Process Interrupts* (RPI) [39] as well as in software. In this work we focus on the software random delays.

6.2.1 Random delays in software

A common way of implementing random delays in software is placing loops of "dummy" operations (like `NOP` instructions) at some points of the program. The number of loop iterations varies depending on the delay value.

A straightforward method is to generate individual delays independently with durations uniformly distributed in the interval $[0, a]$ for some $a \in \mathbb{N}$. We refer to this method as *plain uniform delays*. It is easily implementable in cryptographic devices as most of them have a hardware random number generator (RNG) on board.

Work [156] presents an example of a loop of "dummy" operations in an 8051 assembly language. We will present implementation of random delays for an 8-bit Atmel AVR μC in Section 6.8.

6.2.2 Countering the effect of random delays

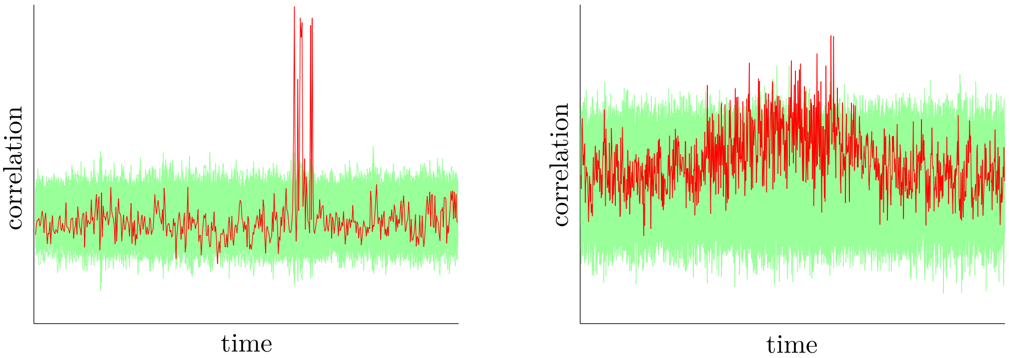
As already mentioned, introducing random delays into the execution flow breaks the synchronization for the attacker. More precisely, the traces (or parts of the traces if multiple dealys are used) in a side-channel attack will be misaligned, and in a fault attack the fault injection precision will be lost.

The ways of overcoming the random delay countermeasure are important in order to have proper assumptions about the attacker's capabilities and to figure out the design criteria for the countermeasure. So we look into this in detail.

In a side-channel attack, an attacker facing random delays has several choices:

- proceed with the attack as is;
- pre-process the traces using integration techniques to reconstruct the signal;
- re-synchronize the traces, removing the delays.

If the attacker proceeds with a side-channel attack as is, without any trace pre-processing or delay removal, the informative signal will be distributed over many points in time which will lead, for example, to a smeared DPA peak as shown in Figure 6.1(b), and therefore greater averaging will be required the succeed. The signal may be reconstructed with the integration techniques, as first described in [39].



(a) No delays (b) Traces misaligned with a delay

Figure 6.1: Effect of random delays in a DPA attack

It was shown in [39] that that the complexity of a DPA attack, expressed as the number of power consumption traces required, grows quadratically (in case traces are used as is) or linearly (in case integration techniques are used) with the standard deviation of the trace displacement in the attacked point. That is why here we are initially interested in making the variance of random delays as large as possible. However, here we will argue that variance is not a good measure of security for complex delay distributions, and one must look instead more generally at the *uncertainty* introduced by the delay. We will recall the relation from the work [100] between the parameters of the timing disarrangement distribution and the complexity of the DPA attack in Section 6.7.

Regarding side-channel trace resynchronization, note that it could be possible to place two sufficiently large and uniformly distributed delays in the beginning and in the end of the execution. That would make each point in the trace uniformly distributed over time when looking from the start of from the end, which is actually the worst case for an attacker in terms of the uncertainty introduced. Unfortunately, in this case it would be relatively easy to synchronize the traces with the the help of cross-correlation [108] or frequency-based techniques [92]. This is called *static alignment*.

So we assume that the static alignment of traces *can* be performed by an attacker. Therefore, the execution should be interleaved with delays in multiple places. To minimize the performance overhead in this setting, individual delays should be possibly shorter. An attacker would typically face a cumulative sum of the delays between the synchronization point and the target event. The cumulative delay should increase the uncertainty of an attacker about the location of the target event in time.

Removing multiple delays from a side-channel trace is called *dynamic alignment*. A dynamic alignment technique from [158] called elastic alignment was reported to be able to reduce the effect of multiple delays. It can be also possible to detect software random delays in a side-channel trace because of a regular instruction pattern. To partially hinder this, “dummy” random data may be processed within a loop. Work [98] applied spectral analysis techniques to eliminate the effect of multiple random delays. Within the scope of this work we do not address dynamic alignment, assuming that even with this technique it is harder to eliminate multiple delays than just a few ones.

In a fault attack spoiled by random delays, an attacker does not have an option of post-processing. One could think of on-the-fly synchronization of fault injection by observing patterns in side-channel leakage, but this seems hard to implement and to our knowledge no concrete implementations were reported, though a technical solution for on-the-fly synchronization exists [127]. An approach similar to the computational safe-error attack [165] could also be applicable in some cases, but it will anyway require an increased amount of fault injection attempts.

6.2.3 Assumptions and design criteria

To summarize, here are our preliminary assumptions about the attacker’s capabilities.

1. An attacker knows the times when the cryptographic algorithm execution starts and ends. This is commonly possible by monitoring I/O operations of a device, or operations like EEPROM access.

2. It is harder for an attacker to eliminate multiple random delays than a few ones.

3. The method of delay generation and its parameters are known to an attacker.

So an attacker will typically face the cumulative sum of several random delays. Following the Central Limit Theorem, the distribution of the sum of N *independent* (and *not necessarily uniform*) delays converges to normal with mean $N\mu_d$ and variance $N\sigma_d^2$, where μ_d and σ_d^2 are correspondingly the mean and the variance of the duration of an individual random delay. In other words, the distribution of the sum of independent delays depends only on the mean and the variance of individual delays but *not* on their particular distribution. In this case, increasing the uncertainty for the attacker is equivalent to increasing the variance of the (approximately) normal distribution of the cumulative sum.

With all the above in mind, we adhere to the following criteria for random delay generation.

1. Multiple random delays should be introduced in the execution flow.
2. The sum of random delays from start or end of the execution to some target event within the execution should introduce the maximum possible uncertainty for the attacker about the location of the target event.
3. The performance overhead should be possibly minimal.

6.2.4 Related work

First detailed treatment of the countermeasure was done by Clavier et al. in [39]. As already mentioned, they showed that the number of traces for a successful DPA attack against RPI grows quadratically or linearly (when integration is used) with the standard deviation of the delay. Mangard presented statistical analysis of random disarrangement effectiveness in [100]. Amiel *et al.* [3] performed practical evaluation of random delays as a protection against fault attacks. Works of Bucci *et al.* [33] and Lu *et al.* [97] on random delays in hardware should also be mentioned here, the former suggesting the architecture for delay generation at the gate level, the latter implementing it on FPGA and addressing the optimization of delay generation parameters for this architecture. Frequency-based side-channel attacks were applied to severeal random delay generation algorithms (including our new algorithm presented here) in [98].

To date, the only effort to *improve* the random delays countermeasure in software was published by Benoit and Tunstall in [156]. They suggested to modify the distribution of an individual independently generated random delay so that the variance of the sum increases and the mean, in turn, decreases. As a result, they achieve some improvement. We outline their method briefly here in Sect. 6.3.

6.3 Method of Benoit and Tunstall

In [156], Benoit and Tunstall propose a way to improve the efficiency of the random delays countermeasure. Their aim is to increase the variance and decrease the mean of the sum of random delays while not spoiling the distribution of an individual random delay. To achieve this aim, the authors modify the distribution of

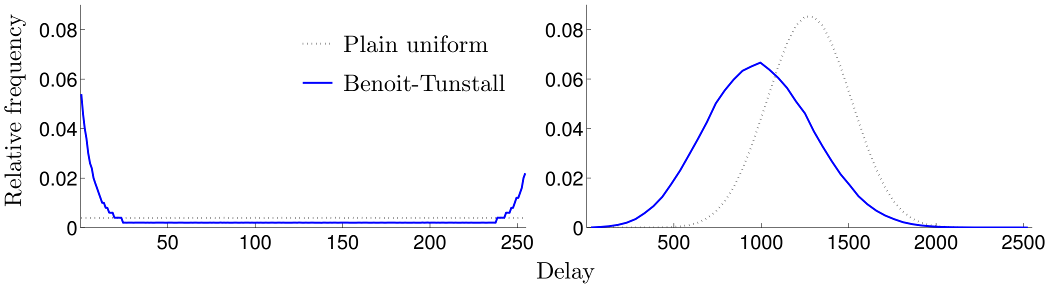


Figure 6.2: Distribution for the method of Benoit and Tunstall [156] compared to plain uniform delays: 1 delay (left) and sum of 10 delays (right)

an independently generated individual delay from the uniform to a pit-shaped one (see Figure 6.2). This increases the variance of the individual delay. Furthermore, some asymmetry is introduced to the pit in order to decrease the mean of an individual delay. The pit-shaped distribution is implemented by tabulating its inverse cumulative distribution function (c.d.f.).

The delays are generated independently, so if an individual delay has mean μ_{BT} and variance σ^2_{BT}, the distribution of the sum of N delays converges to normal (as in the case of plain uniform delays) with mean $N\mu_{\mathrm{BT}}$ and variance $N\sigma^2_{\mathrm{BT}}$.

The authors estimate efficiency of their method by comparing it to plain uniform random delays. In an example, they report an increase of the variance by 33% along with a decrease of the mean by 20%. Distributions for a single delay and for the sum of 10 delays (for the parameters from the example mentioned above, see [156]) are shown in Figure 6.2 in comparison to plain uniform delays.

We note that the authors also pursued an additional criterion for the difficulty of deriving the distribution of the random delay. But it seems reasonable to consider this distribution to be known to an adversary, at least if the method is published.

6.4 New method: Floating Mean

In this Section we present our new method for random delay generation in software. The main idea of the method is to generate random delays non-independently. This significantly improves the variance of the cumulative delay and the method is also more efficient compared to [156] and to plain uniform random delays.

By $x \sim \mathcal{DU}[y, z]$ we will denote a random variable x following discrete uniform distribution on $[y, z]$, $y, z \in \mathbb{Z}$, $y < z$. We consider the parameters of the method below to be integers as in an embedded device integer arithmetic would be the only option when generating delays.

6.4.1 Description

Our method is as follows. First, we fix some $a \in \mathbb{N}$ which is the maximum delay length. Additionally, we fix another parameter $b \in \mathbb{N}$, $b \leq a$. These implementation

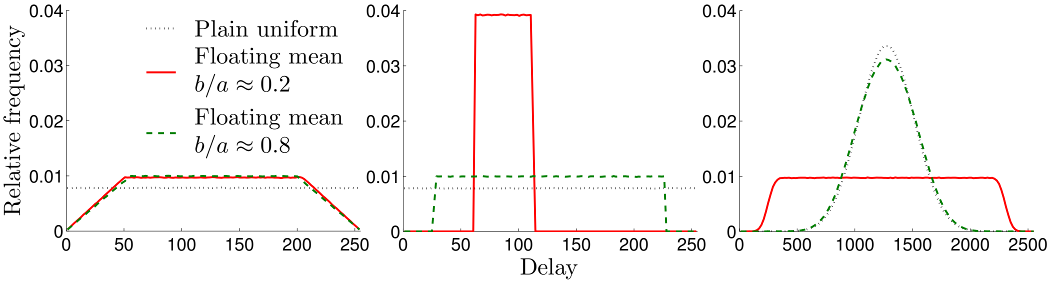


Figure 6.3: Distribution for the Floating Mean method with different b/a ratio compared to plain uniform delays: histogram for 1 delay (left), for 1 delay within a single trace, *i.e.* for some fixed m (center) and for the sum of 10 delays (right), $a = 255$; delay duration counted in atomic delay units

parameters a and b are fixed in an implementation and do not change between different executions of an algorithm under protection.

Now, in each execution, we first produce a value $m \in \mathbb{N}$ randomly uniformly on $[0, a-b]$, and then generate individual delays independently and uniformly on $[m, m+b]$. In other words, within any given execution individual random delays have a fixed mean $m+b/2$. But this mean varies from execution to execution, hence our naming of the method[1].

The resulting histograms in comparison to plain uniform delays are depicted in Figure 6.3. This figure also shows how the properties of the method vary dependent on the ratio b/a of the parameters of the method, that can take possible values between 0 and 1.

In fact, Floating Mean is a pure trade-off between the quality of the distribution of single delay within a trace and that of the sum of the delays. When b/a is small (like the case $b = 50$, $a = 255$, $b/a \approx 0.2$ in Figure 6.3), the distribution of an individual delay within a trace has a comparatively small variance, but the variance of a single delay across traces and of the sum of the delays is large. When b/a is large (like the case $b = 200$, $a = 255$, $b/a \approx 0.8$ in Figure 6.3), the distribution of an individual delay within a trace has large variance, but the distribution of the sum of the delays converges to normal. The extreme case $b/a = 0$ just means that within an execution all delays have same length m, while the distribution of the sum of N delays is uniform on the N-multiples in $[0, aN]$. In the other extreme case, $b/a = 1$, the methods simply converges to plain uniform delays with each delay generated uniformly on $[0, a]$.

6.4.2 Parameters of the distribution

To calculate the parameters of the distribution of the sum S_N of N delays, we represent an individual delay as a random variable $d_i = m + v_i$, where $m \sim \mathcal{DU}[0, a-$

---

[1] Not to be confused with the floating mean counting algorithms that exist in other domains of computer science

$b]$ and $v_i \sim \mathcal{DU}[0,b]$ for $i = 1, 2, \dots N$ are independent random variables. The sum is then expressed as

$$S_N = \sum_{i=1}^{N} d_i = Nm + \sum_{i=1}^{N} v_i \,.$$

For the mean, we have

$$E(S_N) = E(Nm) + E\left(\sum_{i=1}^{N} v_i\right) = N \cdot \frac{a-b}{2} + N \cdot \frac{b}{2} = \frac{Na}{2} \,.$$

For the variance, since m and v_i are independent, all v_i are identically distributed and

$$\mathrm{Var}(m) = \frac{(a-b+1)^2-1}{12} \,, \quad \mathrm{Var}(v_i) = \frac{(b+1)^2-1}{12} \,, \quad i = 1, 2, \dots, N$$

we have

$$\begin{aligned}\mathrm{Var}(S_N) = \mathrm{Var}\left(Nm + \sum_{i=1}^{N} v_i\right) &= N^2 \cdot \mathrm{Var}(m) + N \cdot \mathrm{Var}(v_1) \\ &= N^2 \cdot \frac{(a-b+1)^2-1}{12} + N \cdot \frac{b^2+2b}{12} \,.\end{aligned}$$

So, the variance of the sum of N delays is in $\Theta\left(N^2\right)$, in comparison to plain uniform delays and the method of [156] that both have variances in $\Theta\left(N\right)$. This is because we generate random delays non-independently; namely in our solution the lengths of the individual random delays are correlated: they are short if m is small, or they are longer if m is larger. This enables us to get a much larger variance than if the delays were generated independently, as in the plain uniform method and the method of [156]. Figure 6.4 further shows the advantage of our new method, for parameters $a = 200$, $b = 40$, $N = 100$.

6.4.3 Adding independence from m

At the same time, if we look at the delays within a single execution and thus under fixed m, the mean for the sum of N delays becomes $N(m + b/2)$. This implies that the cumulative delay for a given execution and therefore the length of the execution depends on m. An adversary can thus accept only the short traces, as they have short individual delays, and reject the long ones; this can lower the complexity of the attack.

In order to relieve an adversary of such a benefit, we can generate the first half of random delays (in the first half of the execution) uniformly on $[m, m+b]$ (that is, with mean $m + b/2$), and the second half of delays – uniformly on $[a-m-b, a-m]$ (that is, with mean $a - m - b/2$). In this way, the distribution of the sum of all the $N = 2M$ delays for a given execution is independent of m (the mean is $aN/2$ and the variance is $N(b^2 + 2b)/12$). So an adversary cannot gain any additional

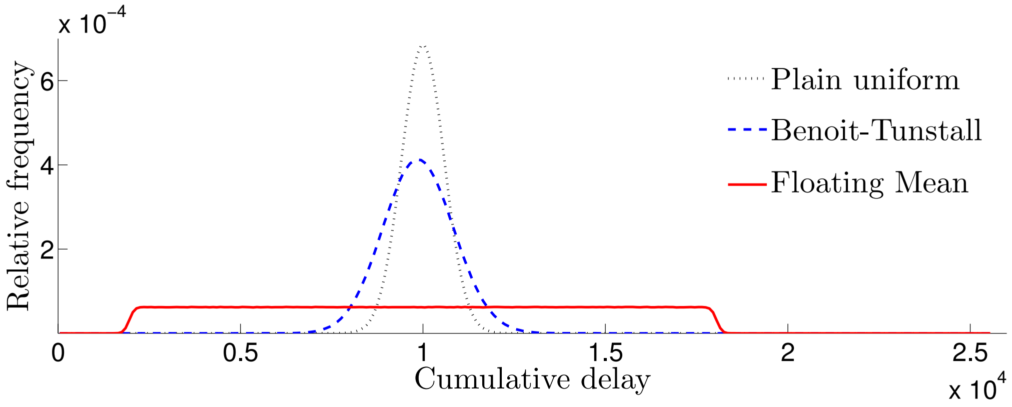


Figure 6.4: Empirical distributions of the sum of 100 delays for random delay generation algorithms, for the case of equal means; delay length counted in atomic delay units

information about the distribution of the delays within an execution by observing its length. Still, the variance of the sum of $L < M$ delays from start or end to some point up to the middle of the execution is in $\Theta(L^2)$.

In [44] we presented another method that was based on the same principle of non-independent individual delay generation but did not quite work because it was not possible to make the distribution of the cumulative sum independent from m.

6.4.4 Full algorithm

The full Floating Mean method is described in Algorithm 7. It is easily implementable in software on a constrained platform that has a built-in RNG producing uniformly distributed bytes. The method requires no additional memory, as opposed to [156]. We are describing efficient implementation of Floating Mean in Sect. 6.8.

Algorithm 7: Floating Mean method for generation of random delays

```
Input: $a, b \in \mathbb{N}, b \leq a, N = 2M \in \mathbb{N}$
$m \leftarrow \mathcal{DU}[0, a-b]$
for $i = 1$ to $N/2$ do
    $d_i \leftarrow m + \mathcal{DU}[0, b]$
end for
for $i = N/2 + 1$ to $N$ do
    $d_i \leftarrow a - m - \mathcal{DU}[0, b]$
end for
Output: $d_1, d_2, \ldots, d_N$
```

6.5 Analysis of Floating Mean

In this Section, we show that the Floating Mean method provides less security than expected for small a and b. We explain how to choose the parameters a and b depending on the number of delays N.

6.5.1 Behaviour with different parameters

We begin with taking a detailed look at the distributions of different methods by simulating them with the parameters that we will use in our practical implementation (Section 6.8). In Figure 6.5 we present histograms of the distributions for different methods. Namely, the number of delays in the sum is $N = 32$ and the parameters of the Floating Mean method are $a = 18$, $b = 3$. The histograms present the relative frequency of the cumulative delay against its duration[2]. We clearly see a multimodal distribution for the Floating Mean method: the histogram has a distinct shape of a saw with 16 cogs, and not a flat plateau as onw wold expect from Section 6.4 (Figure 6.4).

These cogs are not good for security since they make it easier for an attacker to mount an attack. The classical DPA will be more efficient since the signal is concentrated on the top of the 16 cogs instead of being spread over the clock cycles. In case of an attack with windowing and integration [39], the attacker would integrate the values around cog maximums, omit the minimums to reduce the noise (assuming noise is the same in all the points of the trace) and thus gain a reduction in the number of traces required for an attack.

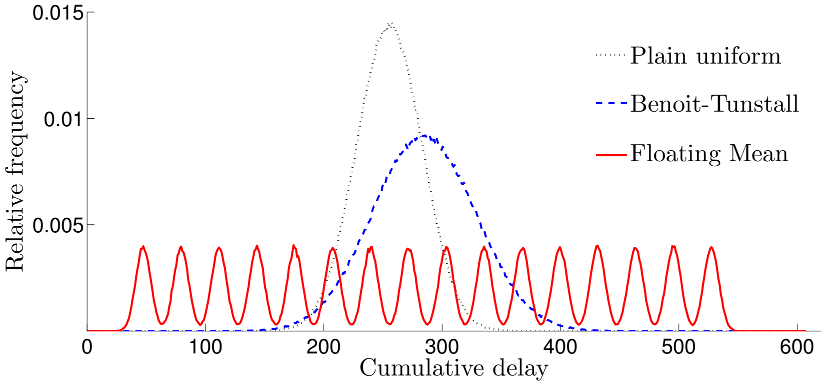


Figure 6.5: Empirical distributions of the sum of 32 delays in our experiments from Section 6.9

---

[2] As in Figure 6.4, the duration in Figure 6.5 is expressed in atomic delay units, *i.e.* as the total number of delay loop iterations. To obtain the value in clock cycles one should multiply this by the length of a single delay loop, which will be 3 clock cycles in our implementation reported in Section 6.8.

6.5.2 Explaining the cogs

Here we explain how cogs arise in the distribution of the Floating mean and we show how to choose the parameters to avoid the cogs.

The distribution for the Floating Mean is in fact a *finite mixture* [104] of $a-b+1$ components with equal weights. Every component corresponds to a given integer value m in $[0, a-b]$. For a given m, the cumulative sum of random delay durations is the sum of N random variables uniformly and independently distributed in $[m, m+b]$. Therefore it can be approximated by a Gaussian distribution with mean

$$\mu_m = N \cdot (m + b/2)$$

and variance

$$V = N\frac{(b+1)^2 - 1}{12} \, .$$

Therefore the probability density of the distribution for random integer $m \in [0, a-b]$ can be approximated by:

$$f(x) = \sum_{m=0}^{a-b} \frac{1}{(a-b+1)\sigma\sqrt{2\pi}} \exp\left(-\frac{(x-\mu_m)^2}{2\sigma^2}\right)$$

where all components have the same standard deviation $\sigma = \sqrt{V}$:

$$\sigma = \sqrt{N} \cdot \sqrt{\frac{(b+1)^2 - 1}{12}} \, .$$

The cog peaks are the modes of the components, located in their means μ_m. The distance between the means of successive components is $\mu_{m+1} - \mu_m = N$. We can consider the cogs distinguishable by comparing the standard deviation σ of the components to the distance N between the means. Namely, the distribution becomes multimodal whenever $\sigma \ll N$. In Figure 6.5 for the Floating Mean we have $a = 18$, $b = 3$ and $N = 32$, which gives $\sigma = 6.3$; therefore we have $\sigma < N$ which explains why the 16 cogs are clearly distinguishable in the distribution. However for $a = 200$, $b = 40$ and $N = 100$ we get $\sigma = 118$ so $\sigma > N$ which explains why the cogs are indistinguishable in Figure 6.4 and a flat plateau is observed instead.

6.5.3 Choosing correct parameters

From the above we derive the simple rule of thumb for choosing Floating Mean parameters. To ensure that no distinct cogs arise, parameter b should be chosen such that $\sigma \gg N$. For sufficiently large b we can approximate σ by:

$$\sigma \simeq \frac{\sqrt{3}}{6} \cdot b \cdot \sqrt{N}$$

Therefore this gives the condition:

$$b \gg \sqrt{N} \, . \tag{6.1}$$

However as observed in Section 6.2 it is better to have a large number of short random delays rather than a small number of long delays; this is because rare longer delays are a priori easier to detect and remove than multiple short delays. But we see that condition (6.1) for Floating Mean requires longer delays since by definition the length of random delays is between $[m, m+b]$ with $m \in [0, a-b]$. In other words, condition (6.1) contradicts the requirement of having many short random delays.

In the next Section we describe a variant of the Floating Mean which does not suffer from this contradiction, *i.e.* we show how to get short individual random delays without having the cogs in the cumulative sum.

6.6 Improved Floating Mean

In the previous Section we have shown that parameters for the Floating Mean have to be chosen so that $b > \sqrt{N}$ to provide better distribution. Here we improve the method itself so that the parameters can be chosen in a wider range.

6.6.1 Description

In the original Floating Mean method an integer m is selected at random in $[0, a-b]$ before each new execution and the length of individual delays is then a random integer in $[m, m+b]$. The core idea of the new method we develop in this Section is to improve the granularity of random delay generation by using a wider distribution for m. More precisely, the improved method works as follows:

1. Initially the integer parameters a and b are chosen so that $b < a$; additionally we generate a non-negative integer parameter k.

2. Prior to each execution, we generate an integer value m' in the interval $[0, (a-b) \cdot 2^k[$.

3. Throughout the execution, the integer length d of an individual delay is obtained by first generating a random integer $d' \in [m', m' + (b+1) \cdot 2^k[$ and then letting $d \leftarrow \lfloor d' \cdot 2^{-k} \rfloor$.

6.6.2 Analysis

We see that as in the original Floating Mean method, the length of individual delays is in the interval $[0, a]$. Moreover if the integer m' generated at step 2 is such that $m' = 0 \mod 2^k$, then we can write $m' = m \cdot 2^k$ and the length d of individual delays is uniformly distributed in $[m, m+b]$ as in the original Floating Mean method.

When $m' \neq 0 \mod 2^k$, writing $m' = m \cdot 2^k + u$ with $0 \leq u < 2^k$, the delay's length d is distributed in the interval $[m, m+b+1]$ with a slightly non-uniform distribution:

$$\Pr[d=i]=\begin{cases}\frac{1}{b+1}\cdot(1-u\cdot 2^{-k}) & \text{for } i=m\\ \frac{1}{b+1} & \text{for } m+1\le i\le m+b\\ \frac{1}{b+1}\cdot u\cdot 2^{-k} & \text{for } i=m+b+1\end{cases}$$

Therefore when m' increases from $m\cdot 2^k$ to $(m+1)\cdot 2^k$ the distribution of the delay length d moves progressively from uniform in $[m,m+b]$ to uniform in $[m+1,m+b+1]$. In Section 6.6.3 below we show that for a fixed m':

$$\begin{aligned}\mathrm{E}[d] &= m'\cdot 2^{-k}+\frac{b}{2}\\ \mathrm{Var}[d] &= \mathrm{E}[d^2]-\mathrm{E}[d]^2=\frac{(b+1)^2-1}{12}+2^{-k}\cdot u\cdot(1-2^{-k}\cdot u)\end{aligned}$$

where $u=m' \mod 2^k$.

6.6.3 Distribution of delay's length d

We have:

$$\mathrm{E}[d]=\frac{1}{(b+1)2^k}\sum_{i=m'}^{m'+(b+1)\cdot 2^k-1}\lfloor i\cdot 2^{-k}\rfloor$$

Write $m'=m\cdot 2^k+u$ with $0\le u<2^k$. This gives:

$$\begin{aligned}\mathrm{E}[d] &= \frac{1}{(b+1)2^k}\sum_{i=m2^k+u}^{(m+b+1)2^k+u-1}\lfloor i\cdot 2^{-k}\rfloor\\ &= \frac{1}{(b+1)2^k}\left(\sum_{i=m2^k+u}^{(m+1)2^k-1}\lfloor i\cdot 2^{-k}\rfloor+\sum_{i=(m+1)2^k}^{(m+b+1)2^k-1}\lfloor i\cdot 2^{-k}\rfloor+\sum_{i=(m+b+1)2^k}^{(m+b+1)2^k+u-1}\lfloor i\cdot 2^{-k}\rfloor\right)\\ &= \frac{1}{(b+1)2^k}\left(m\cdot(2^k-u)+2^k\sum_{j=m+1}^{m+b}j+(m+b+1)\cdot u\right)\\ &= \frac{1}{(b+1)2^k}\left(m\cdot 2^k+(b+1)\cdot u+b\cdot 2^k\cdot\left(m+\frac{b+1}{2}\right)\right)\\ &= \frac{1}{(b+1)2^k}\left(m\cdot(b+1)\cdot 2^k+(b+1)\cdot u+b\cdot 2^k\cdot\frac{b+1}{2}\right)\\ &= m+u\cdot 2^{-k}+\frac{b}{2}=m'\cdot 2^{-k}+\frac{b}{2}\end{aligned}$$

Similarly we have:

$$\begin{aligned}
\mathrm{E}[d^2] &= \frac{1}{(b+1)2^k} \sum_{i=m2^k+u}^{(m+b+1)2^k+u-1} \lfloor i \cdot 2^{-k} \rfloor^2 \\
&= \frac{1}{(b+1)2^k} \left(\sum_{i=m2^k+u}^{(m+1)2^k-1} \lfloor i \cdot 2^{-k} \rfloor^2 + \sum_{i=(m+1)2^k}^{(m+b+1)2^k-1} \lfloor i \cdot 2^{-k} \rfloor^2 + \sum_{i=(m+b+1)2^k}^{(m+b+1)2^k+u-1} \lfloor i \cdot 2^{-k} \rfloor^2 \right) \\
&= \frac{1}{(b+1)2^k} \left(m^2 \cdot (2^k - u) + 2^k \sum_{j=m+1}^{m+b} j^2 + (m+b+1)^2 \cdot u \right)
\end{aligned}$$

After simplifications this gives:

$$\mathrm{Var}[d] = \mathrm{E}[d^2] - \mathrm{E}[d]^2 = \frac{(b+1)^2 - 1}{12} + 2^{-k} \cdot u(1 - 2^{-k}u)$$

6.6.4 Analysis continued

For a fixed m' the cumulative sum of random delay durations is the sum of N independently distributed random variables. Therefore it can be approximated by a Gaussian distribution with mean:

$$\mu_{m'} = N \cdot \mathrm{E}[d] = N \cdot \left(m' \cdot 2^{-k} + \frac{b}{2} \right) \tag{6.2}$$

and variance

$$V_{m'} = N \cdot \mathrm{Var}[d] = N \cdot \left(\frac{(b+1)^2 - 1}{12} + 2^{-k} \cdot u \cdot (1 - 2^{-k} \cdot u) \right)$$

For random $m' \in [0, (a-b) \cdot 2^k[$ the probability density of the cumulative sum can therefore be approximated by:

$$f(x) = \sum_{m'=0}^{(a-b)\cdot 2^k - 1} \frac{1}{(a-b)2^k \sigma_{m'} \sqrt{2\pi}} \exp\left(-\frac{(x - \mu_{m'})^2}{2\sigma_{m'}^2} \right)$$

where $\sigma_{m'} = \sqrt{V_{m'}}$. We have:

$$\sigma_{m'} > \sigma = \sqrt{N} \cdot \sqrt{\frac{(b+1)^2 - 1}{12}}$$

where σ is the same as for the original Floating Mean method.

As previously we obtain a multimodal distribution. The distance between the means of successive components is $\mu_{m'+1} - \mu_{m'} = N \cdot 2^{-k}$ and the standard deviation of a component is at least σ. Therefore the cogs become indistinguishable when $\sigma \gg N \cdot 2^{-k}$ which gives the condition:

$$b \gg \sqrt{N} \cdot 2^{-k}$$

instead of $b \gg \sqrt{N}$ for the original Floating Mean. Therefore by selecting a sufficiently large k we can accommodate a large number of short random delays (large N and small b). In practice, already for k as small as 3 the effect is considerable; we confirm this practically in Section 6.9.

We now proceed to compute the mean and variance of the cumulative sum for random m'. Let us denote by S_N the sum of the N delays. We have from (6.2):

$$\mathrm{E}[S_N] = \mathrm{E}[\mu_{m'}] = N \cdot \left(\frac{a}{2} - 2^{-k-1}\right)$$

which is the same as the original Floating Mean up to the 2^{-k-1} term.

To compute the standard deviation of S_N, we represent an individual delay as a random variable $d_i = m + v_i$ where $m = \lfloor m' \cdot 2^{-k} \rfloor$ and v_i is a random variable in the interval $[0, b+1]$. Since m' is uniformly distributed in $[0, (a-b) \cdot 2^k[$, the integer m is uniformly distributed in $[0, a-b[$; moreover the distribution of v_i is independent of m and the v_i's are identically distributed. From

$$S_N = \sum_{i=1}^{N} d_i = Nm + \sum_{i=1}^{N} v_i \, .$$

we get:

$$\mathrm{Var}(S_N) = N^2 \cdot \mathrm{Var}(m) + N \cdot \mathrm{Var}(v_1)$$

For large N we can neglect the term $N \cdot \mathrm{Var}(v_1)$ which gives:

$$\mathrm{Var}(S_N) \simeq N^2 \cdot \mathrm{Var}(m) = N^2 \cdot \frac{(a-b)^2 - 1}{12}$$

which is approximately the same as for the original Floating Mean method. As for the original Floating Mean the variance of the sum of N delays is in $\Theta\left(N^2\right)$ in comparison to plain uniform delays and the method of [156] that both have variances in $\Theta\left(N\right)$.

6.6.5 Illustration

The result is shown in Figure 6.6 compared to the original Floating Mean. The parameters of both methods were the same as in Figure 6.5: $a = 18$, $b = 3$ and $N = 32$. We take $k = 3$ for our Improved Floating Mean method. We can see that we have flattened out the cogs while keeping the same mean. This is because we still have $\sigma \simeq 6.3$ but the distance between successive cogs is now $N \cdot 2^{-k} = 32 \cdot 2^{-3} = 4$ instead of 32 so the cogs are now almost indistinguishable.

6.6.6 Full algorithm

The Improved Floating Mean method is formally defined by Algorithm 8. By $\mathcal{DU}[y, z[$ we denote discrete uniform distribution on $[y, z[$, $y, z \in \mathbb{Z}$, $y < z$. Note that as with the Floating Mean, we apply the technique of “flipping” the mean in the middle of the execution to make the duration of the entire execution independent of m'. In Section 6.8 we will show that Improved Floating Mean can be efficiently implemented on a constrained platform.

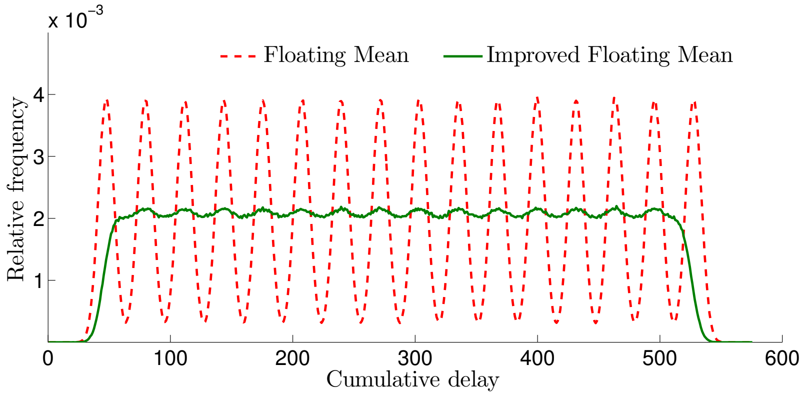


Figure 6.6: Improved Floating Mean with $k = 3$ compared to the original Floating Mean; $a = 18$, $b = 3$ and $N = 32$ for both methods.

Algorithm 8: Improved Floating Mean

```
Input: $a, b, k, M \in \mathbb{N}, b \leq a, N = 2M$
$m' \leftarrow \mathcal{DU}[0, (a-b) \cdot 2^k[$
for $i = 1$ to $N/2$ do
    $d_i \leftarrow \left\lfloor \left( m' + \mathcal{DU}\left[0, (b+1) \cdot 2^k\right[ \right) \cdot 2^{-k} \right\rfloor$
end for
for $i = N/2 + 1$ to $N$ do
    $d_i \leftarrow \left\lfloor \left( a \cdot 2^k - m' - \mathcal{DU}\left[0, (b+1) \cdot 2^k\right[ \right) \cdot 2^{-k} \right\rfloor$
end for
Output: $d_1, d_2, \ldots, d_N$
```

6.7 Criterion for efficiency of random delays

In this Section we address the problem of measuring and comparing the efficiency of different random delay generation methods. We demonstrate that the variance of the cumulative delay is an inappropriate measure of the security introduced by the countermeasure and can even be misleading. We suggest a criterion which is optimal with the respect to the desired properties of the countermeasure. The criterion is information-theoretically sound.

6.7.1 Variance as a bad measure of security

As already mentioned, in the work [156] the design criterion for the delay generation method was increasing the variance of the cumulative sum of the delays while decreasing its mean. In our initial work [44] we also followed this approach and suggested to use the coefficient of variation σ/μ to measure and compare the efficiency of the random delay generation methods[3], where a higher value of this ratio meant a better efficiency. This criterion has shown that the Floating Mean countermeasure has great advantage over the methods with independent individual delays since its variance grows quadratically with the number of delays in the cumulative sum, and so the coefficient of variation tends to a constant, as Table 6.1 shows. The theoretical expectations were roughly matching the experimental data that will be presented here in Section 6.9.

Table 6.1: Efficiency ratios σ/μ for different random delay generation methods

Plain uniform	Benoit-Tunstall	Floating Mean
$\frac{1}{\sqrt{3N}} = \Theta\left(\frac{1}{\sqrt{N}}\right)$	$\frac{\sigma_{\mathrm{BT}}}{\mu_{\mathrm{BT}}} \cdot \frac{1}{\sqrt{N}} = \Theta\left(\frac{1}{\sqrt{N}}\right)$	$\frac{\sqrt{N((a-b+1)^2-1)+b^2+2b}}{a\sqrt{3N}} = \Theta\left(1\right)$

However, the correct criterion for the security introduced by the countermeasure is the amount of *uncertainty* introduced by the cumulative delay. We will show below that the number of traces required for a side-channel attack is directly linked to the uncertainty introduced by the countermeasure. The standard deviation σ is in general a very rough way to estimate the uncertainty and therefore the security of the countermeasure. It works well only for very similar distributions like two approximately normal distributions, as is the case of plain uniform delays and the method of Benoit and Tunstall. We have already shown in the analysis of Floating Mean in Section 6.5 that for a complex distribution the variance does not reflect the real security against side-channel attacks.

We further illustrate this with a simple example. Consider the uniform distribution U of integers on some interval $[a,b], a,b \in \mathbb{Z}$, and the distribution X with $\Pr[X = a] = \Pr[X = b] = 1/2$. We have $\mathrm{Var}(U) = ((a-b+1)^2-1)/12$ and $\mathrm{Var}(X) = (a-b)^2/4$, so $\mathrm{Var}(X) > \mathrm{Var}(U)$. Therefore the efficiency of X counted in σ/μ will be higher than for U. But with X the DPA signal is only divided by 2 instead of $(b-a+1)$ with U, so the number of traces required to break an implementation with U will be smaller than with X. So in this case the criterion based on the variance is misleading.

6.7.2 Recalling the effect of timing disarrangement

An accurate estimate for the security introduced by the random delay countermeasure is the maximum $\hat{p}$ of the probability mass function (p.m.f.)[4] of the distribution of the cumulative sum.

From [39] and [100] we recall that the number of traces T required for a DPA attack is determined by the maximal correlation coefficient ρ_{max} observed in the correlation trace for the correct key guess. Namely, the number of traces can be

---

[3] Here σ is the standard deviation of the cumulative sum across various executions, as opposed to Sections 6.5 and 6.6 where σ was the standard deviation for a single execution with a fixed m

[4] and not p.d.f. since the distribution is discrete

estimated as

$$T = 3 + 8\left(\frac{Z_\alpha}{\ln\left(\frac{1+\rho_{max}}{1-\rho_{max}}\right)}\right)^2 \tag{6.3}$$

where Z_α is a quantile of a normal distribution for the 2-sided confidence interval with error $1-\alpha$. Since for $x < 0.2$, $\ln\left(\frac{1+x}{1-x}\right) \approx 2x$ holds, we can approximate (6.3) for $Z_{\alpha=0.9} = 1.282$ as

$$T \approx 3/\rho_{max}^2$$

when $\rho_{max} < 0.2$. So the number of traces is in ρ_{max}^{-2}. In turn, the effect of the timing disarrangement on ρ_{max} is in $\hat{p}$ in case no integration is used. So the number of traces is in $1/\hat{p}^2$.

6.7.3 The new criterion

We propose a better criterion for the efficiency E of the random delays countermeasure:

$$E = 1/(2\hat{p} \cdot \mu)$$

where $\hat{p}$ is the maximum of the probability mass function and μ is the mean of the distribution of the cumulative sum of the delays. This criterion is normalized and *optimal* with respect to the desired properties of the countermeasure, as we show below.

With the countermeasure, we want to maximize the number of traces in an attack, *i.e.* minimize $\hat{p}$, while keeping the smallest possible overhead, *i.e.* smallest mean μ. One can see that from all distributions with the given $\hat{p}$, the one with the smallest μ is uniform on $[0, 1/\hat{p}]$. In this case, $\mu = 1/(2\hat{p})$ and the criterion E is equal to 1. In all other cases (same $\hat{p}$ but larger μ) the value of the criterion will be smaller, and the closer to zero – the farther is the distribution from the optimal one (*i.e.* the uniform one).

This tightly relates the criterion to the entropy of the distribution. Namely, the new criterion is directly linked to min-entropy, which is defined for a random variable S as

$$H_\infty(S) = -\log \max_i p_i = -\log\, \hat{p}.$$

Note that $H_\infty(S) \le H(S)$, where $H(S) = -\sum_i p_i \log p_i$ is the Shannon entropy, so min-entropy can be considered as a worst-case measure of uncertainty. Now we have $\hat{p} = 2^{-H_\infty(S)}$ and the new efficiency criterion is expressed as

$$E = \frac{2^{H_\infty(S)-1}}{\mu}.$$

Indeed, for a fixed worst-case cumulative delay, the distributions with the higher entropy, *i.e.* maximizing uncertainty for the attacker, will have lower $\hat{p}$, larger number of traces to attack and thus more efficient as a countermeasure.

This criterion is easily computable once the designer have simulated the real distribution for the concrete parameters of a method (taking into consideration the number of clock cycles per delay loop) and obtained $\hat{p}$.

6.8 Implementation for an 8-bit AVR platform

Here we show that our new method can be efficiently implemented on an embedded μC. We present the reference implementation of different delay generation methods for 8-bit AVR μCs. The implementation is written in the 8-bit AVR assembly language. We tested it on the ATmega16 μC, using Atmel AVR Studio 4 as a development environment.

In our implementation we make use of the fact that we can efficiently produce uniform random integers in the interval $[0, 2^i - 1]$ by a bit-wise `AND` of an 8-bit uniform random value with a bit mask $2^i - 1$, $i = 1, 2, ..., 8$.

Throughout the code, the following registers are reserved:

`RND` for obtaining the random numbers and delay durations;

`FM` for storing the value of m used in the Floating Mean and in the Improved Floatimg Mean during the execution;

`MASKB` for the bitmask applied to the delay length;

`MASKM` for the bitmask applied to obtain the value of m in Floating Mean;

`TMP` temporary helper register.

RNG simulation. Common 8-bit AVR μCs do not have a built-in RNG. Hence, we have simulated the RNG by pre-loading a pool of pseudorandom numbers to μC's SRAM from the host PC prior to each execution and pointing the `X` register at the beginning of the pool. Random numbers are then loaded successively from SRAM to `RND` register by calling the `randombyte` function:

```
randombyte:
    ld  RND, X+           ; X is the dedicated address register
    ret                   ;  that is used only in this function
```

Basic delay generation routine. First, here is the basic delay generation routine. It produces delays of length $3 \cdot \texttt{RND} + C$ cycles, where C is the constant overhead per delay. To reduce this overhead, the delay generation can be implemented inline to avoid the cost of entering and leaving the function (the same can be done with the `randombyte` function). So, each delay is a multiple of 3 μC cycles; this granularity cannot be further reduced for this platform.

```
randomdelay:
    rcall randombyte      ; obtain a random byte in RND
    ; <placeholder for method-specific code>
    tst   RND             ; balancing the cycle length between
    breq  zero            ;  zero and non-zero delay values
    nop
    nop
dummyloop:
    dec   RND
    brne  dummyloop
zero:
    ret
```

Since the conditional instructions `breq` (branch if equal) and `brne` (branch if not equal) of the 8-bit AVR take different amount of cycles depending on whether the condition is satisfied or not, the loop code is specially arranged to balance the case when the delay duration value is zero. Otherwise, the resulting delay distribution would be biased from the original one.

The part of the code specific for delay generation methods is omitted (the position in the above code is marked by a placeholder) and will be given below.

Plain uniform delays. The method-specific code part for plain uniform delays is just a single instruction:

```
and   RND, MASKB      ; truncate random value to the desired length
```

Method of Benoit and Tunstall. For the method of Benoit and Tunstall, the delay value is generated as follows:

```
ldi ZH, high(bttable) ; load the table address high byte
mov ZL, RND           ; load the table address low byte (lookup index)
ld RND, Z             ; perform the lookup
```

Here `bttable` is the table of 256 byte entries with the inverse c.d.f. of the pit-shaped distribution that is pre-loaded into SRAM along with the pool of random numbers. The table used in our experiments reported in Section 6.9 is given in Table 6.2. Note that the table should be aligned with the 256-byte boundary to facilitate indexing as it is done in the above code snippet.

Table 6.2: Tabulated inverse c.d.f. of the pit-shaped distribution, in hexadecimal notation

```
00,00,00,00,00,00,00,00,00,00,00,00,00,00,00,00,00,00,00,00,00,00,00,00,
00,00,00,00,00,00,00,00,00,00,00,00,00,00,00,00,00,01,01,01,01,01,01,01,
01,01,01,01,01,01,01,01,01,01,01,01,01,01,01,01,01,01,01,01,01,01,02,02,
02,02,02,02,02,02,02,02,02,02,02,02,02,02,02,02,02,02,03,03,03,03,03,03,
03,03,03,03,03,03,03,03,04,04,04,04,04,04,04,04,04,04,05,05,05,05,05,05,
05,06,06,06,06,06,06,07,07,07,07,08,08,08,09,09,09,0a,0a,0a,0b,0b,0b,0c,
0c,0c,0c,0d,0d,0d,0d,0d,0e,0e,0e,0e,0e,0e,0f,0f,0f,0f,0f,0f,0f,0f,0f,10,
10,10,10,10,10,10,10,10,10,10,10,11,11,11,11,11,11,11,11,11,11,11,11,11,
11,11,11,11,12,12,12,12,12,12,12,12,12,12,12,12,12,12,12,12,12,12,12,12,
12,12,12,12,13,13,13,13,13,13,13,13,13,13,13,13,13,13,13,13,13,13,13,13,
13,13,13,13,13,13,13,13,13,13,13,13,13,13,13,13
```

(Improved) Floating Mean. The method-specific code for the Improved Floating Mean includes addition of the value m and then “squeezing” the range by shifting the register with the delay right by k bits.

```
and   RND, MASKB      ; truncate to the desired length including k
add   RND, FM         ; add 'floating mean'
```

```
lsr   RND             ;
...                   ; logical shit right by k bits
lsr   RND             ;
```

The value of k is hard-coded by k right shift instructions since there are no multiple-bit shift instructions in the AVR instruction set. Note that when $k = 0$ the method is just Floating Mean and the code is just 1 single-cycle instruction longer than in plain uniform delays.

Floating Mean also requires initialization (namely, generation of m) in the beginning of each execution:

```
rcall randombyte      ; obtain a random byte in RND
and RND, MASKM        ; truncate 'foating mean' to the desired length
mov FM, RND           ; store 'floating mean' on register FM
```

and "flipping" `FM` in the middle of the execution to make the total execution length independent of the value of m.

```
mov TMP, MASKM
sub TMP, FM
mov FM, TMP
```

Practical parameter choice. Here we explain how to translate the delay generation method parameters a and b used in the previous Sections into the bitmasks `MASKB` and `MASKM`.

For the plain uniform delays, the bitmask for the delay duration is set as

$$\texttt{MASKB} = 0 \ldots 0 \underbrace{1 \ldots 1}_{t}$$

With this mask, the delay durations will be generated in the interval $[0, a]$, $a = 2^t - 1$. For example, for our experiments we have chosen $t = 4$ which yields $a = 15$. In general, for a device with n-bit registers $t \leq n$ should hold.

For the Floating Mean and its improved version, the bitmasks for the delay duration and for the value of m have the following form:

$$\texttt{MASKB} = 0 \ldots 0 \underbrace{1 \ldots 1}_{t} \underbrace{1 \ldots 1}_{k}$$
$$\texttt{MASKM} = 0 \ldots 0 \underbrace{1 \ldots 1}_{s} \underbrace{1 \ldots 1}_{k}$$

In the case of Floating Mean (when $k = 0$), this yields $b = 2^t - 1$ and $a - b = 2^s - 1$. In the case of Improved Floating Mean (when $k > 0$), this yields $b = 2^t - 1$ but $a - b = 2^s$, which is slightly different and consistent with the Improved Floating Mean description. For example, for Floating Mean we have chosen $s = 4$ and $t = 2$ which yields $a = 18$ and $b = 3$. For the Improved Floating Mean, we have chosen $s = 4$, $t = 2$ and $k = 3$ which yield $a = 19$ and $b = 3$. To ensure that the operations are performed on a single register and no overflow occurs on an n-bit μC, s, t, and k here should be chosen such that $\max(s, t) + k + 1 \leq n$.

Comparison. It can be seen that Improved Floating Mean is more “lightweight” in terms of both memory and code than the table method of Benoit and Tunstall. Our method requires no additional memory, as opposed to [156], and can take less cycles for the computation of delay duration. Compared to the plain uniform delays, our Floating Mean requires $1+k$ more cycles for the computation of delay duration (the duration of the single delay loop iteration is the same for all methods in our implementation), the additional generation of the m value and “flipping” it once per execution; thus the overhead is negligible.

6.9 Resistance to practical attacks

Here we present comparison between the practical implementations of plain uniform delays, table method of Benoit and Tunstall [156] and the new Floating Mean method by mounting a Correlation Power Analysis (CPA) attack [31] against them. We applied the random delays to protect AES encryption implemented on an ATmega16 μC.

6.9.1 Protected AES implementation

Random delays were introduced into AES-128 encryption. The delay generation was implemented as described in Section 6.8. We put 10 delays per each round: before `AddRoundKey`, 4 per `SubBytes+ShiftRows`, before each `MixColumn` and after `MixColumns`. 3 “dummy” AES rounds that also incorporated random delays were added in the beginning and in the end of the encryption. Thus, the first `SubByte` operation of the first encryption round, which is the target for our attacks, is separated from the start of the execution, which is in turn our synchronization point, by $N = 32$ random delays.

The parameters of the methods were chosen to ensure (nearly) the same performance overhead across the methods. They were made sufficiently small to enable attacks with a reasonable number of traces; for a real application the values can be larger.

- For the plain uniform delays (PU), the individual delay values were generated on $[0, 15]$.
- For the table method of Benoit and Tunstall, the p.d.f. of the pit-shaped distribution was generated using the formula $y = \lceil ak^x + bk^{M-x} \rceil$ from [156] with the parameters $M = 19$, $a = 40$, $b = 34$ and $k = 0.7$. These parameters were chosen so that they lead to the table of 256 entries with the inverse c.d.f. of the distribution (see Table 6.2 in Section 6.8). We use this table to produce delay values on $[0, 19]$ by indexing it with a random byte.
- For Floating Mean (FM) we used $a = 18$, $b = 3$.
- For Improved Floating Mean (IFM), the parameters were $a = 19$, $b = 3$, $k = 3$. The value of a is different from the one in FM, as discussed in Section 6.8, so

the mean for IFM is larger that for FM, but still the estimated efficiency is 1.8 times higher.

Due to the implementation efficiency considerations we could not make the performance overhead for all the methods to be exactly the same, but it is close enough for comparison.

6.9.2 CPA attack

We mounted a CPA attack [31] in the Hamming weight power consumption model against the first AES key byte for each of the methods, first `SubByte` operation being the attack target. As a reference benchmark for our measurement conditions we performed a CPA attack against the implementation without random delays. For implementations with random delays, we used power consumption traces *as is* without any alignment or integration to make a consistent comparison. Figure 6.7 presents CPA attack results for the 4 methods. The correlation coefficient evolution plot is not presented for Improved Floating Mean since even with 150000 traces the attack did not succeed.

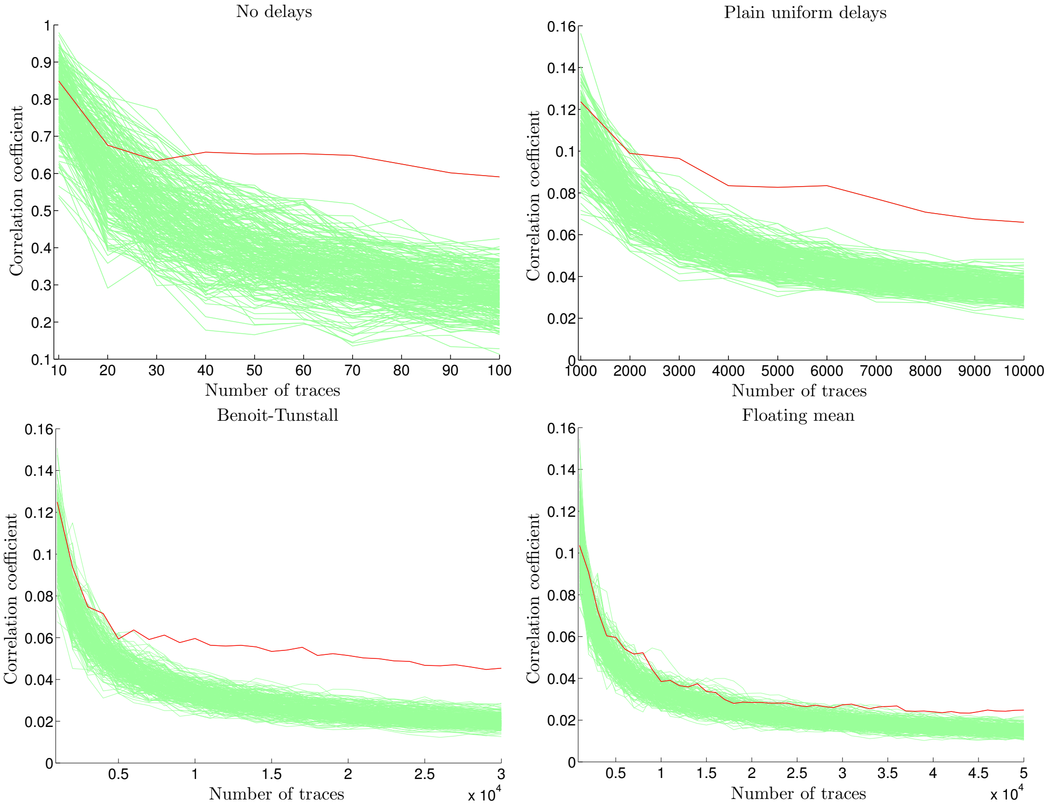


Figure 6.7: CPA against random delays: correlation coefficient for all key byte guesses against the number of power consumption traces. The trace for the correct guess is highlighted.

It can be seen that our new methods are more secure in practice already for small delay durations and for a small number of delays. To break our implementation, we require 45000 traces for Floating Mean and 7000 traces for the method of Benoit and Tunstall. That is, for the *same performance penalty* the Floating Mean method requires 6 times more curves to be broken. The Improved Floating Mean we require $> 150 \cdot 10^3$ traces to be broken, which is > 20 times more than for the method of Benoit and Tunstall for the same performance overhead. This ratio will increase with the number of delays. This is important for symmetric algorithm implementations that are relatively short. For inherently long implementations of public key algorithms the number of delays in the sum will be naturally large.

6.9.3 Comparing efficiency

In our example in Sect. 6.6.5, with the Improved Floating Mean method we have decreased $\hat{p}$ by a factor 2, as illustrated in Figure 6.6. So the number of traces for the successful straightforward CPA attack must be in principle almost 4 times larger (around $160 \cdot 10^3$), and according to the optimal criterion the efficiency must be almost 2 times higher. This is what we observed in practice: we could not break the Improved Floating Mean with these parameters with $150 \cdot 10^3$ traces.

Table 6.3 presents the values of the new efficiency criterion E we introduced for random delays in Section 6.7 and the practically observed attack measurement complexities from Figure 6.7. The values for E were computed from the maximum $\hat{p}$ of the p.m.f. and the mean μ of the cumulative delay (the distributions were shown in Figures 6.5 and 6.6), taking into consideration the real granularity of the delay loop (3 cycles).

Table 6.3: New efficiency criterion for different random delay generation methods

	ND	PU	BT [156]	FM	IFM
μ, cycles	0	720	860	862	953
$\hat{p}$	1	0.0144	0.0092	0.0040	0.0020
$E = 1/(2\hat{p}\mu)$	–	0.048	0.063	0.145	0.259
T_{CPA}, traces	50	2500	7000	45000	> 150000

Due to its definition, the new criterion reflects well the number of traces observed in the experimental attack. For example, looking at the new criterion we expect the number of traces for the Floating Mean be $0.145^2/0.063^2 = 5.3$ times higher than for the table method of Benoit and Tunstall [156]. In the experiment, it was $45000/7000 = 6.4$ times higher.

6.10 Conclusion

In this Chapter we proposed a new method for random delay generation in embedded software—the Floating Mean method. We suggested how to choose the parameters of the method so that it generates a good distribution; however this requires to generate longer delays while in practice it is preferable to have multiple shorter delays. We proposed the Improved Floating Method that allows for a wider choice of parameters.

We suggested an information-theoretically sound criterion for measuring the efficiency of the random delays countermeasure. The criterion reflects how much traces are requires to break an implementation relatively to the performance overhead introduced by the countermeasure

We presented the lightweight implementation of our method for protection of AES encryption on an 8-bit AVR platform. Finally, we mounted practical DPA attacks showing that for the same level of performance the reference implementation of the new method requires significantly more power curves to be broken compared to the existing methods. Thus, our method is significantly more efficient and secure.

In an independent work [98], our Floating Mean method was used to counteract a frequency-based DPA attack; in this case its security was still higher than that of the other methods.

Bibliography

[1] Onur Acıiçmez, Çetin Kaya Koç, and Jean-Pierre Seifert. Predicting secret keys via branch prediction. In *CT-RSA*, volume 4377 of *LNCS*, pages 225–242. Springer, 2007. 10

[2] Onur Acıiçmez and Çetin Kaya Koç. Trace-driven cache attacks on AES (short paper). In *ICICS'06*, volume 4307 of *LNCS*, pages 112–121. Springer, 2006. 51, 53, 61, 70, 72

[3] Frederic Amiel, Christophe Clavier, and Michael Tunstall. Fault analysis of DPA-resistant algorithms. In *FDTC'06*, volume 4236 of *LNCS*, pages 223–236. Springer, 2006. 17, 114

[4] Frederic Amiel, Karine Villegas, Benoit Feix, and Louis Marcel:. Passive and active combined attacks: Combining fault attacks and side channel analysis. In *FDTC*, pages 92–102. IEEE Computer Society, 2007. 8

[5] Julien Doget an Emmanuel Prouff, Matthieu Rivain, and François-Xavier Standaert. Univariate side channel attacks and leakage modeling. Proceedings of COSADE 2011, 2011. 13

[6] Cédric Archambeau, Eric Peeters, François-Xavier Standaert, and Jean-Jacques Quisquater. Template attacks in principal subspaces. In *CHES'06*, volume 4249 of *LNCS*, pages 1–14. Springer, 2006. 23

[7] ARM Ltd. Processors. `http://www.arm.com/products/processors/`, 2010. 52

[8] Atmega128 datasheet. `http://www.atmel.com/dyn/resources/prod_documents/doc2467.pdf`. 104

[9] Hagai Bar-El, Hamid Choukri, David Naccache, Michael Tunstall, and Claire Whelan. The sorcerer's apprentice guide to fault attacks. In *Proceedings of the IEEE*, volume 94, pages 370–382, 2006. 16

[10] Lejla Batina, Benedikt Gierlichs, and Kerstin Lemke-Rust. Comparative evaluation of rank correlation based DPA on an AES prototype chip. In *ISC 2008*, volume 5222 of *LNCS*, pages 341–354. Springer, 2008. 79, 83, 84

[11] Lejla Batina, Benedikt Gierlichs, and Kerstin Lemke-Rust. Differential cluster analysis. In *CHES'09*, volume 5747 of *LNCS*, pages 112–127. Springer, 2009. 14, 24

[12] Lejla Batina, Benedikt Gierlichs, Emmanuel Prouff, Matthieu Rivain, François-Xavier Standaert, and Nicolas Veyrat-Charvillon. Mutual information analysis: a comprehensive study. *Journal of Cryptology*, 24(2):269–291, 2011. 23, 48

[13] Arthur O. Bauer. Some aspects of military line communication as deployed by the German armed forces prior to 1945. `http://www.cdvandt.org/Wirecomm99.pdf`, 2004. 4

[14] Mihir Bellare and Phillip Rogaway. The exact security of digital signatures: How to sign with RSA and Rabin. In *EUROCRYPT'96*, volume 1070 of *LNCS*, pages 399–416. Springer, 1996. 92

[15] Daniel J. Bernstein. Cache-timing attacks on AES. `http://cr.yp.to/antiforgery/cachetiming-20050414.pdf`, 2004. 14, 50, 53

[16] Guido Bertoni, Vittorio Zaccaria, Luca Breveglieri, Matteo Monchiero, and Gianluca Palermo. AES power attack based on induced cache miss and countermeasure. In *ITCC'05*, volume 1, pages 586–591. IEEE, 2005. 14, 51, 53, 75

[17] Alexandre Berzati, Cécile Canovas, Jean-Guillaume Dumas, and Louis Goubin. Fault attacks on RSA public keys: Left-to-right implementations are also vulnerable. In *CT-RSA*, volume 5473 of *LNCS*, pages 414–428. Springer, 2009. 17

[18] Bigdigits multiple-precision arithmetic source code, version 2.2. `http://www.di-mgt.com.au/bigdigits.html`. 104

[19] Eli Biham, Yaniv Carmeli, and Adi Shamir. Bug attacks. In *CRYPTO'08*, pages 221–240, 2008. 8

[20] Eli Biham and Adi Shamir. *Differential Cryptanalysis of the Data Encryption Standard*. Springer, 1993. 16

[21] Eli Biham and Adi Shamir. Differential fault analysis of secret key cryptosystems. In *CRYPTO'97*, volume 1294 of *LNCS*, pages 513–525. Springer, 1997. 16

[22] Alex Biryukov, Andrey Bogdanov, Dmitry Khovratovich, and Timo Kasper. Collision attacks on Alpha-MAC and other AES-based MACs. In *CHES'07*, volume 4727 of *LNCS*, pages 166–180. Springer, 2007. 26

[23] Alex Biryukov and Dmitry Khovratovich. Two new techniques of side-channel cryptanalysis. In *CHES'07*, volume 4727 of *LNCS*, pages 195–208. Springer, 2007. 26

[24] Andrey Bogdanov. Improved side-channel collision attacks on AES. In *SAC'07*, volume 4876 of *LNCS*, pages 84–95. Springer, 2007. 24, 25, 26, 27, 28, 51, 53

[25] Andrey Bogdanov. Multiple-differential side-channel collision attacks on AES. In *CHES'08*, volume 5154 of *LNCS*, pages 30–44. Springer, 2008. 24, 26, 28, 48

[26] Andrey Bogdanov, Thomas Eisenbarth, Christof Paar, and Malte Wienecke. Differential cache-collision timing attacks on AES with applications to embedded CPUs. In *CT-RSA'10*, volume 5985 of *LNCS*, pages 235–251. Springer, 2010. 53

[27] Andrey Bogdanov and Ilya Kizhvatov. Beyond the limits of DPA: Combined side-channel collision attacks. To appear in *IEEE Transactions on Computers*, 2011. Preprint available at `http://eprint.iacr.org/2010/590`. 18, 21

[28] Andrey Bogdanov, Ilya Kizhvatov, and Andrey Pyshkin. Algebraic methods in side-channel collision attacks and practical collision detection. In *INDOCRYPT 2008*, volume 5365 of *LNCS*, pages 251–265, 2008. 18, 21, 24, 26, 28, 37, 39, 46, 66

[29] Dan Boneh, Richard A. DeMillo, and Richard J. Lipton. On the importance of checking cryptographic protocols for faults. *Journal of Cryptology*, 14:101–119, 2001. 5, 16, 91, 92, 93, 94

[30] Joseph Bonneau. Robust final-round cache-trace attacks against AES. Cryptology ePrint Archive, Report 2006/374, 2006. `http://eprint.iacr.org/2006/374`. 51, 52, 53, 55, 59, 60, 68, 75

[31] Eric Brier, Christophe Clavier, and Olivier Benoit. Correlation power analysis with a leakage model. In *CHES 2004*, volume 3156 of *LNCS*, pages 135–152. Springer, 2004. 13, 23, 24, 30, 46, 50, 82, 131, 132

[32] Julien Brouchier, Nora Dabbous, Tom Kean, Carol Marsh, and David Naccache. Thermocommunication. Cryptology ePrint Archive, Report 2009/002, 2009. `http://eprint.iacr.org/`. 9

[33] Marco Bucci, Raimondo Luzzi, Michele Guglielmo, and Alessandro Trifiletti. A countermeasure against differential power analysis based on random delay insertion. In *IEEE International Symposium on Circuits and Systems – ISCAS 2005*, volume 4, pages 3547–3550, May 2005. 114

[34] Çetin Kaya Koç, editor. *Cryptographic Engineering.* Springer, 2009. 6, 9

[35] Suresh Chari, Josyula R. Rao, and Pankaj Rohatgi. Template attacks. In *CHES'02*, volume 2523 of *LNCS*, pages 51–62. Springer, 2003. 14, 23, 24

[36] Xavier Charvet and Herve Pelletier. Improving the DPA attack using wavelet transform. NIST Physical Security Testing Workshop, 2005. 46

[37] Christophe Clavier. An improved SCARE cryptanalysis against a secret A3/A8 GSM algorithm. In *ICISS*, pages 143–155, 2007. 87

[38] Christophe Clavier. Secret external encodings do not prevent transient fault analysis. In *CHES*, volume 4727 of *LNCS*, pages 181–194. Springer, 2007. 17

[39] Christophe Clavier, Jean-Sébastien Coron, and Nora Dabbous. Differential power analysis in the presence of hardware countermeasures. In *CHES'2000*, volume 1965 of *LNCS*, pages 252–263. Springer, 2000. 13, 111, 112, 114, 119, 126

[40] Christophe Clavier, Benoit Feix, Georges Gagnerot, and Mylène Roussellet. Passive and active combined attacks on AES: Combining fault attacks and side channel analysis. In *FDTC*, pages 10–19. IEEE Computer Society, 2010. 8

[41] Don Coppersmith. Small solutions to polynomial equations, and low exponent vulnerabilities. *Journal of Cryptology*, 10:233–260, 1997. 94, 95

[42] Jean-Sébastien Coron. Optimal security proofs for PSS and other signature schemes. In *EUROCRYPT'02*, volume 2332 of *LNCS*, pages 272–287. Springer, 2002. 93

[43] Jean-Sébastien Coron, Antoine Joux, Ilya Kizhvatov, David Naccache, and Pascal Paillier. Fault attacks on RSA signatures with partially unknown messages. In *CHES 2009*, volume 5747 of *LNCS*, pages 444–456. Springer, 2009. Full version available at `http://eprint.iacr.org/2009/309`. 19, 91

[44] Jean-Sébastien Coron and Ilya Kizhvatov. An efficient method for random delay generation in embedded software. In *CHES 2009*, volume 5747 of *LNCS*, pages 156–170. Springer, 2009. Extended version available at `http://eprint.iacr.org/2009/419`. 19, 109, 118, 125

[45] Jean-Sébastien Coron and Ilya Kizhvatov. Analysis and improvement of the random delay countermeasure of CHES 2009. In *CHES 2010*, volume 6225 of *LNCS*, pages 95–109. Springer, 2010. 19, 109

[46] Jean-Sébastien Coron, David Naccache, and Julien P. Stern. On the security of RSA padding. In *CRYPTO'99*, volume 1666 of *LNCS*, pages 1–18. Springer, 1999. 94

[47] Jean-Sébastien Coron, David Naccache, Mehdi Tibouchi, and Ralf-Philipp Weinmann. Practical cryptanalysis of ISO/IEC 9796-2 and EMV signatures.

In *CRYPTO'09*, volume 5677 of *LNCS*, pages 428–444. Springer, 2009. Full version: `http://eprint.iacr.org/2009/203.pdf`. 94

[48] Jean-Christophe Courrège, Benoit Feix, and Mylène Roussellet. Simple power analysis on exponentiation revisited. In *Smart Card Research and Advanced Application*, volume 6035 of *LNCS*, pages 65–79. Springer, 2010. 13

[49] Joan Daemen and Vincent Rijmen. *The Design of Rijndael: AES – The Advanced Encryption Standard*. Springer, 2002. 52, 53, 55

[50] Benoît Dupasquier, Stefan Burschka, Kieran McLaughlin, and Sakir Sezer. Analysis of information leakage from encrypted Skype conversations. *International Journal of Information Security*, 9(5):313–325, 2010. 6

[51] Stefan Dziembowski and Krzysztof Pietrzak. Leakage-resilient cryptography. In *FOCS*, pages 293–302. IEEE Computer Society, 2008. 15

[52] Thomas Eisenbarth, Tim Güneysu, Stefan Heyse, and Christof Paar. MicroEliece: McEliece for embedded devices. In *CHES 2009*, volume 5747 of *LNCS*, pages 49–64. Springer, 2009. 78

[53] EMV integrated circuit card specifications for payment systems, book 2. security and key management. version 4.2. `http://www.emvco.com`, June 2008. 93

[54] François-Xavier Standaert et al. Electromagnetic analysis and fault attacks: State of the art. ECRYPT deliverable D.VAM.4, 2005. Available at `http://www.ecrypt.eu.org/ecrypt1/documents/D.VAM.4-3.pdf`. 9

[55] Nigel Smart et al. ECRYPT II yearly report on algorithms and keysizes (2009-2010). ECRYPT II deliverable, 2010. Revision 1.0, available at `http://www.ecrypt.eu.org/documents/D.SPA.13.pdf`. 7

[56] Julie Ferrigno and Martin Hlaváĉ. When AES blinks: introducing optical side channel. *IET Information Security*, 2(3):94–98, 2008. 9

[57] Richard P. Feynman and Ralph Leighton. *Surely You're Joking, Mr. Feynman!: Adventures of a Curious Character*. W. W. Norton, 1985. 4

[58] Jacques Fournier and Michael Tunstall. Cache based power analysis attacks on AES. In *ACISP'06*, volume 4058 of *LNCS*, pages 17–28. Springer, 2006. 49, 51, 53, 55, 56, 58, 59, 60, 61, 67, 74, 75

[59] Jean-François Gallais and Ilya Kizhvatov. Error-tolerance in trace-driven cache collision attacks. Proceedings of COSADE 2011, 2011. 18, 49

[60] Jean-François Gallais, Ilya Kizhvatov, and Michael Tunstall. Improved trace-driven cache-collision attacks against embedded AES implementations. In *WISA 2010*, volume 6513 of *LNCS*, pages 243–257. Springer, 2011. Extended version available at `http://eprint.iacr.org/2010/408`. 18, 49

[61] Karine Gandolfi, Christophe Mourtel, and Francis Olivier. Electromagnetic analysis: Concrete results. In *CHES'01*, volume 2162 of *LNCS*, pages 251–261. Springer, 2001. 50

[62] Catherine H. Gebotys. *Security in Embedded Devices*. Springer, 2010. 6

[63] Benedikt Gierlichs, Lejla Batina, Pim Tuyls, and Bart Preneel. Mutual information analysis. In *CHES'08*, volume 5154 of *LNCS*, pages 426–442. Springer, 2008. 14, 23, 24

[64] Benedikt Gierlichs, Kerstin Lemke-Rust, and Christof Paar. Templates vs. stochastic methods. In *CHES'06*, volume 4249 of *LNCS*, pages 15–29. Springer, 2006. 23

[65] Philipp Grabher, Johann Großschädl, and Dan Page. Non-deterministic processors: FPGA-based analysis of area, performance and security. In *WESS*. ACM, 2009. 15

[66] Johann Großschädl, Elisabeth Oswald, Dan Page, and Michael Tunstall. Side-channel analysis of cryptographic software via early-terminating multiplications. In *ICISC*, volume 5984 of *LNCS*, pages 176–192. Springer, 2010. 10

[67] Jorge Guajardo and Bart Mennink. On side-channel resistant block cipher usage. In *ISC*, volume 6531 of *LNCS*, pages 254–268. Springer, 2010. 15

[68] Peter Gutmann. Data remanence in semiconductor devices. 10th USENIX Security Symposium, 2001. 8

[69] Helena Handschuh, Pascal Paillier, and Jacques Stern. Probing attacks on tamper-resistant devices. In *CHES*, volume 1717 of *LNCS*, pages 303–315. Springer, 1999. 8

[70] Helena Handschuh and Bart Preneel. Blind differential cryptanalysis for enhanced power attacks. In *SAC'06*, volume 4356 of *LNCS*, pages 163–173. Springer, 2006. 26

[71] Neil Hanley, Michael Tunstall, and William P. Marnane. Unknown plaintext template attacks. In *WISA*, volume 5932 of *LNCS*, pages 148–162. Springer, 2009. 14

[72] Mathias Herrmann and Alexander May. Solving linear equations modulo divisors: On factoring given any bits. In *ASIACRYPT'08*, volume 5350 of *LNCS*, pages 406–424. Springer, 2008. 94, 95, 98, 99

[73] Nicholas Howgrave-Graham. Finding small roots of univariate modular equations revisited. In *Cryptography and Coding*, volume 1355 of *LNCS*, pages 131–142. Springer, 1997. 98

[74] Nicholas Howgrave-Graham. Approximate integer common divisors. In *CALC*, pages 51–66, 2001. 100

[75] ISO/IEC 9796-2, Information technology – Security techniques – Digital signature scheme giving message recovery, Part 2: Mechanisms using a hash-function, 1997. 93, 94

[76] ISO/IEC 9796-2:2002 Information technology – Security techniques – Digital signature schemes giving message recovery – Part 2: Integer factorization based mechanisms, 2002. 94

[77] Marc Joye, Arjen Lenstra, and Jean-Jacques Quisquater. Chinese remaindering cryptosystems in the presence of faults. *Journal of Cryptology*, 21:27–51, 1999. 92

[78] John Kelsey, Bruce Schneier, David Wagner, and Chris Hall. Side channel cryptanalysis of product ciphers. *Journal of Computer Security*, 8:141–158, 2000. 50, 53

[79] Auguste Kerckhoffs. La cryptographie militaire. *Journal des sciences militaires*, IX:5–38,161–191, 1883. 6

[80] Chong Hee Kim. Differential fault analysis of AES: Toward reducing number of faults. Cryptology ePrint Archive, Report 2011/178, 2011. `http://eprint.iacr.org/`. 16

[81] Chong Hee Kim, Philippe Bulens, Christophe Petit, and Jean-Jacques Quisquater. Fault attacks on public key elements: Application to DLP-based schemes. In *EuroPKI*, volume 5057 of *LNCS*, pages 182–195. Springer, 2008. 17

[82] Chong Hee Kim and Jean-Jacques Quisquater. Fault attacks for CRT based RSA: new attacks, new results, and new countermeasures. In *WISTP 2007*, volume 4462 of *LNCS*, pages 215–228. Springer, 2007. 17, 104

[83] Chong Hee Kim, Jong Hoon Shin, Jean-Jacques Quisquater, and Pil Joong Lee. Safe-error attack on SPA-FA resistant exponentiations using a HW modular multiplier. In Kil-Hyun Nam and Gwangsoo Rhee, editors, *ICISC*, volume 4817 of *LNCS*, pages 273–281. Springer, 2007. 17

[84] Jongsung Kim, Yuseop Lee, and Sangjin Lee. DES with any reduced masked rounds is not secure against side-channel attacks. *Computers & Mathematics with Applications*, 60(2):347–354, 2010. 26

[85] Ilya Kizhvatov. Side channel analysis of AVR XMEGA crypto engine. In *WESS 2009*. ACM, 2009. 19, 77

[86] Paul Kocher, Joshua Jaffe, Benjamin Jun, and Pankaj Rohatgi. Introduction to differential power analysis. *Journal of Cryptographic Engineering*, 1(1):5–27, 2011. 9, 11

[87] Paul C. Kocher. Timing attacks on implementations of Diffie-Hellman, RSA, DSS, and other systems. In *CRYPTO'96*, volume 1109 of *LNCS*, pages 104–113. Springer, 1996. 5, 9, 14, 22

[88] Paul C. Kocher, Joshua Jaffe, and Benjamin Jun. Differential power analysis. In *CRYPTO'99*, volume 1666 of *LNCS*, pages 338–397. Springer, 1999. 5, 9, 12, 13, 22, 23, 24, 28, 50, 51

[89] V. F. Kolchin, B. A. Sevastyanov, and V. P. Chistyakov. *Random Allocations.* Nauka, Moscow, 1976. In Russian. 71

[90] Markus G. Kuhn. Compromising emanations: Eavesdropping risks of computer displays. Technical Report UCAM-CL-TR-577, University of Cambridge, 2003. Available from `http://www.cl.cam.ac.uk/techreports/UCAM-CL-TR-577.pdf`. 4, 5

[91] Cédric Lauradoux. Collision attacks on processors with cache and countermeasures. In *WEWoRC'05*, volume P-74 of *Lecture Notes in Informatics*, pages 76–85. Gesellschaft für Informatik, Bonn, 2005. 53, 75

[92] Thanh-Ha Le, Jessy Clédière, Christine Servière, and Jean-Louis Lacoume. Efficient solution for misalignment of signal in side channel analysis. In *ICASSP'07*, volume II, pages 257–260. IEEE, 2007. 113

[93] Hervé Ledig, Frédéric Muller, and Frédéric Valette. Enchancing collision attacks. In *CHES'04*, volume 3156 of *LNCS*, pages 176–190. Springer, 2004. 26

[94] Arjen Lenstra, Hendrik Lenstra, Jr., and László Lovász. Factoring polynomials with rational coefficients. *Mathematische Annalen*, 261:513–534, 1982. 94, 96

[95] Zhe Liu, Johann Großschädl, and Ilya Kizhvatov. Efficient and side-channel resistant RSA implementation for 8-bit AVR microcontrollers. Proceedings of the 1st Workshop on the Security of the Internet of Things (SECIOT 2010), 2010. 12

[96] Joe Loughry and David A. Umphress. Information leakage from optical emanations. *ACM Transactions on Information and System Security*, 5(3):262–289, August 2002. Available at `http://applied-math.org/optical_tempest.pdf`. 6

[97] Yingxi Lu, Maire P. O'Neill, and John V. McCanny. FPGA implementation and analysis of random delay insertion countermeasure against DPA. In *International Conference on Field-Programmable Technology – FPT 2008*, pages 201–208, 2008. 114

[98] Qiasi Luo. Enhance multi-bit spectral analysis on hiding in temporal dimension. In *CARDIS'10*, volume 6035 of *LNCS*, pages 13–23. Springer, 2010. 113, 114, 134

[99] Stefan Mangard. A simple power-analysis (SPA) attack on implementations of the AES key expansion. In *ICISC*, volume 2587 of *LNCS*, pages 343–358. Springer, 2002. 13

[100] Stefan Mangard. Hardware countermeasures against DPA – a statistical analysis of their effectiveness. In *CT-RSA'04*, volume 2964 of *LNCS*, pages 222–235. Springer, 2004. 13, 113, 114, 126

[101] Stefan Mangard, Elisabeth Oswald, and Thomas Popp. *Power Analysis Attacks: Revealing the Secrets of Smart Cards.* Springer, 2007. 6, 9, 15, 37, 39, 48, 81, 88

[102] Stefan Mangard, Elisabeth Oswald, and Francois-Xavier Standaert. One for all - all for one: Unifying standard DPA attacks. To appear in IET Information Security. Preprint available at `http://eprint.iacr.org/2009/449`. 13, 23

[103] Robert P. McEvoy, Michael Tunstall, Claire Whelan, Colin C. Murphy, and William P. Marnane. All-or-nothing transforms as a countermeasure to differential side-channel analysis. Cryptology ePrint Archive, Report 2009/185, 2009. `http://eprint.iacr.org/`. 88

[104] Geoffrey McLachlan and David Peel. *Finite Mixture Models.* John Wiley & Sons, 2000. 120

[105] Marcel Medwed, François-Xavier Standaert, Johann Großschädl, and Francesco Regazzoni. Fresh re-keying: Security against side-channel and fault attacks for low-cost devices. In *AFRICACRYPT*, volume 6055 of *LNCS*, pages 279–296. Springer, 2010. 15

[106] Thomas S. Messerges. Using second-order power analysis to attack DPA resistant software. In *CHES*, volume 1965 of *LNCS*, pages 238–251. Springer, 2000. 13

[107] Amir Moradi, Oliver Mischke, and Thomas Eisenbarth. Correlation-enhanced power analysis collision attack. In *CHES'10*, volume 6225 of *LNCS*, pages 125–139. Springer, 2010. 26, 28

[108] Sei Nagashima, Naofumi Homma, Yuichi Imai, Takafumi Aoki, and Akashi Satoh. DPA using phase-based waveform matching against random-delay countermeasure. In *IEEE International Symposium on Circuits and Systems – ISCAS 2007*, pages 1807–1810, May 2007. 113

[109] Michael Neve and Jean-Pierre Seifert. Advances on access-driven cache attacks on AES. In *SAC'06*, volume 4356 of *LNCS*, pages 147–162. Springer, 2007. 14, 51, 53

[110] NSA TEMPEST documents. Online collection, `http://cryptome.org/nsa-tempest.htm`. 5

[111] NXP B.V. LPC2114/2124 single-chip 16/32-bit microcontrollers. `http://www.nxp.com/documents/data_sheet/LPC2114_2124.pdf`, 2007. 65

[112] Olimex. LPC-H2124 header board for LPC2124 ARM7TDMI-S microcontroller. `http://www.olimex.com/dev/lpc-h2124.html`. 65

[113] Siddika Berna Örs, Frank K. Gürkaynak, Elisabeth Oswald, and Bart Preneel. Power-analysis attack on an ASIC AES implementation. In *ITCC*, volume 2, pages 546–552, 2004. 79, 83

[114] Dag Arne Osvik, Adi Shamir, and Eran Tromer. Cache attacks and countermeasures: The case of AES. In *CT-RSA'06*, volume 3860 of *LNCS*, pages 1–20. Springer, 2006. 14, 51, 53

[115] Daniel Page. Theoretical use of cache memory as a cryptanalytic side-channel. Technical report CSTR-02-003, University of Bristol, 2002. 10, 14, 50, 53

[116] Daniel Page. Defending against cache-based side-channel attacks. *Information Security Technical Report*, 8(P1):30–44, 2003. 75

[117] J. Pan, J. I. den Hartog, and Jiqiang Lu. You cannot hide behind the mask: Power analysis on a provably secure S-box implementation. In *Information Security Applications*, volume 5932 of *LNCS*, pages 178–192. Springer, 2009. 26, 28

[118] Eric Peeters, François-Xavier Standaert, and Jean-Jacques Quisquater. Power and electromagnetic analysis: Improved model, consequences and comparisons. *Integration, the VLSI Journal*, 40(1):52 – 60, 2007. 10

[119] B. Poettering. AVRAES: The AES block cipher on AVR controllers, 2006. Available from `http://point-at-infinity.org/avraes`, accessed 6 August 2009. 80, 82

[120] Emmanuel Prouff. DPA attacks and S-boxes. In *FSE*, volume 3557 of *LNCS*, pages 424–441. Springer, 2005. 13

[121] Emmanuel Prouff and Matthieu Rivain. Theoretical and practical aspects of mutual information based side channel analysis. *International Journal of Applied Cryptography*, 2(2):121–138, 2010. 14, 23, 43, 48

[122] Jean-Jacques Quisquater and David Samyde. Electromagnetic analysis (EMA): Measures and counter-measures for smart cards. In *Smart Card Programming and Security, International Conference on Research in Smart Cards — E-smart 2001*, volume 2140 of *LNCS*, pages 200–210. Springer, 2001. 9, 22, 50

[123] Jean-Jacques Quisquater and David Samyde. Eddy current for magnetic analysis with active sensor. In *Proceedings of eSMART*, pages 185–194, 2002. 16

[124] Chester Rebeiro and Debdeep Mukhopadhyay. Cryptanalysis of CLEFIA using differential methods with cache trace patterns. In *CT-RSA'11*, LNCS. Springer, 2011. 53, 65

[125] Mathieu Renauld and François-Xavier Standaert. Algebraic side-channel attacks. In *Inscrypt*, volume 6151 of *LNCS*, pages 393–410. Springer, 2009. 14

[126] Mathieu Renauld, François-Xavier Standaert, and Nicolas Veyrat-Charvillon. Algebraic side-channel attacks on the AES: Why time also matters in DPA. In *CHES'09*, volume 5747 of *LNCS*, pages 97–111, 2009. 14, 24, 48, 51

[127] Riscure. Inspector datasheet: icWaves. `http://www.riscure.com/fileadmin/images/Inspdatasheet/icw_datasheet.pdf`, accessed 2011-03-29. 12, 113

[128] Matthieu Rivain. Differential fault analysis on DES middle rounds. In *CHES*, volume 5747 of *LNCS*, pages 457–469. Springer, 2009. 16

[129] Matthieu Rivain. Securing RSA against fault analysis by double addition chain exponentiation. In *CT-RSA*, volume 5473 of *LNCS*, pages 459–480. Springer, 2009. 17

[130] Matthieu Rivain, Emmanuelle Dottax, and Emmanuel Prouff. Block ciphers implementations provably secure against second order side channel analysis. In *FSE*, volume 5086 of *LNCS*, pages 127–143. Springer, 2008. 15

[131] Matthieu Rivain and Emmanuel Prouff. Provably secure higher-order masking of aes. In *CHES*, volume 6225 of *LNCS*, pages 413–427. Springer, 2010. 15

[132] Ronald Rivest, Adi Shamir, and Leonard Adleman. A method for obtaining digital signatures and public key cryptosystems. *Communications of the ACM*, 21:120–126, 1978. 92

[133] Ronald Rivest et al. The MD6 hash function, 2009. `http://groups.csail.mit.edu/cis/md6/docs/2009-02-21-md6-report.pdf`. 78

[134] Sebastian Rohde. Protocols and light-weight algorithms for wireless authentication through side channels in IEEE 802.11 communication. Master's thesis, Ruhr-Universität Bochum, 2008. 78

[135] Sebastian Rohde, Thomas Eisenbarth, Erik Dahmen, Johannes Buchmann, and Christof Paar. Fast hash-based signatures on constrained devices. In *CARDIS 2008*, volume 5189 of *LNCS*, pages 104–117. Heidelberg, 2008. 78

[136] SAGE, mathematical library. `http://www.sagemath.org`. 103

[137] Werner Schindler, Kerstin Lemke, and Christof Paar. A stochastic model for differential side channel cryptanalysis. In *CHES'05*, LNCS, pages 30–46. Springer, 2005. 14, 23, 24

[138] Jörn-Marc Schmidt and Christoph Herbst. A practical fault attack on square and multiply. In *FDTC 2008*, pages 53–58. IEEE Computer Society, 2008. 16, 104

[139] Jörn-Marc Schmidt and Michael Hutter. Optical and EM fault-attacks on CRT-based RSA: Concrete results. AUSTROCHIP, 2007. 16

[140] Jörn-Marc Schmidt, Michael Hutter, and Thomas Plos. Optical fault attacks on AES: A threat in violet. In *FDTC*, pages 13–22. IEEE Computer Society, 2009. 16

[141] Kai Schramm, Gregor Leander, Patrick Felke, and Christof Paar. A collision-attack on AES: Combining side channel- and differential-attack. In *CHES'04*, volume 3156 of *LNCS*, pages 163–175. Springer, 2004. 24, 25, 26, 28, 51, 53

[142] Kai Schramm, Thomas Wollinger, and Christof Paar. A new class of collision attacks and its application to DES. In *FSE'03*, volume 2887 of *LNCS*, pages 206–222. Springer, 2003. 14, 24, 26

[143] Torsten Schütze. Side-channel analysis (SCA) – a comparative approach on smart cards, embedded systems, and high security solutions. Workshop on Applied Cryptography, Lightweight Cryptography and Side-Channel Analysis, Nanyang Technological University, Singapore, December 3, 2010. 4

[144] Hermann Seuschek. DPA-Analyse von Implementierungen Symmetrischer Kryptographischer Algorithmen. Dimplomaarbeit, TU München, April 2005. In German. Available from `http://www.torsten-schuetze.de/`, accessed 2011-04-01. 12, 45, 46

[145] Adi Shamir and Eran Tromer. Acoustic cryptanalysis. `http://tau.ac.il/~tromer/acoustic/`, 2004. 9

[146] Sergei Skorobogatov. Low temperature data remanence in static RAM. Technical Report UCAM-CL-TR-536, University of Cambridge, Computer Laboratory, June 2002. 8

[147] Sergei Skorobogatov. Optically enhanced position-locked power analysis. In *CHES*, volume 4249 of *LNCS*, pages 61–75. Springer, 2006. 8

[148] Sergei Skorobogatov. Using optical emission analysis for estimating contribution to power analysis. In *FDTC*, pages 111–119. IEEE Computer Society, 2009. 9

[149] Sergei Skorobogatov. Synchronization method for SCA and fault attacks. *Journal of Cryptographic Engineering*, 1(1):71–77, 2011. 12

[150] Sergei Skorobogatov and Ross Anderson. Optical fault induction attack. In *CHES*, volume 2523 of *LNCS*, pages 2–12. Springer, 2002. 16

[151] François-Xavier Standaert, François Koeune, and Werner Schindler. How to compare profiled side-channel attacks? In *ACNS*, volume 5536 of *LNCS*, pages 485–498, 2009. 14

[152] François-Xavier Standaert, Tal Malkin, and Moti Yung. A unified framework for the analysis of side-channel key recovery attacks. In *EUROCRYPT'09*, volume 5479 of *LNCS*, pages 443–461. Springer, 2009. 14, 35, 42, 48

[153] François-Xavier Standaert, Olivier Pereira, Yu Yu, Jean-Jacques Quisquater, Moti Yung, and Elisabeth Oswald. Leakage resilient cryptography in practice. Cryptology ePrint Archive, Report 2009/341, 2009. `http://eprint.iacr.org/`. 15

[154] TEMPEST: A signal problem. *Cryptologic Spectrum*, 2(3), 1972. Declassified in 2007, available at `http://www.nsa.gov/public_info/_files/cryptologic_spectrum/tempest.pdf`. 4

[155] Kris Tiri, David Hwang, Alireza Hodjat, Bo-Cheng Lai, Shenglin Yang, Patrick Schaumont, and Ingrid Verbauwhede. Prototype IC with WDDL and differential routing - DPA resistance assessment. In *CHES 2005*, volume 3659 of *LNCS*, pages 354–365. Springer, 2005. 79

[156] Michael Tunstall and Olivier Benoit. Efficient use of random delays in embedded software. In *WISTP 2007*, volume 4462 of *LNCS*, pages 27–38. Springer, 2007. 111, 114, 115, 117, 118, 124, 125, 131, 133

[157] Wim van Eck. Electromagnetic radiation from video display units: An eavesdropping risk? *Computers & Security*, 4:269–286, 1985. 5

[158] Jasper G. J. van Woudenberg, Marc F. Witteman, and Bram Bakker. Improving Differential Power Analysis by elastic alignment. `http://www.riscure.com/fileadmin/images/Docs/elastic_paper.pdf`, 2009. 113

[159] Ingrid M. R. Verbauwhede, editor. *Secure Integrated Circuits and Systems.* Springer, 2010. 6, 9

[160] Andreas Wiemers. Collision Attacks for Comp128 on Smartcards. ECC-Brainpool Workshop on Side-Channel Attacks on Cryptographic Algorithms, Bonn, Germany, December 2001. 26

[161] Charles V. Wright, Lucas Ballard, Fabian Monrose, and Gerald M. Masson. Language identification of encrypted VoIP traffic: Alejandra y Roberto or Alice and Bob? In *16th USENIX Security Symposiumm*, pages 4:1–4:12. USENIX Association, 2007. 6

[162] Peter Wright. *Spycatcher: The Candid Autobiography of a Senior Intelligence Officer.* Heinemann Publishers Australia, 1987. 5

[163] Sung-Min Yen and Marc Joye. Checking before output may not be enough against fault-based cryptanalysis. *IEEE Transactions on Computers*, 49(9):967–970, 2000. 17

[164] Sung-Ming Yen, Seungjoo Kim, Seongan Lim, and Sang-Jae Moon. A countermeasure against one physical cryptanalysis may benefit another attack. In *ICISC*, volume 2288 of *LNCS*, pages 414–427. Springer, 2001. 17

[165] Sung-Ming Yen, Seungjoo Kim, Seongan Lim, and Sang-Jae Moon. RSA speedup with Chinese remainder theorem immune against hardware fault cryptanalysis. *IEEE Transactions on Computers*, 52:461–472, 2003. 113

[166] John Young. How old is TEMPEST? Online response collection, `http://cryptome.org/tempest-old.htm`, 2000. 4

[167] Xin-Jie Zhao and Tao Wang. Improved cache trace attack on AES and CLEFIA by considering cache miss and S-box misalignment. Cryptology ePrint Archive, Report 2010/056, 2010. `http://eprint.iacr.org/2010/056`. 53, 55

[168] XMEGA A manual preliminary, revision G, April 2009. `http://www.atmel.com/dyn/resources/prod_documents/doc8077.pdf`. 78, 79, 81, 82

[169] Atmel secure products, revision D, October 2008. `http://www.atmel.com/dyn/resources/prod_documents/doc6523.pdf`. 81

[170] 8-bit AVR instruction set, revision H, July 2009. `http://www.atmel.com/dyn/resources/prod_documents/doc0856.pdf`. 79

[171] FIPS PUB 197: Specification for the Advanced Encryption Standard, 2001. `http://www.csrc.nist.gov/publications/fips/fips197/fips-197.pdf`. 26, 52, 78, 80

[172] FIPS PUB 46-3: Data Encryption Standard, 1999. `http://csrc.nist.gov/publications/fips/fips46-3/fips46-3.pdf`. 78, 79

List of publications

Andrey Bogdanov and Ilya Kizhvatov. Beyond the limits of DPA: Combined side-channel collision attacks. To appear in *IEEE Transactions on Computers*, 2011. Preprint available at `http://eprint.iacr.org/2010/590`.

Alex Biryukov, Ilya Kizhvatov, and Bin Zhang. Cryptanalysis of the Atmel cipher in SecureMemory, CryptoMemory and CryptoRF. In *ACNS 2011*, volume 6715 of *LNCS*, pages 91–109. Springer, 2011.

Jean-François Gallais and Ilya Kizhvatov. Error-tolerance in trace-driven cache collision attacks. Proceedings of COSADE 2011.

Zhe Liu, Johann Großschädl, and Ilya Kizhvatov. Efficient and side-channel resistant RSA implementation for 8-bit AVR microcontrollers. Proceedings of the 1st Workshop on the Security of the Internet of Things (SECIOT 2010), 2010.

Johann Großschädl and Ilya Kizhvatov. Performance and security aspects of client-side SSL/TLS processing on mobile devices. In *CANS 2010*, volume 6467 of *LNCS*, pages 44–61. Springer, 2010.

Jörn-Marc Schmidt, Michael Tunstall, Roberto Avanzi, Ilya Kizhvatov, Timo Kasper, and David Oswald. Combined implementation attack resistant exponentiation. In *LATINCRYPT 2010*, volume 6212 of *LNCS*, pages 305–322. Springer, 2010.

Jean-Sébastien Coron and Ilya Kizhvatov. Analysis and improvement of the random delay countermeasure of CHES 2009. In *CHES 2010*, volume 6225 of *LNCS*, pages 95–109. Springer, 2010.

Jean-François Gallais, Ilya Kizhvatov, and Michael Tunstall. Improved trace-driven cache-collision attacks against embedded AES implementations. In *WISA 2010*, volume 6513 of *LNCS*, pages 243–257. Springer, 2011. Extended version available at `http://eprint.iacr.org/2010/408`.

Ilya Kizhvatov. Side channel analysis of AVR XMEGA crypto engine. In *WESS 2009*. ACM, 2009.

Jean-Sébastien Coron and Ilya Kizhvatov. Analysis of the split mask countermeasure for embedded systems. In *WESS 2009*. ACM, 2009.

Jean-Sébastien Coron, Antoine Joux, Ilya Kizhvatov, David Naccache, and Pascal Paillier. Fault attacks on RSA signatures with partially unknown messages. In *CHES 2009*, volume 5747 of *LNCS*, pages 444–456. Springer, 2009. Full version available at `http://eprint.iacr.org/2009/309`.

Jean-Sébastien Coron and Ilya Kizhvatov. An efficient method for random delay generation in embedded software. In *CHES 2009*, volume 5747 of *LNCS*, pages 156–170. Springer, 2009. Extended version available at `http://eprint.iacr.org/2009/419`.

Andrey Bogdanov, Ilya Kizhvatov, and Andrey Pyshkin. Algebraic methods in side-channel collision attacks and practical collision detection. In *INDOCRYPT 2008*, volume 5365 of *LNCS*, pages 251–265, 2008.

Andrey Bogdanov and Ilya Kizhvatov. Cryptanalysis of NiVa stream encryption algorithm. *Journal of Information Technology Security*, 3:9–13, MEPhI, 2007. In Russian.

Vladimir Anashin, Andrey Bogdanov, and Ilya Kizhvatov. Security and implementation properties of ABC v.2. SASC 2006—Stream Ciphers Revisited, Leuven, Belgium, February 2-3, 2006.

Vladimir Anashin, Andrey Bogdanov, and Ilya Kizhvatov. ABC: A new fast flexible stream cipher. SKEW—Symmetric Key Encryption Workshop, Aarhus, Denmark, May 26-27, 2005.