Symmetric cryptography applied in different contexts: physical attacks and ransomware.

Hélène Le Bouder

2023

Hélène Le Bouder

Symmetric cryptography applied in different contexts: physical attacks and ransomware.

HDR manuscript

Université de Rennes

IMT Atlantique
Bretagne-Pays de la Loire
École Mines-Télécom

Jury members:

- Rapporteur: Fournier Jacques director of research at STMicroelectronics.
- Rapporteur: Marine Minier professor at Loria.
- Rapporteur: Peter Ryan professor at Luxembourg University.
- Jury President: Sylvain Duquesne professor at Rennes University.
- Examiner: Aurélien Francillon professor at Eurecom.
- Examiner: Arnaud Tisserand director of research at CNRS.

Defence: 9th October 2023.

to Jean-Louis

"Je sers la science et c'est ma joie."

Disciple Basile
Léonard De Groot et Turk.

Acknowledgements

Firstly, I wish to express my deepest gratitude to my unwavering scientific and emotional support over the years: my sweet Ronan Lashermes.

I then wish to extend my thanks to my reviewers: Marine Minier, Jacques Fournier and Peter Ryan and the other jury members: Sylvain Duquesne, Aurélien Francillon and Arnaud Tisseran for evaluating my work.

I am incredibly grateful to Jean-Louis Lanet, who was first an excellent professor, then a perfect mentor, and a wonderful colleague.

I'd like to extend my thanks to my IRISA mentor, Sophie Pinchinat, for all her relevant advice and encouragement.

I would like to express a heartfelt thank you to my dear colleague and friend Gaël Thomas, for all the projects we have undertaken together.

I am grateful to Nicolas Montavont, the head of the "Objets Communicants pour l'Internet du Futur" (OCIF) team, for his unwavering support and impeccable ethics.

I would like to express my gratitude to my colleagues at IMT-Atlantique campus Rennes, especially Baptiste, Fabien and Renzo, and my favourite support functions colleagues: Nardjes, DelphineS, and Alexis. Their presence has made my work more enjoyable.

I would like to thank Ronan Lashermes, Loutfi Nuaymi, Patrick Maille and Gaël Thomas for their valuable comments to the development of this manuscript.

I would like to recognize the work of all my students and PhD students and remind that students are the future of science.

I'd also like to extend my thanks to women who inspire me, in work: Valérie Viet Triem Tong, Géraldine Texier, Caroline Fontaine and in my personal life: Amanda, Pat, Isabelle, Julie, and Irina. They are strong women, and the world needs more women like them.

I want to extend specific acknowledgments to my fun support network, the climbing team at INRIA, my friends from ice skating, artistic rollers, painting, and geek parties (special dedication to: Artefaritaj, Theudeux, Azura, Bichette, and Titoon).

To conclude, I want to express my gratitude to my entire family, particularly my son Gaston; without forgetting my cute pets: Dune, Pixel, Mariette, and Soupir, who always provide comforting hugs.

Contents

Chapter 1

Introduction and context

"*Victoriae mundis et mundis lacrima.*
Bon, ça ne veut absolument rien dire, mais je trouve que c'est assez dans le ton."
Le roi Loth dans Kaamelott
d'Alexandre Astier.

Contents

1.1 Introduction

1.1.1 Professional context

Tracing back to my scientific origins, I hold a master's degree in cryptology from the University of Limoges.

I defended my PhD thesis in microelectronics entitled *A Formalism for Physical Attacks on Cryptographic Device and Its Exploitation to Compare and Research New Attacks*, at École Nationale Supérieure des Mines de Saint-Étienne (ENSMSE) in Gardanne on October 24, 2014.

Then I pursued with a postdoc during approximately two years (2015-2017) at Institut national de recherche en sciences et technologies du numérique (INRIA) Rennes in the threat analysis and mitigation for information security (TAMIS) team. I was funded by the Agence Nationale de la Recherche (ANR) `Cogito` project [17]. The laboratory high security (LHS) of INRIA Rennes has been created at this time. I worked on the development of different experimental benches in the LHS. In 2015, the pandemic of ransomware just began, and I took the opportunity to work on this subject.

My first PhD student, Aurélien Palisse defended in 2019 [18], and two others, Routa Moussalieb [18] and Leopold Ouairy [19] in 2020. Aurélien Palisse and Routa Moussalieb worked on ransomware [1–3, 6, 7]. The PhD of Léopold Ouairy was about fuzzing on smart cards [20–23].

Since 2017, I have been an associate professor at IMT-Atlantique in OCIF team. This Institut de Recherche en Informatique et Systèmes Aléatoires (IRISA) team is focusing on the internet of things (IoT) in all its aspects, including security.

I recruited Slaheddine Zerkan for a postdoc on a project of security of wireless communications for battery management systems in cars. This project was in the OCIF team context with Nicolas Montavont, Georgios Papadopulos and the PhD student Guillaume Le Gall. Unfortunately, Covid and confinement aborted the project.

Today, I am leading a project funded by the Brittany region: "Attaques Physiques sur les Chiffrements Intègres Légers (APCIL)" on the vulnerabilities due to physical attacks on the new lightweight cryptographic algorithms proposed in the national institute of standards and technology (NIST) competition. Additionally, I am involved in other research initiatives, such as the Beyond 5G project funded by France Relance. I am also a permanent member of the Cyber CNI Chair at IMT-Atlantique.

In this context I am advising three theses. Modou Sarry works on the APCIL project. Awaleh Houssein works in the Cyber CNI Chair, on using physical information leakage such as sound to detect anomalies in programmable logic controllers. The last PhD student Pierre-Marie Lechevalier is focused on guaranteeing trust and security in the composition of distributed network services and in the Beyond 5G project.

My publications and results span various domains of security in defence and attack. I have a strong background in the field of symmetric cryptography. But, symmetric cryptography is not the only security tool I am working with. `PIN` code is another important as shown in [9, 24]. In this manuscript, I choose to focus on ransomware and physical attacks with a symmetric cryptographic view, even if they are not my only work topics.

1.1.2 Personal motivation to the HDR

Since my nomination as an associate professor at IMT-Atlantique, teaching has been a very time-consuming. Indeed, many of the lessons I teach did not exist at the school before. On the other hand, during three years, I was the co-head pedagogical, along with Romaric Ludinard, of the Mastère Spécialisé en Cyber-Sécurité (MSCS). This responsibility could be a full-time job.

Obtaining an HDR is a good way to refocus on my research career. My research team, OCIF, is restructuring, and it is an opportunity for me to highlight my research topics. The HDR gives me the possibility to gain true research independence. I have worked on different subjects and with different people in computer security, including: physical attacks, security of embedded devices, vehicle security, malware, trust, network security, and cryptography. I have learned a lot and have a broad vision of cybersecurity. In the future, I wish to primarily refocus on my preferred research topic: the physical attacks.

1.1.3 Scientific context

1.1.3.1 Computer security and cryptography

Computer security, also called cybersecurity is a critical aspect of modern computing that focuses on protecting computer systems and networks, from malicious activities. As technology continues to advance and becomes more integrated into our daily lives, the need for robust computer security measures becomes increasingly important. Without proper security measures in place, individuals, organizations, and even governments are vulnerable to cyber attacks, threats that can have serious consequences. Ransomware is an example of the most virulent attacks of our time.

Cryptography, etymologically the science of secrecy, is a subdomain of computer security. Modern cryptography focuses on four properties:

1. **confidentiality**: to limit data access to authorized people;
2. **integrity**: to ensure that the information cannot be altered;
3. **authentication**: to validate the origin of data;
4. **non-repudiation**: to allow a person to take part in a contract without the possibility to denounce it later.

Modern cryptography can be split in two subdomains: symmetric and asymmetric cryptography. This document focuses only on symmetric cryptography.

Although the principles of cryptography are as old as the world, it never ceases to evolve to meet security needs. It must adapt to new needs and new devices, while being robust to attacks.

1.1.3.2 IoT and embedded devices

The Internet is a fantastic concept where all systems can interconnect, using a standard and recognized approach with common protocols. Internet of things (IoT) is a good illustration of this global interoperability, where embedded systems, serving as sensors or actuators, are connected to the Internet. IoT connects everyone and everything. It results that such devices can operate as servers and can make resources available to various clients, including our traditional web browser on our computers. The IoT avoids deploying systems in silos, by reusing existing principles and protocols to implement a specific application, which solves the problems of modularity, extensibility and integration. This is the main reason for the rapid growth of the IoT in many different domains, ranging from e-agriculture to smart cities, including Industry 4.0. However, as these interconnected networks became more popular among the general public, they also became susceptible to attacks. A simplified vision of the IoT could be to say that now that all systems are interconnected, any node or communication between nodes could be compromised and attackers could benefit from many different entry points and may penetrate deeply in a given system. So cybersecurity is the main feature for the sustainability of these systems.

To meet the security requirements, robust and practical solutions are expected, especially when considering embedded devices that usually have low capacity in terms of energy and computation.

The functional correctness of IoT is an absolute prerequisite. In summary, IoT needs security as cryptography [25] and `PIN` code.

Today, an increasing number of embedded systems have multifunctional capabilities, similar to our mobile phones. However, the traditional definition is as follows. An **embedded system** is defined as an autonomous digital system, with three particular characteristics.

1. It is often specialized in a specific task, for example a smart card can be dedicated to payment.
2. An embedded system has space constraints. Some devices are getting smaller and smaller.
3. Finally, it has energy constraints, so restricted consumption.

Due to their spatial and energy constraints, the resources of an embedded system are generally limited. In this context, specific cryptographic algorithms called "**lightweight**" are developed.

These autonomous devices are more and more widespread, especially in IoT. These devices are prime targets for physical attacks.

1.1.3.3 Physical attacks

Between the mathematical algorithm and its practical implementation in a circuit as software or hardware implementation, there are various levels of abstraction. At each level of abstraction, new vulnerabilities are possible, which is why theoretical algorithms cannot anticipate all attacks (present and future) at once. A comprehensive list of the different abstraction levels is the following.

1. **Algorithm**: the mathematical theory.
2. **Software**: the different programming language such as: C, java, python, *etc.* Once designed, the algorithm is described by a computer scientist in a programming language. The program is then compiled to be translated into machine language into a set of instructions.
3. **Architecture and microarchitecture**. The processor or central processing unit (CPU) (Central Processing Unit) is the component of the computer that executes machine instructions. A processor built into a single integrated circuit is called a microprocessor. A programmer may choose to operate directly at this level and code directly in assembly code (machine language).
4. **Register transfer logic (RTL)**. An arithmetic logic unit (ALU) instruction can be split down into Boolean functions. They are implemented in electronics in the form of logic gates connected by wires.
5. **Transistors**: semiconductor devices that control a current or voltage. Logic gates are built from several transistors connected appropriately. A logical 0 corresponds to a voltage below a certain value, otherwise it corresponds to a logical 1.

Even if an algorithm is proved secure mathematically, cryptanalysis has another dimension: **physical attacks**. These attacks rely on the interaction of the computing unit with the physical environment. Thus, these attacks occur at the levels RTL and transistors.
Physical attacks can be split in two parts: fault injection attack (FIA) and side channel analysis (SCA).

The **physical cryptanalysis** is the research field, in which an attacker uses physical parameters to attack cryptographic primitives. In this domain the impact of the attack is at the level 1 or 2. According to their goal, physical symmetric cryptanalysis can be classified into two categories. The attacks are those the two following categories.

➤ **Confidentiality**

- Sometimes, the attacker attempts to obtain the secret key.
- In the case of reverse engineering, an attacker aims to extract a portion or all of an algorithm function.

➤ **Integrity**

- The attacker aims to alter the data, for instance, by substituting a part of a message with a selection of their own. This constitutes an attack on data integrity at the algorithmic level 1.
- The attacker wants to change the program execution. They attempt to skip instructions or replace them with others, for instance, to avoid using a particular secret and substitute it with another.

In the literature, some frameworks have been published to evaluate and classify physical attacks [26–29]. During my PhD, I worked to find a framework to describe physical attacks [30–32].

1.1.4 Scientific challenges with a cryptographer view

In this manuscript, I have chosen to present results and future work in two contexts: ransomware and physical attacks, with symmetric cryptography as a common point of view.
A primary motivation is to study new algorithms of symmetric cryptography in the field of physical attacks. Even if I am not a pure cryptographer, I enjoy studying cryptography from different perspectives and applications. Also, in this manuscript, cryptographic primitives are sometimes a threat as in chapter 2, a target as in chapter 3 or a defensive tool as in chapter 4.

1.1.4.1 Ransomware

When discussing ransomware, it can be compared to a global pandemic, as attacks have become a significant threat across various sectors. The ethical motivation to find solutions to ransomware is clear.

Ransomware has turned encryption into a weapon, forcing us to fight back against our own robust tool. From a cryptographer point of view, it is stimulating to see encryption properties from an opposite perspective. In the ransomware context, encryption is a threat.

1.1.4.2 Physical attacks

There are more and more small circuits such as smart cards and connected devices in our daily lives. These circuits can contain critical or private information and can play a decisive role for their owner. However, if these circuits fall into the wrong hands, they can become targets for physical attacks. Also, studying physical attacks to better anticipate them improving the protection is a primary motivation.

1.1.5 Contributions

1.1.5.1 Ransomware

I worked for six years on the detection and remediation of ransomware. I was the advisor of two PhD students on this topic: Aurélien Palisse [18] and Routa Moussalieb [33]. A platform, malware-o-matic (MoM), for testing ransomware has been developed by these PhD students in the LHS. Different contributions about the detection and mitigation of ransomware behaviour were published [2–7] in various conferences. A survey was published in [1].

1.1.5.2 Physical attacks

Since my PhD defence, I have been collaborating to the LHS platforms development. Different attacks [8–11] in this domain have been developed with different target cryptographic primitives and `PIN` code implementations. Other publications about countermeasures have been published too [15,16]. Currently, I advise the PhD student Modou Sarry on side channel analysis (SCA) on authenticated encryption with associated data (AEAD). This manuscript presents published contributions, work in progress, and future projects.

1.1.6 Organization

The common link between the different chapters of this manuscript is symmetric cryptography. Indeed symmetric cryptography algorithms are studied as a threat in the chapter 2, a target in the chapter 3 and a solution in the chapter 4. In other words :

- chapter 2 explains how crypto-ransomware use symmetric encryption and solutions to stop them;
- chapter 3 presents SCA attacks against classic symmetric cryptography algorithms and new generations of lightweight symmetric cryptography algorithms: AEAD;
- chapter 4 develops the idea to use symmetric cryptography to protect processor instructions.

Finally, the conclusion is drawn in chapter 5.

These three chapters have a logical time sequence. Indeed chapter 2 presents results during these last years; the chapter 3 is a big part of my current work; the chapter 4 is past and future work.

In the continuation of this chapter 1, the context of symmetric cryptography is presented in section 1.2. More specifically, the algorithms studied in the report are described in section 1.2.3. To conclude this chapter, the section 1.3 is dedicated to teaching.

1.2 Symmetric cryptography prerequisites

This section 1.2 presents the classical symmetric cryptography 1.2.1 and its evolution in 1.2.2. Then in the algorithms used in the different chapters are described as prerequisites in 1.2.3.

1.2.1 Classical and older symmetric cryptography

Classical and older symmetric cryptography regroups the two following tool families.

1. Symmetric encryption is used for confidentiality.
   An **encryption** algorithm transforms a message M called plaintext, with a key K, into data called ciphertext C, unintelligible for anyone who does not have the key to decrypt. The **decryption** algorithm takes as inputs the ciphertext C and key K and returns the plaintext M. In a symmetric encryption algorithm, the same key is used to encrypt and decrypt. This key K must remain secret.
2. **Message authentication code (MAC)** and hash-based message authentication code (HMAC) are used for integrity.
   A **MAC** takes in input M a text and a secret key K and returns a tag T of integrity. Only the owners of the key can verify the integrity of the message.

These tools: symmetric encryption, MAC and HMAC are built from symmetric primitive functions.

- In a **stream cipher**, the ciphertext is the result of an xor between plaintext and a vector of the same size. This vector is a random secret and disposable key or a pseudo-random sequence computed with a secret key and an initial seed.
- A **block cipher** is a deterministic primitive function that operates on fixed-length inputs, called blocks, and a secret key K. Therefore, the message needs to be divided into blocks and each block is then ciphered individually.
- A **hash function** noted $\mathcal{H}$ maps a text of arbitrary size to data of fixed size. A cryptographic hash function resists collision attacks and pre-image attacks.
- The mode of operation, called **mode**, describes how to repeatedly apply a block cipher or sometimes a hash function.

So a symmetric encryption can be a stream cipher or block cipher with a mode. A MAC is built with by a block cipher and a mode and a HMAC with a hash function and a mode.

Symmetric cryptography security applies the following Shannon properties [34].

- The **diffusion** property applied to a symmetric primitive function is the idea that changing one bit in the input, should independently change each output bit with probability 0.5.
- The **confusion** property applied to a symmetric primitive function is an ambition to make the link between the input and the output as complex as possible.

1.2.2 New generation of symmetric cryptography algorithms

In numerous use cases, users require both confidentiality and integrity. Therefore, with older symmetric cryptography, they had to utilize a specific mode to guarantee integrity in symmetric encryption, or they had to combine MACs and encryption.
On the other hand, there are more and more small devices with low computing power, that need cryptography. So lightweight algorithms are required.
In this context, the NIST started the competition [35] for lightweight cryptography candidates for AEAD.

An **authenticated encryption with associated data (AEAD)** illustrated in Fig 1.1, should ensures confidentiality and integrity. In the specification of an AEAD the primitive and the mode are fixed. The user does not have choice. An AEAD takes as input different parameters: a plaintext denoted by M, data associated to the plaintext denoted by A, a secret key K, and an initialization vector N also called a nonce. The nonce is

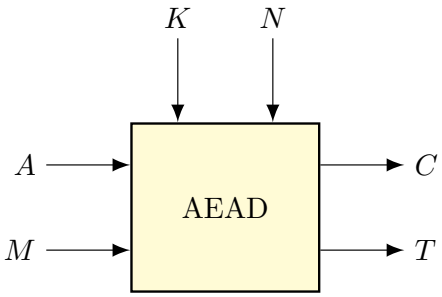


Figure 1.1: Scheme of AEAD .

public but must be different for each new plaintext. The algorithm ensure confidentiality of the plaintext and integrity of both the plaintext and the associated data. It returns a ciphertext C and a tag T.

NIST received 57 submissions to be considered for standardization. In March 2021, only 10 finalists were selected: ASCON [36], Elephant [37], GIFT-COFB [38], Grain128-AEAD [39], ISAP [40], Photon-Beetle [41], Romulus [42], SPARKLE [43], TinyJambu [44] and Xoodyak [45]. Finally, in February 2023, NIST decided to standardize ASCON [36].

I started to study these algorithms in 2020, so I did not know which algorithm would become the winner.

1.2.3 Algorithms studied in the report

There are numerous symmetric algorithms, but in the following, only algorithms studied in the report are described as prerequisites.

1.2.3.1 Block ciphers

1.2.3.1.1 Advanced encryption standard (AES)

The advanced encryption standard (AES) is a standard established by the NIST [46] for symmetric key cryptography. It is a block cipher.

The encryption first consists of mapping the plaintext M of 128 bits into a two-dimensional array of $4\times4 = 16$ bytes called the state. Rows and columns are respectively indexed i and j.

Rounds description After a preliminary xor (denoted $\oplus$) between the input and the key K_0, the advanced encryption standard (AES) executes 10 times a round function that operates on the state. The operations used during these rounds are:

- **SubBytes (SB)** composed of non-linear transformations: 16 S-boxes, working independently on individual bytes of the state;
- **ShiftRow (SR)** a byte-shifting operation in each row of the state;
- **MixColumns (MC)** a linear matrix multiplication on $\mathrm{GF}(2^8)$, working on each column of the state;
- **AddRoundKey** a xor between the state and the round key K_r, $r \in [\![0, 10]\!]$.

The algorithm is summarized in Figure 1.2.

The derived key K denotes the master key. The size of the master key is 128 bits. K_r is the round key used in round r, K_r is represented by a two-dimensional array of $4\cdot4$ bytes, like the state. $K_r^{i,j}$ is the round-key byte at row i and column j. The round key K_{r+1} depends on the round key K_r with $K_0 = K$. More precisely, the round keys are computed with a KeyExpansion function described by the system of equations (1.1), where SB is the S-Box function and Rcon is a constant matrix of size $4 \cdot 10$.

$$\begin{cases} K_{r+1}^{i,0} = K_r^{l,0} \oplus \mathrm{Rcon}(i,r) \oplus SB\left(K_r^{i+1 \bmod 4,3}\right) & \forall i \in [\![0,3]\!] \\ K_{r+1}^{i,j} = K_r^{i,j} \oplus K_{r+1}^{i,j-1} & \forall i \in [\![0,3]\!] \text{ and } j \in [\![1,3]\!] \end{cases} \tag{1.1}$$

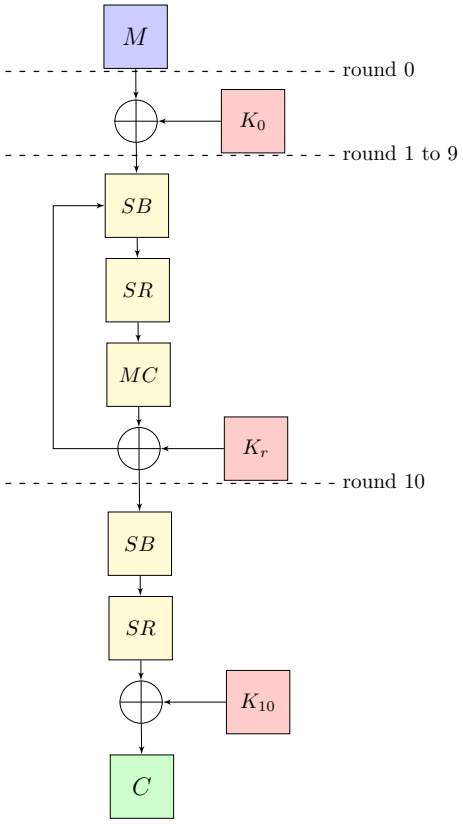


Figure 1.2: scheme of block cipher AES

1.2.3.1.2 TWINE TWINE is a 64-bit block cipher presented in [47] aiming at a reduced circuit area while avoiding non-software friendly operation (such as bit permutations) to maintain decent performances when implemented in software. It is a generalized Feistel network (GFN) with 16 blocks β, 4 bits each, that can accept either 80-bit or 128-bit keys. The permutation layer that rearranges the blocks at the end of each round was chosen to maximize diffusion between blocks, according to a result of [48]. Compared to the cyclic shift, this permutation requires twice less rounds for an input difference to influence all the blocks. One round of TWINE is depicted on FIGURE 1.3. The whole encryption process consists in 36 rounds of this scheme for both key lengths. The Feistel function is used 8 times per round and is consecutively made of:

- a 4-bit $\oplus$ with a subkey block,
- a single 4×4 S-box.

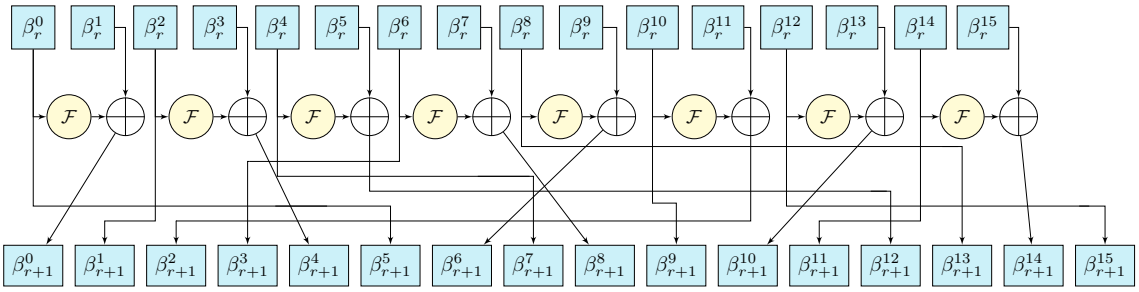


Figure 1.3: scheme of TWINE round

1.2.3.2 Pelican

Pelican [49] is an iterative MAC function based on AES block cipher.

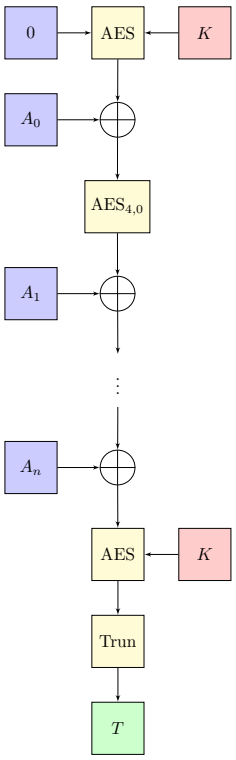


Figure 1.4: Block scheme of Pelican, where $AES_{4,0}$ denotes a 4-rounds Rinjdael with round key set to 0.

1.2.3.3 Encryption modes

1.2.3.3.1 Electronic code book (ECB) The electronic code book (ECB) mode encryption is a simple mode. The data are divided into blocks, and each block is encrypted separately (1.2).

$$C_i = En(M_i, K). \tag{1.2}$$

It is illustrated in Figure 1.5. This mode is vulnerable to replay attacks in among others, but it still used.

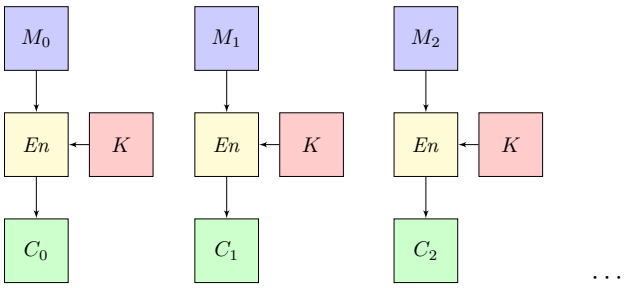


Figure 1.5: Scheme of ECB mode encryption

1.2.3.3.2 Cipher block chaining (CBC) In cipher block chaining (CBC) mode, each block of plaintext M is xored with the previous ciphertext block before being encrypted. An initialization vector IV is xored with the first block M_0. The relation is defined in (1.3).

$$\begin{cases} C_0 & = En(M_0 \oplus IV, K) \\ C_{i+1} & = En(M_{i+1} \oplus C_i, K) \end{cases} \tag{1.3}$$

It is illustrated in Figure 1.6.

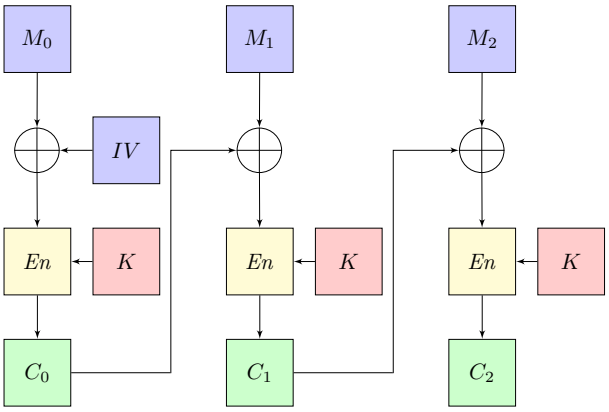


Figure 1.6: Scheme of CBC mode encryption

1.2.3.3.3 Counter (CTR) The counter (CTR) mode, described in Figure 1.7 operates like a stream cipher. The block cipher takes a nonce (the counter) as input. Subsequently, the plaintext block is xored with the output of the block cipher. The advantage of counter (CTR) mode over CBC mode is that blocks can be decrypted in parallel.

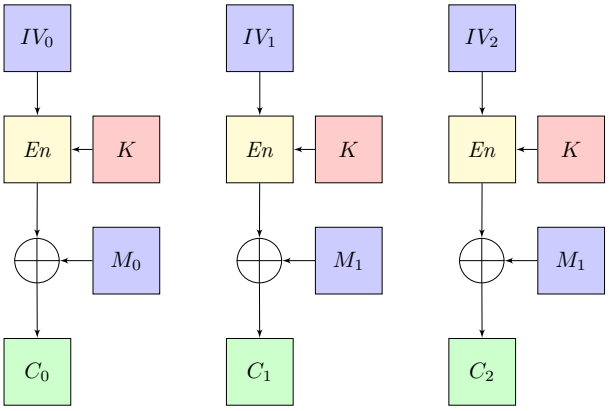


Figure 1.7: Scheme of CTR

1.2.3.3.4 Galois counter mode (GCM)

The Galois counter mode (GCM) [50] ensures confidentiality and integrity.

Encryption operates like a stream cipher like CTR mode, with the block cipher primitive used to generate random data.

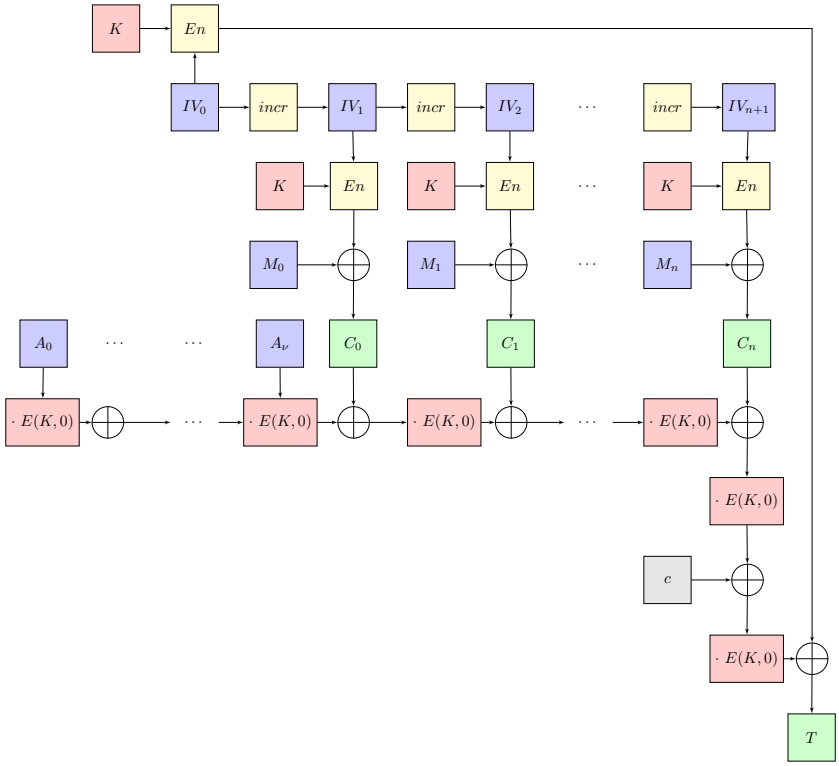


Figure 1.8: Scheme of Galois counter mode (GCM)

1.2.3.4 AEADs

1.2.3.4.1 Elephant

Elephant [37, 51] is an AEAD finalist to the NIST lightweight cryptography competition. It is an Encrypt − then − MAC construction that combines CTR mode [52] encryption with a variant of the protected counter sum [53]. It uses a cryptographic permutation masked with LFSRs in an Even − Mansour − like − fashion [54] in place of a block cipher.

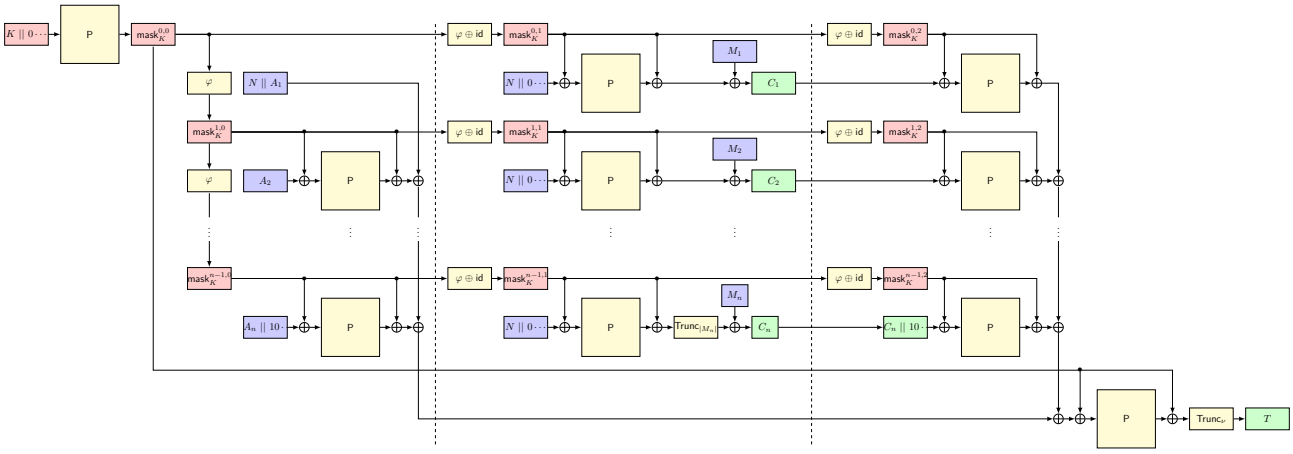


Figure 1.9: Elephant plaintext encryption (left), ciphertext authentication (middle), and associated data authentication (end).

Let P be an n-bit cryptographic permutation, and φ an n-bit LFSR.
Let the function $\mathsf{mask} : \{0,1\}^{128} \times \mathbb{N} \times \{0,1,2\} \to \{0,1\}^n$ be defined as follows:

$$\mathsf{mask}_K^{i,j} = \mathsf{mask}(K,i,j) = (\varphi \oplus \mathsf{id})^j \circ \varphi^i \circ \mathsf{P}\left(K||0^{n-128}\right), \tag{1.4}$$

where id is the identity function. Let Split be the function that splits an input into n-bit blocks, where the last block is zero-padded. Let Trunc_ν be the function that returns ν left-most bits of an input.

Encryption En under Elephant takes as input a 128-bit key K, a 96-bit nonce N, associated data A, and a plaintext M. It outputs a ciphertext C as large as M, and a ν-bit tag T. The description is depicted on Figure 1.9.

Decryption gets as input a 128-bit key K, a 96-bit nonce N, associated data A, a ciphertext C, and ν-bit tag T. It outputs a plaintext M as large as C if the tag T is correct, or the symbol $\perp$ otherwise. The description easily follows from that of encryption.

Elephant comes in three flavours which differ on the n-bit cryptographic permutation P and the LFSR φ used, as well as the tag size ν.
Dumbo uses the 160-bit permutation Spongent [55], the LFSR φ_{Dumbo} given by equation (1.5) and illustrated on Figure 1.10, and has tag size $\nu = 64$ bits.

$$\varphi_{\mathsf{Dumbo}} : (B_0, \cdots, B_{19}) \mapsto (B_1, \cdots, B_{19}, (B_0 \lll 3) \oplus (B_3 \ll 7) \oplus (B_{13} \gg 7)) \tag{1.5}$$

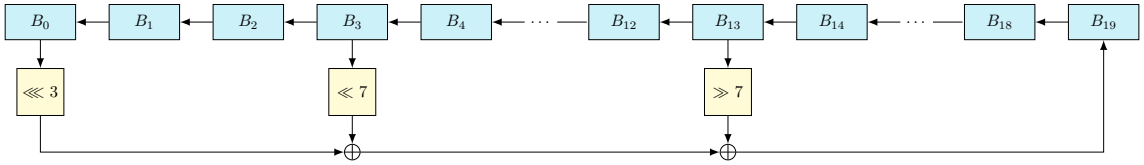


Figure 1.10: 160-bit LFSR φ_{Dumbo}.

Jumbo uses the 176-bit permutation Spongent [55], the LFSR φ_{Jumbo} given by equation (1.6) and illustrated on Figure 1.11, and has tag size $\nu = 64$ bits.

$$\varphi_{\mathsf{Jumbo}} : (B_0, \cdots, B_{21}) \mapsto (B_1, \cdots, B_{21}, B_0 \lll 1 \oplus B_3 \ll 7 \oplus B_{19} \gg 7) \tag{1.6}$$

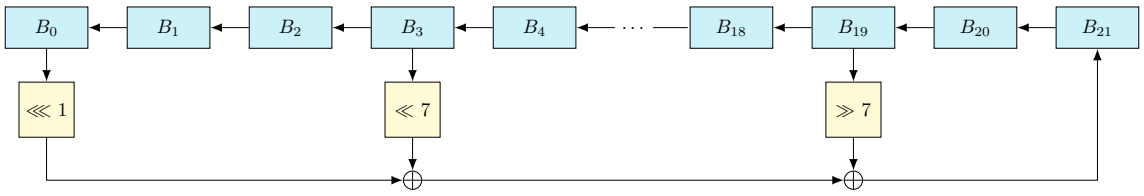


Figure 1.11: 176-bit LFSR φ_{Jumbo}.

$$\varphi_{\mathsf{Delirium}} : (B_0, \cdots, B_{24}) \mapsto (B_1, \cdots, B_{24}, B_0 \lll 1 \oplus B_2 \lll 1 \oplus B_{13} \ll 7) \tag{1.7}$$

Delirium uses the 200-bit permutation Keccak [56, 57], the LFSR $\varphi_{\mathsf{Delirium}}$ given by equation (1.7) and illustrated on Figure 1.12, and has tag size $\nu = 128$ bits.

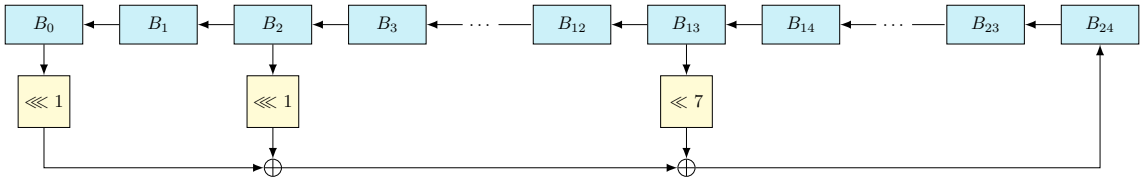


Figure 1.12: 200-bit LFSR $\varphi_{\mathsf{Delirium}}$.

The three n-bit LFSRs used for the variants of Elephant are the $GF(2)$-linear maps given at the byte-level by the equations (1.5), (1.6) and (1.7).

1.2.3.4.2 Sparkle

Schwaemm The main instance of Schwaemm is $\mathsf{Schwaemm}_{256-128}$ [43], illustrated in Figure 1.13. It takes

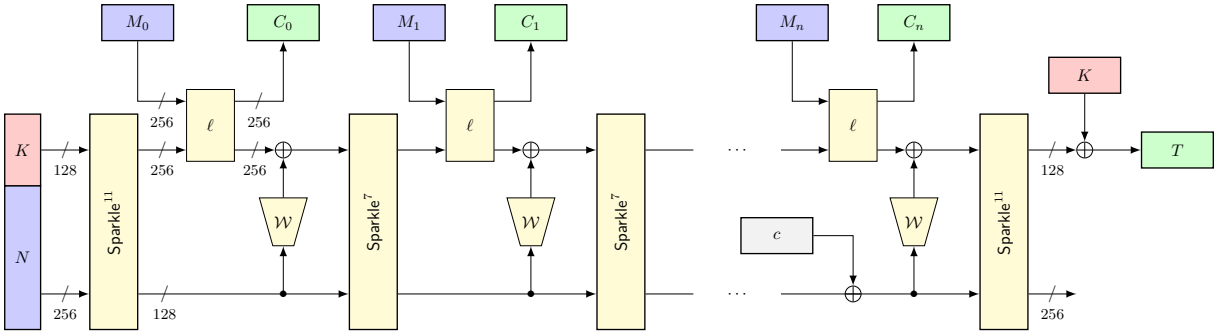


Figure 1.13: Scheme of $\mathsf{Schwaemm}_{256-128}$

a 256-bit nonce N, a 128-bit key K and output a 128-bit authentication tag T. The master key is:

$$K = K_1||K_2,$$

where:

$$|K_1| = |K_2| = 64 \text{ bits.}$$

$\mathsf{Schwaemm}_{256-128}$ is a permutation-based AEAD that builds upon the Beetle variant of the Duplex construction by adding an extra whitening layer, from the inner state into the outer state.

Zoom on $\mathsf{Sparkle}^{11}_{384}$ Sparkle is a family of cryptographic permutations. There are the permutations $\mathsf{Sparkle}^{r}_{256}$, $\mathsf{Sparkle}^{r}_{384}$ and $\mathsf{Sparkle}^{r}_{512}$ with block sizes of 256, 384, and 512 bits, respectively. The parameter r refers to the number of steps and a permutation can be defined for any $r \in \mathbb{N}$.

$\mathsf{Sparkle}^{7}_{384}$ and $\mathsf{Sparkle}^{11}_{384}$ are used in the authenticated encryption $\mathsf{Schwaemm}_{256-128}$, therefore the number of branches equal to 6.

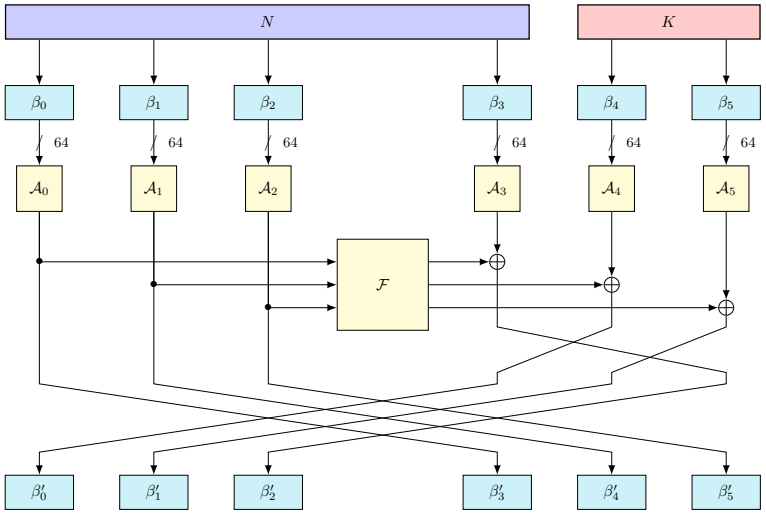


Figure 1.14: The first round of the $\mathsf{Sparkle}^{11}_{384}$ permutation

$\mathsf{Sparkle}^{11}_{384}$ takes as input N and K. So according to the Figure 1.14 N is loaded on the left and the first right branch, and K on the last two branches on the right.

The $\mathsf{Sparkle}^{11}_{384}$ permutations are built using the following main components :

- an S-box Alzette denoted $\mathcal{A}_i$, for $i \in [\![1, 5]\!]$, a 64-bit block cipher with a 32-bit key
- a Feistel type linear diffusion layer $\mathcal{F}$.

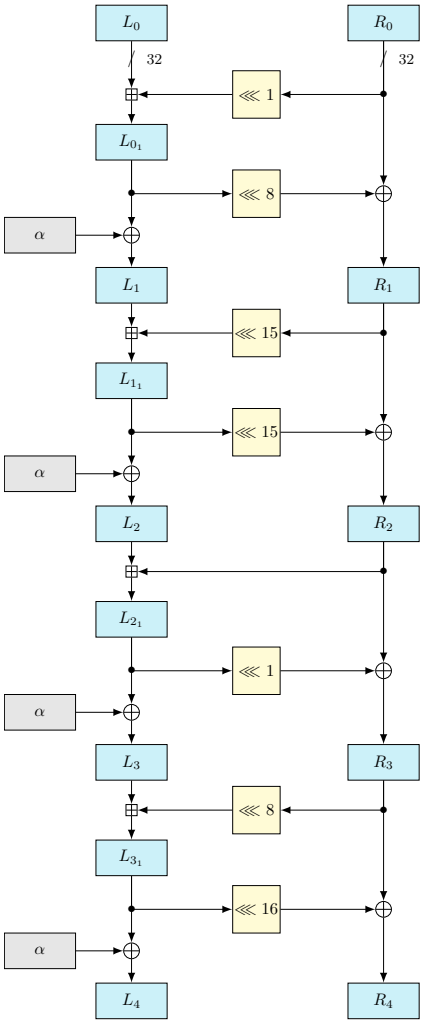


Figure 1.15: An Alzette $\mathcal{A}_i$ with a constant α

Zoom on Alzette Alzette, denoted $\mathcal{A}_i$, is a 64-bit block cipher. Alzette is defined as the permutation:

$$(L_0, R_0) \rightarrow \mathcal{A}(L_0, R_0, \alpha).$$

Alzette takes as input two 32 bits words. It is composed of four rounds where each round uses operations like modular addition, rotation and xor. After each round, the 32-bit constant is xored to the left word. Its round constant ensures that the computations in each branch of the Sparkle permutation are different from one another to break the symmetry of the permutation structure.

Alzette illustrated in Figure 1.15 can be described with the following equations mod 32:

$$
\begin{aligned}
L_{0_1} &= L_0 \boxplus (R_0 \lll 1); && (1.8)\\
L_1 &= L_{0_1} \oplus \alpha; && (1.9)\\
R_1 &= (L_{0_1} \lll 8) \oplus R_1; && (1.10)\\
L_{1_1} &= L_1 \boxplus (R_1 \lll 15); && (1.11)\\
L_2 &= L_{1_1} \oplus \alpha; && (1.12)\\
R_2 &= (L_{1_1} \lll 15) \oplus R_2; && (1.13)\\
L_{2_1} &= L_2 \boxplus R_2; && (1.14)\\
L_3 &= L_{1_1} \oplus \alpha; && (1.15)\\
R_3 &= (L_{0_1} \lll 1) \oplus R_3; && (1.16)\\
L_{3_1} &= L_3 \boxplus (R_3 \lll 8); && (1.17)\\
L_4 &= L_{2_1} \oplus \alpha; && (1.18)\\
R_4 &= (L_{0_1} \lll 16) \oplus R_3 \quad . && (1.19)\\
& && (1.20)
\end{aligned}
$$

1.3 Teaching

Given my enthusiasm for the topic, I intend to conclude this first chapter on teaching.

At IMT-Atlantique, I teach various types of students, including engineers, professional engineers pursuing master's degrees, and students in MSCS. Furthermore, I teach embedded security and cryptography in diverse contexts, including the IoT, blockchain, among others.

I am also invested in new pedagogical techniques, such as massive open online courses (MOOCs). I am a co-author, along with Gaël Thomas, of the cryptography section of Romaric Ludinard's MOOC on blockchain. I believe that fun and humour are crucial for facilitating learning. In this MOOC, cryptography is explained using farm animals. The sheep want to communicate and the wolf is the adversary. A link of these videos is: `https://www.youtube.com/playlist?list=PLjXls-kqM6JBZds1p4-lyyZmOv1xpH_CK`. I advocate for my work to include free content.

I am leading another MOOC project on physical attacks. For this project, I am collaborating with INRIA and ENSMSE, and I conduct guest interviews with people from Commissariat à l'Énergie Atomique (CEA), Agence Nationale de la Sécurité des Systèmes d'Information (ANSSI), and ST-microelectronics. The synopsis is as follows: a chicken wants to protect the hen house from the malicious fox. To do this, the chicken use authentication with `PIN` code. The fox uses physical attacks to try to break it.

Figure 1.16: Main characters of the different MOOC design by the MoonCat studio.

Another significant part of my activity is scientific popularization. I regularly go to middle schools and high schools to popularize science. I hope to transmit the taste for science, especially to young ladies. Writing an illustrated book about hardware security is a project that I hold dear for the future.

Chapter 2

Crypto ransomware

"*Dorénavant vous saurez qu'il ne faut jamais charger une arbalète avec un boomerang.*"
Gaston dans *Lagaffe mérite des baffes*
d'André Franquin.

Contents

2.1 Introduction

A **ransomware** is a malware that asks a payment to the legitimate users for accessing their machine or computer files. There are two types of ransomware: some prevent the usage of the computer, others encrypt files. So in this document, a **crypto-ransomware** is a malware which encrypts user data; then asks a ransom in exchange for the decryption key.

Ransomware constitutes a sensitive topic since they are rapidly spreading. They are one of the most serious security threats on the Internet. The real first occurrence of a ransomware was in 1989 [58]. But, the pandemic began in 2012, when the number of ransomware victims increased significantly. Cryptocurrencies such as bitcoin made the success of this kind of malware. Indeed there is a pseudo-anonymity of transaction actors. That probably explains why the number of ransomware victims has significantly increased.

2.1.1 Motivation

Ransomware attacks impact various sectors: education, entertainment, financial services, healthcare, *etc.* The number of attacks is in constant increase in recent years. The attack consequences are often catastrophic. So, it is very important to find solutions to this evil. Thus the motivation to join the malware battle specifically ransomware is the ethical motivation: find protection for the all society.

From my point of view, there is a very strong challenge in this battle. Before ransomware, encryption was always a tool to defend against attackers. Ransomware changes that and transforms encryption into a terrible weapon. So how to fight against our own robust tool? It is very stimulating for cryptographers to see encryption properties in a different view. That was my main motivation for this work.

2.1.2 Contributions

I started to work on ransomware in 2015. It was just the starting of the pandemic. So some countermeasures presented in this document, have evolved.

My contributions are on the topic of detecting and stopping ransomware infections. The main goal is to detect ransomware as fast as possible. In other words, with a minimal loss of files.

In practice, I was sequentially the advisor of two PhD students.

- The first was Aurélien Palisse. He defended his PhD "Analysis and detection of ransomware" [18] in March 2018. The director was Jean-Louis Lanet from INRIA and Colas Le Guernic from Direction Génerale de l'Armement Maîtrise de l'Information (DGA-MI) was the co-advisor.
- Then, Routa Moussalieb defended her PhD "Log Analysis for Malicious Software Detection" [33] in 2020. The directors were Jean-Louis Lanet from INRIA and Nora Cuppens from IMT-Atlantique at first, then Yann Busnel from IMT-Atlantique replaced Nora Cuppens.

The two PhD students worked hard to design an automated analysis platform called MoM (Malware O Matic). Countermeasures against the most specific behaviour ransomware have been developed [2–7]. Finally, a practical survey was published [1].

Then in 2021, I continued to work on this subject with David Lubicz from DGA-MI and Alexandre Gonzalvez at INRIA.

2.1.3 Organization

This chapter is organized as follows. The context and a short state of the art are presented in the section 2.2. In section 2.3 different solutions to detect and/or stop encryption by malware are presented. The section 2.4 proposes other countermeasures of ransomware. The platform of tests and some results are presented in 2.5. The conclusion and future works are drawn in the section 2.6.

2.2 Context

2.2.1 History

In recent years, the majority of organizations that have suffered a ransomware attack experienced significant impact to their business. Every area is affected: industry, individuals, hospital, etc. Indeed, many ransomware appear each year, others see significant upgrades. The Table 2.1 displays a glimpse of ransomware families, according to their initial release date. It represents the majority of the samples discussed and analysed in the literature.

Years	Ransomware
1989	AIDS Trojan
2005	Gpcoder
2011	Xorist
2012	Reveton, Cryptolocker, Graphtor.
2013	Cryptowall, Dirty Decrypt, Urausy, Kovter, Browlock, Nymaim.
2014	CTB-Locker, Tox, CryptoWall, Koler, TorrentLocker, Citroni, Dumb, Power Worm, Linkup, ViroLock, CryptoDefense, Bitman.
2015	TeslaCrypt 1.0 et 2.0, Locky, Umbrecrypt, Linux Encoder, Hi Buddy, TorrentLocker, Hidden Tear, Chimera, Fusob.
2016	Petya, Locky, Cerber, Zerber, Bart, DMALocker, CryptXXX, TeslaCrypt 3.0 et 4.0, Wannacry, Powerware, Jigsaw, Rokku, Radamant, Purge, Sage, Hydracrypt, Dharma, CrypVault, Deshacop, Gamarue, Shifu, Fsysna, Shade, Dalexis, Usteal, MRCR.
2017	GPCode, Ryuk, Spora, WannaCry V2, Jaff, Globe, Spora, BTCWare, NotPetya, CrySis, Nemucod, Cryptomix, Nemsis, Globe Imposter, DoppelPaymer.
2018	Gandcrab, Zeus, SamSam, BitPaymer/FriedEx/IEncrypt, Ryuk, LockerGoga, Deadmin, StopDjvuMoka.
2019	Megacortex, Maze, RansomOps, Conti, Sudinnokibi/REvil, Pysa, SuncryptRanzy, Nemty, Sekhmet, AKO, RansomExx, RagnarLocker, Ragnarok, Everest, Darkside, Avaddon, NetWalker, LockBit, SnatchClop, RobinHood, Eris, Phobos, Paradise, NemtyRevenge, Estemani, Ordinypt, Fsnyna.
2020	Egregor, Avaddon, Darkside, LV, Netfilm, Prometheus, Marketo, Mount Locker, Astro Team, Payload bin, Zeppelin, Maoloa, Conti, CLOP.
2021	MedusaLocker, Babuk, Xing, Lorenz, Grief, NoName, SynACK, Sodinokibi, Sodin, Katyusha, Pewcrypt, Lockergoga, Kaseya, Seon2, Hive, Ransomhouse.
2022	Everest, Lapsus, Blackcat, BlackBasta, Karakurt, Bianlian, Yanluowang, Quantum, RedAlert, Onyx, Cheers, Ragna Locker, LockBit V2.
2023	LockBit V3, Royal, Play, Mallox, ViceSociety, Stormous, ...

Table 2.1: A non-exhaustive list of the most famous ransomware according there first apparition.

The timeline in Figure 2.1 summarizes the important evolution of the ransomware.

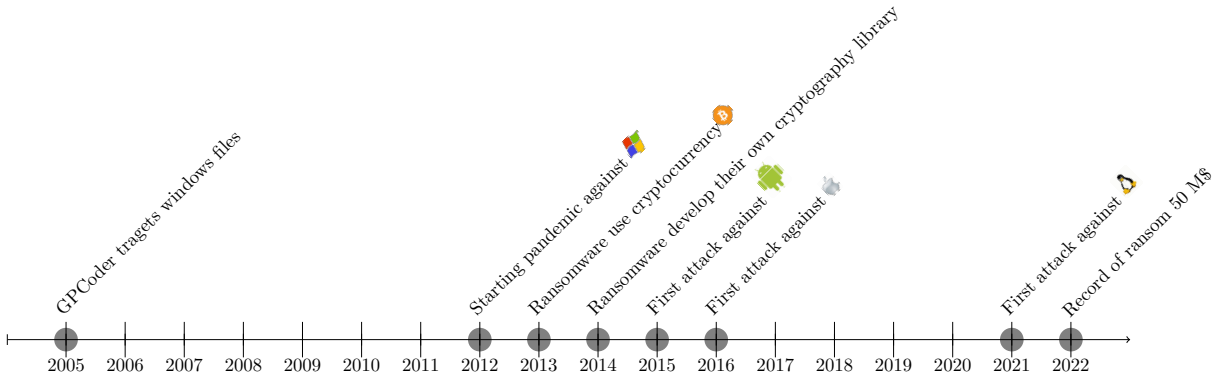


Figure 2.1: Timeline of pertinent evolution in ransomware.

The rise of ransomware started with the use of cryptocurrency. One has to remark that ransomware first targeted Windows, but now all OS are attacked.

2.2.2 Classic crypto ransomware behaviour

The classic behaviour of a cryptographic ransomware is described as follows.

1. **Infection vector**
   Generally ransomware comes from spam emails, self-propagation or drive-by downloads.

2. **Trigger of malicious behaviour**
   Many malware programs detect the user activity by social engineering methods, before exhibiting malicious activity. It could be opening a file, a web search on Google or a simple click.

3. **File System Traversal**
   Ransomware checks if the system has not already been encrypted, and then searches for files to be encrypted. It needs to perform a complete file traversal, they mostly use Windows Application programming interface (API) to find attached removable devices, and browse the file system. The ransomware browses files to find specific victims (.doc, .xls, .jpg, .pdf, .mp3)

4. **Initiation of Encryption process**
   Generally ransomware uses symmetric encryption as AES [46], described in paragraph 1.2.3.1.1, to encrypt data. The first step is to generate symmetric keys with a random number generator. One symmetric key is associated with one victims file. Then a public key is used to encrypt symmetric keys. Sometimes the public key is embedded with the code. Some ransomware programs use the crypto API of Windows while others use their own embedded cryptography.

5. **Encryption of data**
   Once the keys are generated, the encryption of files begins.

6. **Ransom note**
   A ransom note is displayed to explain how to pay to the victim. This marker must be significantly visible, being part of the business model.

7. **Ransom payment**
   Finally if the victim agrees to pay, the ransomware program rescues the files by handing the symmetric keys. In practice, it is often the case but sometimes the attacker do not restore data.

2.2.3 Summary of state of the art [1]

Different surveys have been published about ransomware protections [59–61]. In the context of the Routa Moussalieb's PhD, a survey [1] was published. The different countermeasures are classified according to the different steps of the infection process. Our survey focuses on Windows. A summary is given in this section.

The first and best idea to protect data against ransomware attacks is to use data back-up as explained in [62, 63]. In practice, it is not possible to have at any time an up-to-date back-up of data and to decide when and which data need to be restored. It is necessary to have a mechanism to detect a ransomware attack when it occurs.

The classic detection techniques used for all malware, such as network intrusion detection systems (NIDS) can be used for ransomware. Signature based detection remains a valid way to detect an infection. Network traffic is compared to previously well-known malware patterns, called signatures as in [6, 64–66]. The limitation of these techniques is that they are unable to detect new malware, never seen before. This is a serious problem with ransomware because new ransomware appears regularly.

The detection of specific behaviours of ransomware has to be used. Some detection mechanisms specific to ransomware can be to watch how files are touched [7] or to detect ransom notes writing. Some papers [67] studied the transaction for the ransom payment.

The most specific behaviour of ransomware is to overwrite files with encrypted data. The main problem is that these methods still generate a lot of false alarms.

Often ransomware does not come with their own cryptographic libraries and rather relies on the victim's own tools. So many protections are based on controlling accesses on the cryptographic tools in Windows [2, 68–70].

One other way to detect ransomware is to insert honey pots. These are files that trigger an alarm when they are modified [71, 72].

Finally, there is a family of ransomware detection mechanisms which rely on encryption detection. Indeed encryption produces data which have a pseudo-random distribution, therefore increasing the entropy of the files. There is an important literature about statistical tests of randomness which can be used to detect encrypted data [3, 73–76].

2.3 Encryption detection and protection

As explained in the section 2.1, detecting and stopping encryption is my main motivation in the battle against ransomware. Here, encryption is an attack and not a defence tool. The cryptographic properties are analysed as witnesses to the attack.

This section is split in three parts.

- Some ransomware programs use the cryptography tool of the victim. So our first idea is to monitor these tools. This approach [2] is summarized in 2.3.1.
- In 2015, ransomware with their own cryptography appeared. In practice, some of them do not use cryptography correctly. So classical cryptanalysis can help. More details are presented 2.3.2.
- The last part is the detection of encryption. The cryptographic properties of diffusion and confusion (presented in 1.2 implies a high entropy in the ciphertext. So detecting ciphertext writing is the same as detecting random data writing. In this way, two classical statistical tools are exploited: χ^2 test and Markov chain. More details are presented 2.3.3.

2.3.1 Monitoring the crypto libraries [2]

Cryptovirology refers to the use of cryptography to malware. The proof of concept of cryptovirology has been presented by Young et al. in [58]. Ransomware may use Microsoft's cryptographic application programming interface (MS CAPI). This section presents a generic countermeasure against a malicious usage of this API. This works was published in 2016 [2]. It was the first strong result of Aurélien Palisse PhD.

2.3.1.1 Microsoft's Cryptographic API

Windows operating systems, starting with Windows 95, include an easy-to-use API that supplies cryptographic services to userland applications through MS CAPI. Cryptographic primitives are embedded in dynamic link libraries and divided in cryptographic service providers (CSPs) each one offering a set of primitives classified by their type (hash function, signature, encryption). This API aims to provide an interface for programming by non-cryptographic specialists. Moreover due to its extensible architecture design, it is possible to add cryptographic modules once Microsoft signs them. For example, hardware security module vendors can implement white-box or home-made cryptographic algorithms and comply with federal information processing standard (FIPS) approved algorithm standards. It can be noticed that no user authentication is performed in order to access the primitives, authentication relies exclusively on the operating system. The framework does not provide persistent storage of keys or archival directly, you should use separate APIs provided by Microsoft. Moreover each time a process instantiates a dynamic link library (DLL), code is shared but data remains unique for each instance and thus no key is exposed at load time. Each cryptomodule is dedicated to one process so compartmentalization is ensured. Extended details can be found in [58, 77] which give a list of the exported functions and an in-depth documentation.

MS CAPI is old (appeared with Windows 95), but is still widely spread notably in banking infrastructure. For legacy or backward compatibility in newer Windows operating systems, MS CAPI is deprecated but still present; it is strongly recommended to use the cryptography API next generation (CNG) beginning with Windows Vista. Despite this recommendation, OpenSSL continues to initialize its deterministic random bit generator with some entropy coming from `CryptGenRandom` from MS CAPI.

No elliptic curve cryptographic (ECC) is supported with default providers, NIST Suite B Cryptography is fully available in CNG as well as additional chaining modes as for example GCM [50] presented in section 1.2.3.3.4 and the DES-X [78] block cipher.

One has to remark that the windows crypto API has evolved since these works (2016).

2.3.1.2 System Protection

2.3.1.2.1 Presentation In the context of protection against ransomware, the main question is: "How to prevent malicious usages of MS CAPI without intrusive methods" ? The goal is to provide a fully transparent solution, as simple as possible, and ransomware must not be informed about any analysis. Contrary to Bromium report [79] where an intrusive method based on the instrumentation library Detours [80] is chosen, a CSP is implemented and then plugged in the system. When incorporating a cryptographic module in the system, we are in a legitimate position to observe malicious cryptographic behaviour but also legitimate calls. We have to provide real services to user applications and so the OpenSSL library for cryptographic primitives is used.

2.3.1.2.2 Implementation The OpenSSL primitives are integrated in collaboration with MS CAPI architecture in order to get a functional provider. Version 1.0.2 dated from December 2015 is used and the provider was statically linked with the resulting library. The following services are provided:

- symmetric encryption with: AES (see in paragraph 1.2.3.1.1) used with CBC and ECB chaining modes (see in paragraph 1.2.3.3),
- asymmetric cryptography: RSA [81] # `PKCS1 v1.5 and v2.0`, RSA textbook,
- hash: MD5, SHA-1 and SHA-256 [82],
- source of randomness.

The above algorithms represent a set of cryptographic primitives sufficiently exhaustive to fit the needs of ransomware based on observations and reports.

2.3.1.2.3 Integration Cryptographic providers on both 32-bit and 64-bit architectures are compiled as DLLs. Providers are then placed in the System32 default path, and system integration is achieved through registry entries. Administrator rights are needed to plug a CSP in the system, registration is performed with `regsvr32.exe`. As explained in 2.3.1.1, the system cannot trust, and thus use, any provider until a valid signature from Microsoft is obtained. For our academic proof of concept, the binary responsible for the authentication mechanism to successfully load out CSP, is patched to authorize it without Microsoft's signature. As described in Figure 2.2, user applications start by calling MS CAPI exported `Crypt(*)` functions through `advapi32.dll`.

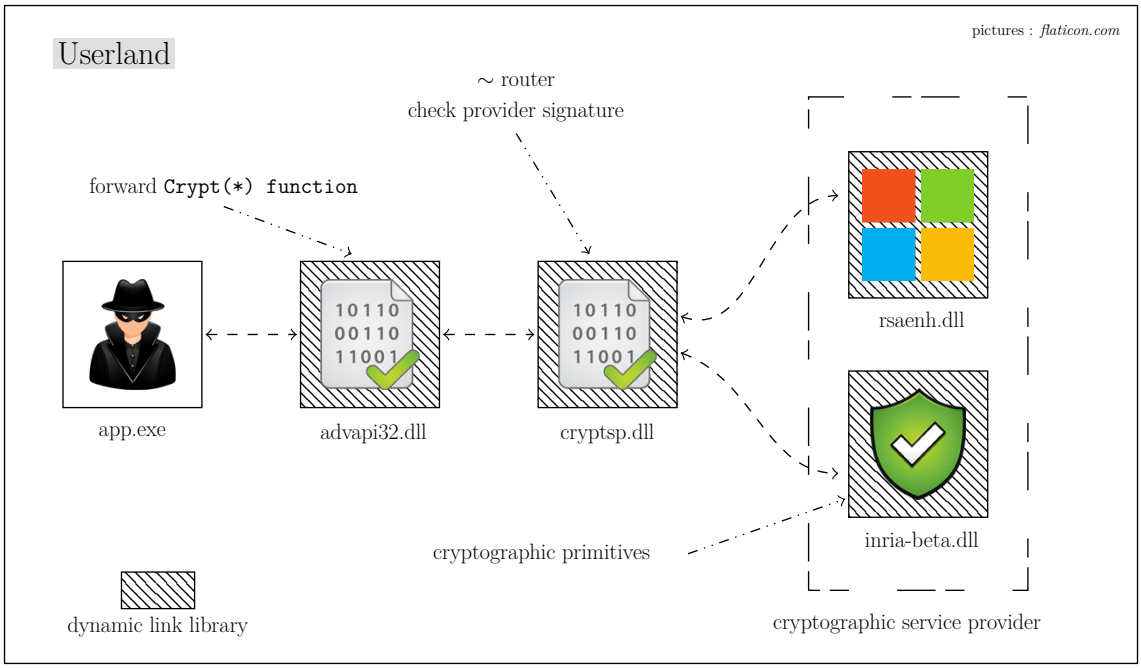


Figure 2.2: Windows PE and Crypto API (picture extracted from [18]).

In fact, this is nothing more than an indirect jump to `cryptsp.dll` which contains the CSP authentication mechanism and most of the framework functionalities.

Still, a ransomware program may use another provider. Indeed, during the initialization phase, a call to `CryptAcquireContext` with the argument `pszProvider` lets the user choose between available providers.

If the provider name is left null, a registry key indicates which is the default provider according to the requested type. So `cryptsp.dll` has been patched again to redirect all explicitly named provider to the default one. The final step is to set the name of the preferred cryptographic provider in registry as our own.

2.3.1.2.4 Detection The malicious behaviour must be stopped and the damage limited. In addition file recovery is needed to render ransomware harmless. We have the control of key generation, encryption, signature, randomness and integrity operations. At load time, no key exists within a CSP and usually deletion, exportation or public-key encryption is performed once operations are done. The advantage of this opportunity to monitor and store the secrets is if malicious activities are detected. Most interesting operations are logged in a special file prepended with a PE header and extension to prevent ransomware encryption.

The difficulty is now to detect any abnormal behaviour, no intelligence has been added to the module yet, but some possibilities can be investigated. MS CAPI is marked as deprecated so very few applications on Windows down to Windows XP use its services. In any case, an intensive use is suspect, particularly encryption operations.

2.3.1.2.5 Limitations It is required to place a patched `cryptsp.dll`. This is enough for our experimental purpose and characterization, bu is unsuitable for any real-world deployment. Signing our provider allows it to be loaded by the legitimate `cryptsp.dll`. Forcing its use is still a theoretical issue, in practice most ransomware use the default one: our own provider. To complete the current solution, it can be considered than ransomware detection based on a supervised machine learning classifier. The provider can be deployed on legitimate hosts then the normal behaviour MS CAPI can train a model. We do not want to see machine learning as a miracle oracle. So, user opinion has to be requested when a malicious behaviour is suspected. This solution can be incorporated in the provider and ransomware are never known about its existence. The solution presented in this paragraph 2.3.1.2 forces ransomware to embed required cryptographic primitives in their binary.

Many ransomware programs use the Microsoft's Cryptographic API. This fact avoids MS CAPI hijacking for manufacturers and aims to prevent malicious access. These two protections are placed at a different level and can be used complementary for best security, leading to a 50% success rate in our experiments. Unfortunately, they are not efficient against all ransomware, some ransomware does not use MS CAPI.

2.3.2 Replay attacks [2]

In this part, a protection against crypto-ransomware using block ciphers, published in [2] is presented. As explained in the section 1.2.3.3, a mode of operation is needed to encrypt data with block ciphers. In 2015, ransomware used two classic modes: ECB mode and CBC mode introduced in 1.2.3.3. The weakness of ECB chaining mode can be exploited to retrieve data even if a ransomware encrypted them. On the other hand, CBC exploitation involves more processing.

2.3.2.1 Ransomware using ECB mode

The ECB mode has a big drawback. Identical plaintext blocks are encrypted into identical ciphertext blocks; thus, it does not correctly hide data patterns. The figure 2.3 illustrates this problem.

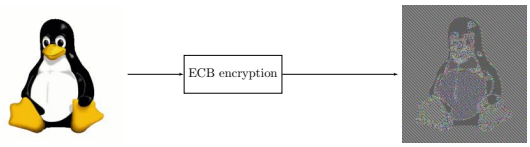


Figure 2.3: ECB mode limitation: all the same plaintext block is encrypted with the same ciphertext block. In consequence: you can still recognize the penguin.

With such a mode, a replay attack [83] is possible. The main idea of our protection is to use the disadvantage of ECB mode to protect our data against ransomware. For this purpose, our protection consists in a smart data encoding. The first step is to expand the data. Each byte of data is padded with 0 bytes to have block size n. In the case of the AES, $n = 16$. So a data file M of size ν bytes, such as:

$$M = M_0, M_1, \cdots M_\nu;$$

is transformed in an expanded data file such as:

$$M_{\text{expanded}} = M_0|\underbrace{0\cdots0}_{n-1}|M_1\underbrace{0\cdots0}_{n-1}\cdots M_\nu|\underbrace{0\cdots0}_{n-1}. \tag{2.1}$$

Additionally a dictionary is created:

$$\text{dictionary} = 0|\underbrace{0\cdots0}_{n-1} \quad 1|\underbrace{0\cdots0}_{n-1} \quad 255|\underbrace{0\cdots0}_{n-1}$$

If ransomware uses the same key to encrypt all files, it also encrypts the dictionary file. The user can retrieve all files thanks to it by matching the encrypted blocks of a file with the encrypted blocks of the dictionary for which the unencrypted values are known.

If ransomware uses a different key for each file, the dictionary can be created at the beginning of each file. So a file M is expended in " dictionary+M_{expended}":

$$\text{dictionary} + M_{\text{expanded}} = 0|\underbrace{0\cdots0}_{n-1} \quad 1|\underbrace{0\cdots0}_{n-1} \quad 255|\underbrace{0\cdots0}_{n-1} \quad M_0|\underbrace{0\cdots0}_{n-1} \quad M_1|\underbrace{0\cdots0}_{n-1}\cdots M_\nu|\underbrace{0\cdots0}_{n-1}.$$

Another solution of defence is to use the entropy of the original file. If M is a text, it can be supposed that the user knows the language used. In this case, no dictionary is necessary, text files are just expended (2.1) and the entropy is used to retrieve data with a classic basic cryptanalysis. Usually the file name is not encrypted, so the entropy could be stored in the file name to avoid the use of a dictionary.

2.3.2.2 Ransomware using CBC mode

If a ransomware program uses a block cipher in CBC mode, the solution described in 2.3.2.1 fails, because of the xor at the start of block encryption. This paragraph presents a more complex protection for this mode, working if the ransomware encrypts newly created file with the same key as the file you want to retrieve. The

data files are expended exactly as in (2.1); but the construction of the dictionary is different. The user does not create a dictionary file, but 256 files dictionary_B^0, one for each possible value of byte $B \in [\![0, 255]\!]$ such as:

$$\text{dictionary}_B^0 = (B|\underbrace{0 \cdots 0}_{n-1}).$$

The ransomware encrypts all data files and the 256 dictionaries. At this step, the user can retrieve the first and only the first byte M_0 of each encrypted files. Then they can create new dictionaries, one for each encrypted file and for each possible value of byte $B \in [\![0, 255]\!]$ such as:

$$\text{dictionary}_B^i = (B|\underbrace{0 \cdots 0}_{n-1}) \oplus C_0.$$

Then the ransomware encrypts the new dictionaries. At this step the user can retrieve all second bytes M_1 of the files. So the user can retrieve all bytes M_i by recurrence in the dictionaries byte:

$$\text{dictionary}_B^{i+1} = (B|\underbrace{0 \cdots 0}_{n-1}) \oplus C_i.$$

The main problem to these countermeasures is the sizes of the files which significantly increase. The data size is multiplied by n. In our description, the size of dictionary elements is fixed to one byte. In practice, it can be any other size. The bigger the size element is, the bigger the dictionary and the file sizes. In future works, it would be interesting to optimize these sizes. It is important to specify that the countermeasure against CBC mode is limited to a ransomware program which:

- always uses the same key K and the same vector IV,
- encrypts all newly created files.

Obviously another limitation is the fact that some ransomware use other modes such as GCM.

2.3.3 Statistical Tools

Ransomware involves a large number of ciphertext going through the file system; to detect such behaviour, statistical tests can be used. The main idea is that ciphertexts content distribution should be hardly distinguishable from a sequence of independent and identically distributed (IID) uniform random values. In this context many statistical tools are used to detect encrypted data, which are similar to random data.

One sample goodness of fit test measures how close an information source is to a theoretical probability distribution function, also known as the "model". In practice, one data set is seen as a distribution X in the set $\mathbb{X}$. This distribution X is compared to a known distribution function F. Then we disprove or not the null hypothesis:

$$\text{H0} : \forall x \in \mathbb{X}, x = F(x). \tag{2.2}$$

It does not prove that both data sets come from the same distribution, but rather that there is no significant difference between them. The objective is to figure out which indicators of compromise are the most relevant to detect ransomware attacks in real time.

The common statistical tool to detect random data is the **Shannon entropy** denoted by H. It is a measure of the uncertainty of a random variable. Let X be a random variable, the entropy H is:

$$H(X) = -\sum_{x \in \mathbb{X}} \mathbb{P}(X = x) \cdot \log_2(\mathbb{P}(X = x)). \tag{2.3}$$

For uniform data, H is maximal. For example, if X is a uniform random distribution of bytes, then $H(X) = 8$.

Authors of the publications [73,76] use Shannon entropy to detect ransomware encryption activity. The main difficulty is that a lot of data are required to estimate Shannon entropy. However, in the case of ransomware detection, it is important to detect infection with as few bits as possible. In this context we have been suggested two other tools:

- the χ^2 test to replace Shannon entropy (2.3), in the section 2.3.3.1;
- Markov chains in the specific case of header files distribution, in the section 2.3.3.2.

2.3.3.1 χ^2 test [3]

This paragraph presents a protection against crypto-ransomware, published in [3].

A **statistical hypothesis test** is a method of statistical inference used to decide whether the data at hand sufficiently support a particular hypothesis. The **χ^2 test** is a statistical hypothesis test. The test is valid when the test statistic is chi-squared distributed under the null hypothesis (2.2). It is a test of distributional accuracy, it measures how closely a set of numbers follows a particular distribution. It is a non-parametric statistical test, meaning that no assumption is made on the sampled distribution.

The observed sequence of data consists of bytes. So it is discrete and arranged in a frequency histogram $[\![0; 255]\!]$ with the degree of freedom equal to 255 (*i.e*, number of possible outcomes minus one). Suppose that Γ_B is the number of hits observed for the byte B, and γ_B is the expected number according to a known distribution function F. The formula for calculating the one-sided χ^2 test is:

$$\chi^2 = \sum_{B=0}^{255} \frac{(\Gamma_B - \gamma_B)^2}{\gamma_B}.$$

A large value indicates that the null hypothesis is not likely verified, the Γ_B is not drawn from the known distribution. The significance level of the test is the probability of rejecting the null hypothesis when it is true. Traditionally, experimenters have used the 0.05 level (e.g, biology), thus we choose the same. The observed test is compared to a boundary value, called the critical value, uniquely determined by the degree of freedom (or equivalently the size of the alphabet) and the desired significance level. If the χ^2 result is more extreme than the critical value [84], the null hypothesis is rejected.

Another problem on the statistical level is that we want to distinguish malign activity from normal behaviour. But if we have a clear statistical model for malign activity, namely random data, we do not have its counterpart for normal behaviour. In this situation, the use of classical χ^2 test is awkward, to say the least, since they are meant to detect improbable events under the hypothesis that the stream of data is random where we actually want to single out improbable events in the case that the stream of data is a normal structured file. This makes it impossible to carry out a precise analysis of the efficiency of the statistical hypothesis tests that using for the detection of ransomware and to design most efficient tests.

In conclusion, this solution is based on the χ^2 test which measures the frequency distribution. This detector is particularly well suited to detect a file encryption. In our experiments $99,37\%$ of ransomware programs have been detected. But it is unable to distinguish an authorized encryption tool, as for example `OpenPGP`, or compressor, as for example `WinZip`, from a ransomware, thus generating false positive (FP).

2.3.3.2 Markov Chain [4]

This paragraph presents a protection against crypto ransomware, I published in [4] realized with Nicolas Bailluet, a student of Rennes 1 University and David Lubicz from DGA-MI.

This work introduces another statistical tool, to detect ransomware based on their encryption behaviour. The main idea is the fact that the header files are very structured data. When a ransomware encrypts a header the structure is changed. To deal with the few data of header files, another statistical tool is used: the binary Markov chain.

2.3.3.2.1 Definition A **binary Markov chain** X is a sequence $(X_j)_{j\geq 0}$ of random variables defined by the following data:

- a memory m, that corresponds to the number of bits saved in a state X_i.
- a transition matrix $\mathcal{T}$ such that:
$$\mathcal{T}(x, y) = \mathbb{P}(X_i = x | X_{i-1} = y);$$
- an initial state X_0 which is a random variable.

$X(m, \mathcal{T}, X_0)$ denotes the Markov chain with memory m, transition matrix and initial state X_0.

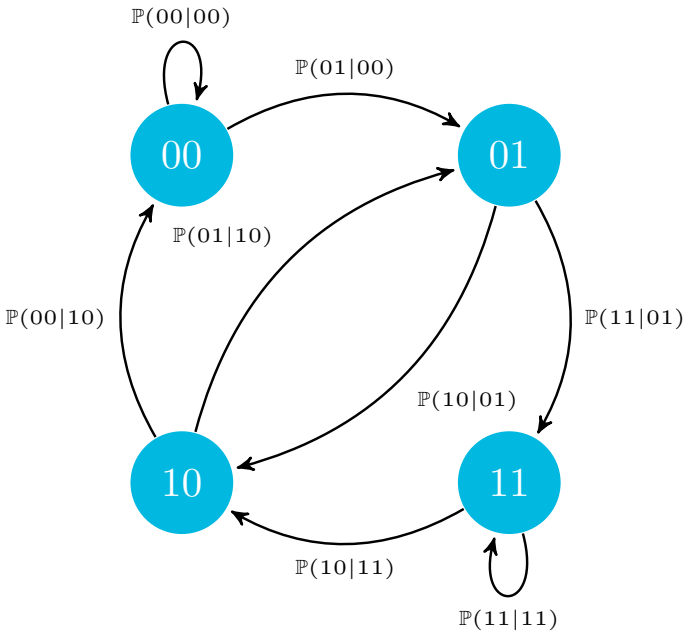


Figure 2.4: An example of Markov chain with $m = 2$ bits, 2^m states and transition probabilities $\mathbb{P}(X_i = y|X_{i-1} = x)$.

2.3.3.2.2 Training the Model The main idea is to intercept write requests, then decide if a sample corresponds to a benign or a malicious write request. To do so, in a learning phase, statistical models of structured file headers are built using Markov chains.

A Markov chain of memory m is used, with the first n bits of each file.

- Let L_n be the list of the first n-bit sequences of each file in the corpus.
- Let $\mathbb{S}$ be the set of the possible successor s.
- Let $X(m, \mathcal{T}, X_0)$ be the Markov chain which represents the best statistical approximation of the sample L_n.
- Let s be a successor of L_n defined in $\mathbb{S}_m$.

To define this Markov chain $X(m, \mathcal{T}, X_0)$, it is needed to compute $\mathcal{T}(s)$ and $\mathbb{P}(X_0 = s), \forall s \in \mathbb{S}_m$.

- $\forall s \in \mathbb{S}_{m+1}$, let $o_{L_n}(s)$ be the occurrence count of s in every sequence of L_n.
- $\forall s \in \mathbb{S}_m$, let $o_{L_{n,0}}(s)$ be the number of sequences in L_n such that s is the m first bits.

$|L_n|$ denotes the size of the list L_n, *i.e* the number of files. Finally, let $|L_n|(\nu) = (n - \nu) \cdot |L_n|$ be the number of sequences of bits of length ν in L_n. Thus $\forall s \in \mathbb{S}_{m+1}$ there is:

$$\mathcal{T}(s) = \frac{o_{L_n}(s)}{|L_n|(m+1)} \quad ;$$

and $\forall s \in \mathbb{S}_m$:

$$\mathbb{P}(X_0 = s) = \frac{o_{L_{n,0}}(s)}{|L_n|}.$$

On the other hand, let $X^{En}(m, \mathcal{T}^{En}, X_0^{En})$ be the Markov chain with memory m which represents the best statistical approximation of an encrypted file. As the statistical behaviour, the encrypted file is indistinguishable from the one of a perfectly random sequence. Thus $\forall s \in \mathbb{S}_{m+1}$ there is:

$$\mathcal{T}^{En}(s) = \frac{1}{2^{m+1}} \quad ;$$

and $\forall s \in \mathbb{S}_m$:

$$\mathbb{P}(X_0^{En} = s) = \frac{1}{2^m}.$$

2.3.3.2.3 Testing Samples against models Then in a detection phase, a maximum likelihood test is used to decide if a sample provided by a write request is normal or malicious.

Let $X(m, \mathcal{T}, X_0)$ and $X^{En}(m, \mathcal{T}^{En}, X_0^{En})$ be two Markov chains. The first is a model of a normal header file, the second represents encrypted data. Let $s \in \mathbb{S}_n$ be a sample header. The goal is to decide which of the two trained models is the more likely to match s. For this, the likelihood ratio test is used.

Let $s \in \mathbb{S}_\nu$ be a sample, the **likelihood ratio** Λ of s with respect to the two Markov chains, is defined as:

$$\Lambda(s, X(\nu), X^{En}(\nu)) = \frac{\mathbb{P}\left(X(\nu) = s\right)}{\mathbb{P}\left(X^{En}(\nu) = s\right)} \quad .$$

The **likelihood ratio test** then is defined by:

$$\begin{cases} 1 & \text{if } \Lambda(s, X(\nu), X^{En}(\nu)) > \alpha \\ 0 & \text{otherwise} \end{cases} \quad .$$

Here α is a constant that allows setting the significance level of the test.

The Neyman-Pearson lemma [85] states that this likelihood-ratio test is the most powerful test for a given significance level.

2.3.3.2.4 Results and limitations To test the trained models against randomness, we have created for each format a corpus half composed of randomly generated files and half composed of files that match the format and were not in the original training corpus. We have used the likelihood ratio test with the constant $\alpha = 1$. The test is positive if it says that the sample corresponds to a structured header and negative if it says that the sample is random.

Our countermeasure based on Markov Chains to detect ransomware is able to distinguish the structured files from random data. This approach is applicable to real-world cases by successfully distinguish structured files from files encrypted with known ransomware whose encryption schemes fit into our models. For the experiments, we have trained multiple models on different file formats: `jpg`, `gif`, `msoffice` and `pdf`. The success rate is of 100% with encrypted data by us. But this solution cannot detect a ransomware which does not encrypt the headers of files.

It is a low cost in terms of computing. It can detect ransomware with only a few processed bits. Optimal results are with a memory $m = 16$ bits and a size $n = 512$ bits. Moreover, the Markov chain models have a low computing cost. So they can be deployed in the kernel to stop malicious processes without slowing down too much the computer.

This countermeasure is not enough on its own. But it's a pertinent additional tool to complete other implemented countermeasures against ransomware. In the fight against ransomware, the more indicators we have, the better is the detection.

2.3.3.3 Distance to a model [5]

This work was realized with Yassine Lemmou a PhD sudent from the University Mohammed V at Rabat and Jean-Louis Lanet from INRIA, then published in [5]. It is an improvement of first countermeasures which detect an encryption. Indeed, the problem is how to distinguish an authorized encryption from a ransomware activity. In this work, the main idea is to use the memory snapshot as a string vector. The distance of a candidate software with a vector representing the allowed cryptographic software is measured. If the distance exceeds a threshold, the suspected process is flagged as a ransomware.

2.3.3.3.1 Memory snapshot To be executed, an executable program must be located in memory. The aim of this work is to demonstrate that snapshots memory can be used to extract relevant information.

If a program is flagged as suspicious with previous indicators such as the χ^2 test presented in 2.3.3.1 or the Markov chain test on the header presented in 2.3.3.2, an analysis of the memory snapshot starts. More precisely, a driver transforms the memory snapshot into a binary vector v that marks the presence or the absence of specific strings. These specific strings rely on the allowed cryptographic tools. Then the vector v is confronted

to a model vector $\mathbf{v}$ to discriminate a malicious or benign program.

When a memory snapshot is analysed, the first step consists in removing useless information from memory. A snapshot contains thousands of strings. Some of them are just noise (string with no semantics) as code sections or previously allocated but uncleaned memory. The first step consists in removing information that is useless for the detection. Two elements are of importance: the loaded DLLs and the name of the functions called. To keep only the strings with relevant meaning, a dictionary approach (white list) is used. It reduces the number of strings by one order of magnitude keeping only relevant information. An efficient hash map structure is used for the dictionary. At that step, the strings are only valid strings. The vector v is built counting their raw value in the memory. v represents a software in two parts: the DLLs section and the function names section. The DLLs part is of a fixed length. It corresponds to all the allowed software encountered at that time. Only a valuable part of the strings is selected.

2.3.3.3.2 term frequency-inverse document frequency (TF-IDF) To compare the vectors dynamically, it is required to normalize its size. For that purpose, information retrieval techniques are applied to select a subset of the strings in the vector. The set of strings of a vector can now be considered as a document and the set of valid applications as a collection of documents. The term frequency-inverse document frequency (TF-IDF) is used to choose the relevant strings to be kept in the vector. The value associated to each term has the following explanation:

- highest when the term occurs many times within a small number of documents leading to a high discriminating power;
- lower when the term occurs fewer times in a document, or occurs in many documents;
- is the lowest when the term occurs in almost all documents.

2.3.3.3.3 Distance to the model In this paragraph, the distance between a candidate vector v (the suspicious one) and the model $\mathbf{v}$ is evaluated. The vector v can be seen has the distribution of the strings in the memory. Various distance/similarity measures are available in the literature to compare two data distributions. To measure the divergence, two metrics are tested, the Euclidean and the cosine distances. The **Euclidean distance** (2.4) between each element of the candidate vector v and the model $\mathbf{v}$ is used, assuming n being the size of both the vectors.

$$\text{distance}_{\text{Euclidean}}(\mathbf{v}, v) = \sum_{i=1}^{n} \sqrt{(\mathbf{v}_i - v_i)^2}. \tag{2.4}$$

It is the ordinary distance between two points.

The **cosine distance**(2.5) is used too. Even if at the end only one metric will remain in the driver.

$$\text{distance}_{\text{cosine}}(\mathbf{v}, v) = \frac{\mathbf{v} \cdot v}{||\mathbf{v}|| \cdot ||v||} = \frac{\sum_{i=1}^{n} \mathbf{v}_i \cdot v_i}{\sqrt{\sum_{i=1}^{n} \mathbf{v}_i^2} \cdot \sqrt{\sum_{i=1}^{n} v_i^2}} \tag{2.5}$$

2.3.3.3.4 Results and other utilization This solution is an improvement of previous solution to reduce FP. Indeed our previous solution detects the encryption. This approach reduces by 100% the FP generated by the χ^2 test solution, presented in 2.3.3.1.

An other utilization of the analysis of memory snapshot is possible in the case of ransomware. More precisely, ransomware writes a ransom note for the user. So specific terms and strings can be retrieved in the memory snapshot.

2.4 Other protection mechanisms

Even if stopping encryption is the main subject, this section presents other countermeasures to stop ransomware. These solutions are not all specific to ransomware infection. Moreover they are not focusing on the encryption step. That is why these solutions are less detailed.

2.4.1 File system traversal behaviour [6]

This section summarizes a ransomware detection technique that serves as an intrusion detection system (IDS), developed by my two PhD students Routa Moussalieb and Aurélien Palisse and published in [6].

This work is more a classification of crypto-ransomware, than a practical detection tool. It focuses on the file system exploration, before the attack (encryption) takes place. Indeed, once it is unpacked, the ransomware payload goal is to explore the file system to find files to encrypt. This search is done from the root of the file system or directly from the user's folder with a depth-first or a breadth-first search. As for the exploration phase, threads that traverse the file system behave similarly and predictably, enabling the possibility of an early detection of the ransomware and deducing its family.

2.4.1.1 Monitoring

Our solution detects ransomware behaviour based only on monitoring file system traversal. Each ransomware's file system traversal can be compared to others in order to know if they belong to the same family, or if they share some code regarding the paths exploration. We have concluded that the majority of ransomware start their encryption process from the root of the hard disk. To get a precise ransomware's classification, classical supervised machine learning techniques are used :

- K nearest neighbour [86],
- decision tree [87],
- random forest [88].

Another behavioural property is additionally investigated, the execution time (the velocity of the file traversal) of a family. Indeed, the samples issued from the same families have similar patterns. Each payload can be packed or obfuscated individually, but the system impact remains the same for a particular family with the exception of new versions, as seen in Figure 2.5.

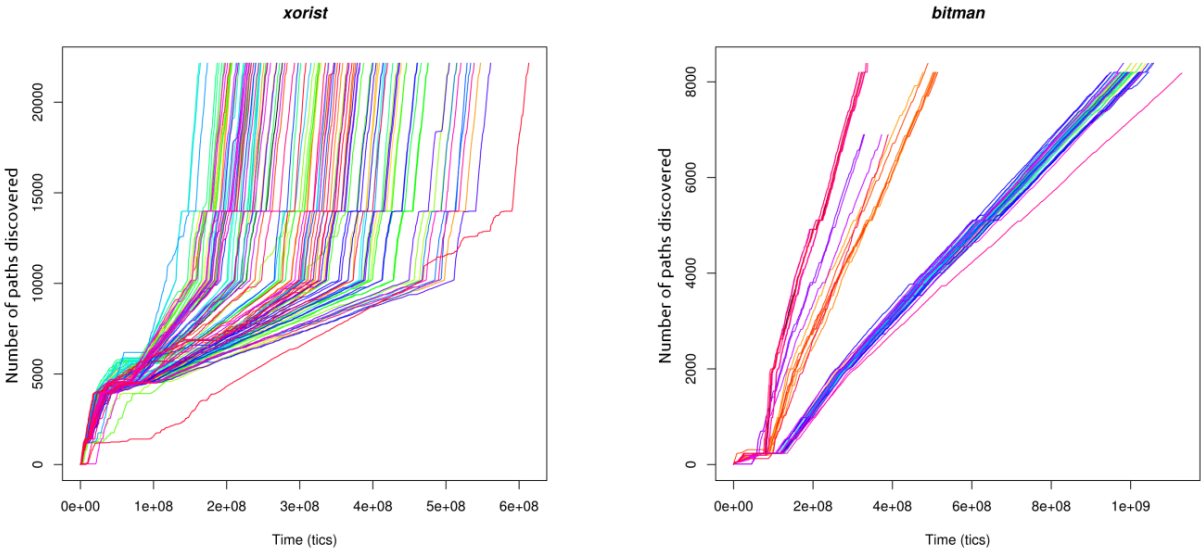


Figure 2.5: The file system's traversal velocity of 125 Xorist samples and 57 Bitman samples.

To each ransomware program, it is possible to obtain a signature of the file system's traversal and its velocity.

2.4.1.2 Limitations

Any software that mimics the behaviour of ransomware's traversal is classified as malicious so our proposal raises false positives in this case. Another limitation occurs when no file system traversal is done prior to encryption.

2.4.2 Ransomware network traffic analysis [7]

This section presents results developed in Routa Moussalieb's thesis and published at [7]. The main idea is to reconstruct ransomware's full activity to check if its network communication is distinguishable from benign network traffic.

In this work, ransomware is detected by the network traffic. Multiple malware samples have been executed. The infection mechanisms such as spam email to detect ransomware download process before its installation on the victim's computer is examined. It results of this work that the majority of ransomware behave similarly and not in a very specific manner of this malware family. Common patterns are found among various families. To get a precise ransomware detection, machine learning techniques are used. To sum up, network alerts represent a first suspicion means informing the user of the presence of potential ransomware. However, some drawbacks exist. This first alarm can take place after the creation of encrypted files or ransom notes as we noticed with some families. In addition, few elements are needed for a classification, we have under-fitting problems (Zerber samples), prone to adversarial attacks. Besides, only decision trees [87] among the tested algorithms provided high detection rates for zero-day attacks. For all the reasons mentioned before, network alerts should be backed up with system data to provide a general detection mechanism, working on all types of ransomware.

As for our future work, we will gather additional information from the file system to present a multi-layer alert strategy to detect ransomware payload as early as possible.

2.5 Experimentations

2.5.1 Malware-O-Matic

During the PhD of Aurélien Palisse [3,18] with the help of a Master student Antoine Durand, the platform MoM has been designed and built into the LHS.

MoM is an automated malware analysis platform that does not use virtual machines, while keeping all the main features of a regular analysis framework. This fully bare-metal platform is built on top of two open-source software, Clonezilla [89] and Viper [90], which makes the experiments reproducible. The platform is made of a single master server and several slaves, each one running the analysis loop in parallel. The whole system is on a dedicated network under the supervision of the network autonomous system and directly connected to the Internet, without using the institute intranet,to emulate a typical home network.

The loop itself consists of a few simple steps:

- setup of the monitoring environment,
- malware execution,
- results gathering and storage;
- clean-up.

MoM is able to analyse up to 360 malware programs per-day with only 1 server and 5 slaves. The end goal of this platform is to run uninterruptedly and thus automate the analysis of malware samples.

MoM can be used in two distinct modes for the experiments: "passive" or "active". In passive mode she only observes behaviour of ransomware. Ransomware has a short time life. So the passive mode of MoM is used to test if a ransomware program is always active The second mode evaluates ransomware countermeasures, especially countermeasures against malicious encryption.

2.5.2 Data Aware Defence

Data aware defence (DaD) is the piece of software to evaluate the in-house countermeasures on real ransomware samples. So DaD is an important part of MoM. Countermeasure developed in DaD can detect and block in real time most of the ransomware in the collection.

Similarly to approaches [91–94] reporting satisfying detection rates, DaD monitors file system activity. An important part of this contribution is the usability of the solution; furthermore, it can be used against zero-day ransomware.

2.5.2.1 File System Activity Monitoring

Windows, as most modern operating systems, splits its memory in several regions with different privilege requirements. The kernel mode (ring 0) has a high privilege level and is responsible, among other things, for managing disk operations.

Standard applications run in userland (ring 3), they are much less privileged and cannot perform disk operations directly. Instead they call the corresponding service from the Windows kernel, let the kernel manage the privileged operations, before safely returning in the userland application code. This separation ensures that no userland code directly manages critical operations in the system. To complete this security feature, the 64-bit versions of the operating system (OS) requires all kernel mode code to be signed by Microsoft to be accepted.

Up to now, to the best of our knowledge most ransomware lives in userland. That is why a countermeasure in kernel space cannot be tampered with by malicious code and is fully transparent. Users' interactions with files are ultimately mapped to operations in the kernel. On top of this stack stand the input/output (I/O) manager and at the bottom, the file system driver (`ntfs.sys`). Microsoft offers a file system drivers framework [95] that allows third-party developers to add functionalities between the two previous layers. Such component is called a file system minifilter driver and it is managed by the filter manager. The position of each minifilter driver in the I/O stack is defined by its "altitude". A minifilter driver that performs full disk encryption is below an anti-virus filter and thus avoid false positive detection of ransomware-like behaviour. A minifilter driver can inspect all the operations that target the disks, regardless of whether the requested operation is an I/O request packet or a fast I/O. In this context DaD is able to monitor write, read operations and so on. However a clever usage of such functionalities has to be done, otherwise a significant performance penalty occurs [94] and the solution cannot be deployed in the real world. Our file system minifilter driver has been extensively tested on Windows 7 and 10, for the 32 and 64-bit versions and follows the Windows Driver Model.

2.5.2.2 Implementation Design

The Figure 2.6 represents the DaD simplified architecture.

In order to catch malicious behaviours efficiently, the monitoring is restricted to write and information operations. The goal is to demonstrate that detecting ransomware behaviours with only two hooks on the `I/O` requests is possible. The so-called information operations allow changing various information about a file object, create a hard link, change the file position or the file name. The information operations are blocked once a thread is marked as malicious to prevent aggressive files renaming. But without malicious activity, the information operations are always accepted and are not used to construct the compromise indicators. For each intercepted write operation, the time spent in the callback function is minimized by collecting only the essential information. Only the content of the buffer is interesting. It is passed through the file system stack, its size, offset, the corresponding absolute file name, process name, process id and thread id. This information is then copied to non-paged memory and passed over to a new dedicated thread that is not part of the file system stack. Only the indispensable data is copied, not the thread context coming from the operation. As a consequence, write operations are immediately authorized to go through. So far, we were the first to inspect the I/O operations with a thread granularity. It is a significant improvement in case of malicious code injection into a benign process.

All costly computations are deferred to threads running at the lowest kernel priority level. This model solves the time restriction for the statistical computations and the synchronization deadlocks on shared resources.

2.5.2.3 Indicators of compromise

Today, DaD uses one or two indicators of compromise in the same times. The indicators of compromise developed in DaD are monitoring API windows (section 2.3.1), χ^2 (section 2.3.3.1), Markov chains (section 2.3.3.2), distance metrics (section 2.3.3.3) and the file le system traversal behaviour (section 2.4.1). An important objective is to combine all the different indicators for a same execution in MoM. Another objective is to have the

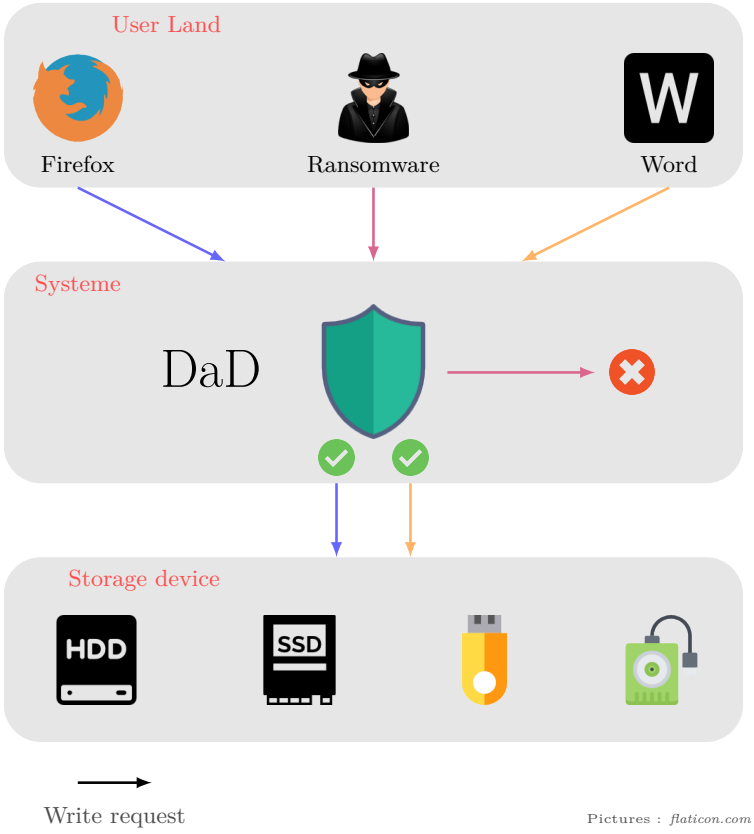


Figure 2.6: Data aware Defence DaD

network solution developed in the thesis of Routa Moussalieb (section 2.4.2). The more indicators we have, the more ransomware programs are stopped. Also, the more indicators we have, the more false positives we have. And the goal is not to block the computer at each benign software execution. That explains why, DaD is in constant evolution at the LHS of INRIA Rennes. Moreover all countermeasures presented in this manuscript are not yet integrated in DaD. Today, an engineer Alexandre Gonzalvez from INRIA and David Lubitz from DGA-MI are still working to improve them. I continue to participate to this subject, with them.

2.5.3 Results

Many ransomware programs have a short life. As a result, it is very difficult to compare different countermeasures between them. Indeed it is impossible to keep the same samples over time.

However, the Table 2.2 summaries all results of our different proposed countermeasures. More precisely, it lists all ransomware tested and stopped or not by the different works. It gives the success rate, i.e. the percentage of ransomware tested and detected.

Often for one ransomware, there are different versions. That explains why, for example, the χ^2 test stopped some Xorist and not all Xorist.

The solution which uses Markov chain is different. The ransomware are stopped on specific files (`pdf`,`msoffice` and`jpeg`) and not for other specific files(`gif`).

An interesting result can be to combine them. It is the work of Alexandre Gonzalvez engineer at INRIA.

Countermeasures	year	success rate	**ransomware detected**	**ransomware tested but not detected**
Crypto API Monitoring [2]	2016	50%	Gpcode,CryptoLocker, CTB-Locker	TeslaCrypt, Petya, Gpcode
Replay attacks [2]	2016	37, 5%	Gpcode, TeslaCrypt	CTB-Locker, Petya, CryptoLocker
χ^2 test [3]	2017	99, 37%	Teslacrypt, Cerber, Xorist, Bitman, Deshacop, Yakes, Zerber, Locky, Gpcode, Gamarue, Shifu, Fsysna, Shade, Dalexis, Usteal	Xorist
File system traversal [6]	2018	100% (need learning)	Teslacrypt, Cerber, Xorist, Bitman, Deshacop, Zerber, Yakes, Locky, Gpcode	
Network level [7]	2019	73, 33%	Bitman, Cerber, Shade, TeslaCrypt, Zerber	Bitman, TeslaCrypt, Yakes, Zerber
χ^2 test + cosine distance [5]	2019	96, 05%	Bitman, Cerber, Fsnyna, Telsacrypt, Xorist, Yake, Zerber, Crysis, CTB-Locker, Gamarue, Graphtor, Locky, Purge, Sage	Xorist
Markov chain [4]	2021	100% on specific types of files	**ransomware tested**: Jigsaw, Eris, Ordinypt, GlobleImposter, Maoloa, Phobos, Nemsis, Deadmin, Seon2, Zeppelin, Paradise, StopDjvuMoka, MegaCortex, NemtyRevenge, Estemani, Sodinokibi, MedusaLocker, Xorist	

Table 2.2: Summary results of our different countermeasures

2.6 Conclusion

Ransomware continues to be a real threat. Today all systems are targeted. More than ever, it is important to follow ANSSI recommendations to keep computers safe.

To detect new ransomware, I have worked on different countermeasures targeting specific behaviour. Some are more efficient than others. These solutions are not perfect and I can imagine undetectable ransomware would avoid all of them.

- First, undetectable ransomware can have a wait a long time before exhibiting its malicious behaviour. In this way, MoM or other test platforms cannot study it, and they class ransomware as non-active.
- The file system traversal is random and slow and not systematic. In practice, encrypting only a few but important documents are pertinent damage and ransomware has less chance to be detected by its activities. New malware as **doxware** and ransomware yet do it this way. A possible countermeasure is to add honey pot files in the system. These honey pots have to be created at different moments, opened periodically, and contains pertinent key words. In other words, a good honey pot has to be undetectable by a performant malware.
- With an undetectable ransomware, the encryption is not detected. First the encryption should not impact the header. Crypto-ransomware gives the entropy of files starts as in [96]. For example, cryptosystem based on Hilbert matrix can be used. In this case all statistical tools to detect pseudo-random written data fail to detect the ransomware sample.
- The message for the ransom can be replaced by a picture. So no specific string appears in the memory snapshot.

In conclusion, there are many works to have a perfect protection against ransomware. The battle is not finished, it is just beginning.

Chapter 3

Blind side channel cryptanalysis

"*I have never tried that before, so I think I should definitely be able to do that.*"
Fifi Brindacier
d'Astrid Lindgren.

Contents

3.1 Introduction

Today, IoT connects everyone and everything. This fact leads to an omnipresence of embedded systems in our life. The deployment of these circuits is a real cybersecurity issue, since one of the possible threats on such embedded systems are physical attacks, when the attacker is capable of grabbing the devices and bring them home. An attacker with access to the circuit can perform physical attacks. Physical attacks rely on the interaction of the computing unit with the physical environment. They are mainly divided in two families: side channel analysis (SCA) and fault injection attack (FIA). This chapter is focused on the first family: SCA.

3.1.1 Motivation

NIST has standardized lightweight cryptographic algorithms that are suitable for use in constrained environments where the performance of previous NIST cryptographic standards is not acceptable. So many new AEAD algorithms have been candidates in this NIST competition [35]. Many attacks already exist on older symmetric cryptography algorithms like AES. Since the AEAD algorithms are relatively new, there are far fewer attacks against them [97–100]. Moreover, these new algorithms are designed to be more resistant to physical attacks than AES. Therefore, one motivation is to test the resistance of these new algorithms against SCA.

Generally, the SCA as in [101, 102] links known data with a measurement. In the case of symmetric cryptographic algorithm, this induces that the attack is often on the first or last round. The framework, described in my thesis [103] and published [32], suggests that other kinds of attack paths are possible. Another approach in the SCA domain is template attacks [104, 105] which compare traces from the targeted device with traces of a profiling device.

In this chapter, the main motivation is to build an attack which uses only traces, no text and no profiling. We are in the case of an attacker who can just observe a leakage but has no access to the device's input/output, hence the name of BSCA. One first idea is to construct an attack path that links one leakage measurement with another one.

One has to remark that retrieving the key without the ciphertext in BSCA may seem absurd in the context of confidentiality. However, in the context of integrity, an attacker could forge a tag for their message after obtaining the secret key.

The dual idea of BSCA in a blind side channel analysis for reverse engineering (BSCARE) context is the ability to recreate a correct output with zero prior knowledge of the algorithm. Only traces, inputs and outputs are used.

3.1.2 Contributions

The main idea of BSCA was initiated during my PhD, under the supervision of my advisor Yanis Linge [106]. Subsequently, the first attack that was developed targeted AES. It represents a chapter of my thesis and has been finally published during my post-doctorate at INRIA [10].

Today I continue to work on this domain. I have obtained funding for a thesis through the APCIL project, thanks to the support of the Brittany region. APCIL focuses on BSCA applied to AEAD. The PhD student working on this project is Modou Sarry, under the direction of Laurent Toutain from OCIF and I co-advise with Gaël Thomas from DGA-MI. Over the past few years, we have hosted two internships on this subject: Awaleh Houssein Meraneh in 2020, and Eïd Maalouf in 2021.

Two new attacks are implemented in simulations [13] on two AEAD: Elephant [37] and Sparkle [43]. An analysis of this algorithm, in this specific context, is shown.

With Ronan Lashermes at INRIA, we have implemented the first blind side channel analysis for reverse engineering (BSCARE). Our works target symmetric encryption and MAC.

3.1.3 Organization

In the first section 3.2, a short state of the art of SCA and side channel analysis for reverse engineering (SCARE) are presented. BSCA against AEAD are developed in section 3.3. BSCARE is presented in section 3.4. Finally, the conclusion is drawn in section 3.5.

3.2 Context and sate of the art for side channel analysis (SCA)

The **side channel analysis (SCA)** are based on observations of the circuit behaviour during the computation. SCA exploits the fact that some physical values of a circuit depend on intermediary values of the computation. This is the so-called leakage of information of the circuit.

Different leakages are possible, different exploitations of these leakages exist. This section summarizes the main different ideas. The domain is very large, as given evidence by the different books [107, 108]. So this section is not a detailed state of the art, it introduces what is needed for the chapter. Different leakages are presented in 3.2.1. Different attack families are summarized in 3.2.2. Although countermeasures are not the primary subject of this chapter, a selection of classical countermeasures is presented in section 3.2.3.

3.2.1 Leakage

Different kinds of leakage can be exploited. Only the most useful leakages are introduced.

3.2.1.1 Timing

The concept of **timing attack** has been introduced by Kocher [109] in 1996. It consists in measuring the execution time of an algorithm. For example, during a conditional branch, two different branches of the code can be called:

- a branch of code that corresponds to the case where the condition is verified;
- a branch of code that corresponds to the case where the condition is not verified.

These two branches of the code may present differences in execution time. An attacker who measures the execution time can deduce whether the condition is verified or not.

3.2.1.2 Power consumption

The idea is to connect directly to the circuit to measure its power consumption [101,102]. An electric current is a movement of electric charges. Each instruction executed by a circuit involves the operation of a certain number of transistors. At any moment, the measurement of the current, known as **power consumption**, reflects the activity of the circuit. Logic gates combine to form a circuit. The power consumption of the device is the sum of the current consumption of each of its logic gates. We have the possibility of distinguishing a transition from 0 to 1 from a transition from 1 to 0. On the other hand, certain operations which are more expensive increase the power consumption of the circuit, in particular because of the use of more components.

3.2.1.3 Electromagnetic (EM) emission

3.2.1.3.1 Principle The **electromagnetic (EM)** emission from a circuit is caused by the flow of the current within it. This circulation of charges in the conductor produces an electromagnetic field composed of a magnetic field and an electric field, mathematically linked by Maxwell's equations. These electromagnetic waves propagate in space and thus can be picked up by a probe. The probe should generally be positioned as close as possible to the active surface of the circuit.

The best advantage of electromagnetic (EM) analyses over power consumption analyses is that the measurements are taken locally. Another advantage is that these attacks are non-invasive. Even better, since the circuit can't easily detect that it's being watched because there is no plugging or even need for physical contact. However, EM measurements are often more sensitive to noise and are very dependent on the positioning of the probe. Power consumption and EM emissions are closely related, so they are often modelled and used very similarly.

3.2.1.3.2 *EMA* [8] When I was in postdoc at LHS of INRIA, I worked on setting up a side channel platform called *EMA*. *EMA* test bench includes the following devices:

- different EM probes from Langer either a `RF-R 0.3-3` or a `RF-R 400-1` both with a `30MHz` to `3GHz` bandwidth;

- a Langer `PA 303` preamplifier (`3GHz` bandwidth) to amplify the signal from the probe;
- a `DSOS404A` oscilloscope from Keysight with a `4GHz` bandwidth able to capture up to 20G samples per second;
- a control computer is used to orchestrate the measurements and perform the analysis with home-made tools.

The FIGURE 3.1 represents a platform to measure the EM emission of a device. This platform used and fully described in [8], is owned by INRIA.

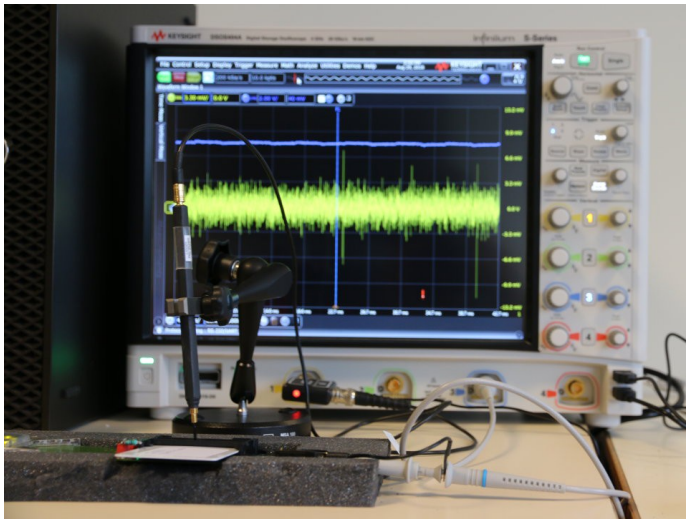


Figure 3.1: *EMA*, side channel platform of LHS INRIA Rennes.

3.2.2 Different kinds of attacks

Even if many different leakages exist, in the following only power consumption and EM are considered. A power consumption or EM leakage is a physical function. As illustrated in the previous paragraphs, to exploit these leakage, the attacks can be split in three families.

- The attacks that extract information directly from the measurement 3.2.2.1.
- The attacks that profile the leakage 3.2.2.2.
- The attacks that use a mathematical model 3.2.2.3.

3.2.2.1 Simple power analysis (SPA)

The **simple power analysis (SPA)** [110–112] is called simple because they determine directly, from an observation of the leakage, during a normal execution of an algorithm, information on the calculation performed or the data manipulated.

3.2.2.2 Profiling attack

3.2.2.2.1 Principle **Profiling attacks** work by learning about the device's behaviour. To implement a profiling attack, a pair of identical devices is needed. One is called the profiling device, the attacker has full control of it; the other is the targeted device. These attacks are in two phases.

1. The first step, called **profiling or learning phase**, is to build a physical model as EM traces for all possible target values with the profiling device.
2. After that, the **attack or inference phase** starts by obtaining traces on the targeted device. These traces are confronted to the different templates.

Template attacks were the first statistical characterizations for side channels. They were introduced as the strongest side channel attacks possible from an information theoretical point of view [104]. They have been used in many attacks, such as [104, 105, 113–117] since. More generally, profiling attacks use machine learning method [118–123].

3.2.2.2.2 An example of template attack against PIN code [9]

I worked to realize a template attack in the case of a personal identification number (PIN) code [9] during my postdoc at Inria. With Jean-Louis Lanet, Ronan Lashermes, Thierno Barry and Damien Couroussé, we have presented the first SCA attack with EM traces of a naïve `Verify PIN` algorithm. It is an application of template attacks to a situation where very few traces are available.

Attack description In this attack, it is assumed that the attacker has the same device as the targeted device, where correct PIN is known. They can obtain all the desired measurements on the profiling device, only a valid candidate PIN is proposed after every other candidate PIN. Namely they can:

- obtain one trace on the targeted device;
- change the correct PIN in the profiling device;
- obtain many traces on the profiling device.

In this way, it is possible to build a template attack on `Verify PIN` algorithms.

The attack is divided in two parts and six steps. The profiling phase is split into:

1. Campaign on the profiling device,
2. Detection of the points of interest,
3. Building templates.

The attack phase is split into:

4. Campaign on the targeted device,
5. Confrontation between measurements,
6. Discrimination of one guess.

Experimentation Once the theoretical framework has been setup, an experimental benchmark has been performed. The `Verify PIN` algorithm is implemented on an ARM-based `STM32-F100RB` microcontroller embedding a `Cortex-M3` core and running in our case at 24MHz. The board used is the `STM32VLdicovery`. This chip does not embed any countermeasures against side channels but it is a popular choice for IoT applications.

The experimental bench is composed of a control computer, a `3405A Picoscope` and an EM probe from Langer (`RF-R0,3-3`) is used to measure the signal. This last one is amplified by a preamplifier from Langer (`PA 303`), before the measurements of the `Picoscope 3405A` (an USB oscilloscope). The bandwidth of the measurement setup is limited by the `Picoscope` with an upper frequency of 100MHz. The traces obtained are composed of 3700 points, with a 1GS/s sampling rate. We remind that 100MHz is a cutoff frequency marking an *attenuation* of the relevant signal, not its total removal.

One has to remark that the required equipment comes at a very low cost, under 2000 euros excluding the computer.

Results We have shown that we can always retrieve a PIN code with 8 traces (9 trials), whatever the number of bytes in the PIN code. For a classical PIN code with 4 bytes in a VERIFY PIN with 3 trials, our attack has a success rate of 48.38% per byte, so finally 5.48% for the whole PIN. For a classical PIN code with 4 bytes in a VERIFY PIN with 4 trials, our attack has a success rate of 60.88% per byte, so finally 13.74% for the whole PIN. These results are not optimal, but the experimental set-up used is very low cost; moreover the biggest template managed had a size of $n = 400000$ traces. We can imagine that an attacker with a better equipment (a high bandwidth oscilloscope and a better computation power to manage more than 400000 traces) can have better results.

An important remark is that the double comparison of the PIN code to protect against fault injection attacks introduces new vulnerabilities for side channel analysis.

By using templates, the attack is made portable. The attacker can perform the profiling phase ahead of time. Then on the target location, the measures and the analysis are fast, as only a few traces are required. Moreover, they can easily perform batch attacks where multiple targets are attacked with the same templates.

A countermeasure against our attack could be to compare the different bytes in a random order. The attacker can retrieve the PIN but not in the right order; so for example, for a PIN code with 4 bytes, with 4 different values, there are $4! = 24$ possibilities. In most cases, this number of possibilities is too big to be authenticated before the device gets blocked.

3.2.2.3 Correlation with a model

3.2.2.3.1 Principle The most commonly used attack is **correlation power analysis (CPA)** [102]. This a family of SCA, which consists in applying a mathematical model to a physical leakage. The first attack of this family was the differential power analysis (DPA) [101].

These attacks have a **divide and conquer** approach. The attacker splits the target into small parts such as bytes or bits for example, which they attack separately. The idea is that for each part, it is easy to enumerate all the guesses.

First, some measurements are acquired during computations that are linked with secrets. On the other hand, a mathematical model and guesses are used to build predictions. The most common models used are the Hamming weight (HW) and the Hamming distance (HD).

The last step consists of comparing the predictions to the traces with a statistical tool called **distinguisher**. This tool brings out a guess. If this guess really corresponds to the real value, the attack is a success. Many distinguishers exist. The choice of the distinguisher depends on the link between the physical function and the model. The mean difference [101], the correlation [102], entropy [124], mutual information [125], the principal component [126], the linear discriminant [127]...

3.2.2.3.2 Hamming weight (HW) model The power consumption or EM leakages are usually correlated to the Hamming weight (HW) of the data. It is a classical model used in the domain of SCA.

One important advantage of using HW is that it rapidly reduces the number of guesses. For example, let B be a byte, so B can take 256 values in $[\![0, 255]\!]$. With the HW of B, the attacker reduces the list of possible values, as shown in Table 3.1.

HW(B)	0	1	2	3	4	5	6	7	8
$\#B$	1	8	28	56	70	56	28	8	1

Table 3.1: Number of possible values for a byte B according its HW.

Another model for the leakage is a HW with an additive Gaussian noise. For a given discrete random variable byte B, and its $\mathrm{HW}(B) \in [\![0, 8]\!]$; $\tilde{\mathrm{HW}}(B)$ denotes the continuous random variable representing the "measured" Hamming weight so called "noisy"; defined in $\mathbb{R}$, such as:

$$\tilde{\mathrm{HW}}(B) = \mathrm{HW}(B) + \sigma_{B,t} \quad ; \tag{3.1}$$

with $\sigma_{B,t}$ an event of the Gaussian random variable $\mathcal{N}\left(0, \sigma^2\right)$ at a time t. The probability density function $\mathbf{F}$ associated to $\mathcal{N}\left(0, \sigma^2\right)$ is given by:

$$\mathbf{F}_\sigma(B) = \frac{1}{\sigma \cdot \sqrt{2\pi}} \cdot \exp\left(-\frac{1}{2} \cdot \left(\frac{B}{\sigma}\right)^2\right) \quad . \tag{3.2}$$

3.2.2.4 Birth of blind side channel analysis (BSCA) [10]

3.2.2.5 BSCA [10]

Blind side channel analysis (BSCA) is a type of SCA where some information that is usually known, is unknown. The main concept is to perform the attack solely based on leakage measurements. Often BSCA is based on a strong assumption: the attacker is supposed to retrieve a noisy HW from the leakage.

Yanis Linge *et.al* have presented the first BSCA in [106]. The goal is to attack a block cipher AES without using the data as plaintext or ciphertext. At the same time, I had the same idea, and finally I published with him and other colleagues a first attack in [10]. Then, our work has been improved by Christophe Clavier *et al.* in [128]. Moreover this contribution introduces for the first time, the name of BSCA. Now it is a new family of SCA [129–131].

BSCA has not yet been proven to be feasible in practice. Most of the research on BSCA has been conducted using simulation traces. However, these attacks can reveal vulnerabilities that are not typically studied in traditional cryptanalysis. So I see this new family of attack as a new tool to analyse symmetric cryptographic primitives.

3.2.2.6 Side channel analysis for reverse engineering (SCARE)

Side channel analysis for reverse engineering (SCARE) is a technique that uses side channels in order to gain information on the inner working of a procedure, mostly a symmetric cipher. It was first introduced by Novak [132] and improved by Clavier [133] where he uses side channels to recover a secret part of the A3/A8 algorithm. With this technique, it is possible to find confidential parts of an algorithm assuming the rest is known such as permutation tables for DES-like [134] or S-Boxes of an AES-like [135]. Previous works have shown that it is possible to apply SCARE to recover a part of a generic cryptographic structure: Feistel [136], SPN [137], stream ciphers [138] have all been shown to be vulnerable to this technique. In [139], Clavier *et al.* demonstrate the feasibility to recover a whole AES-like algorithm with SCARE.

Another direction of SCARE is the side-channel based disassemblers, to recover executed instructions: from the traces they try to find the instructions and data handled by the chip. Eisenbarth *et al.* [140] build a leakage model for all instructions then compare their actual leakage to these models. With this technique, they can recover part of the software of the procedure.

Cristiani *et al.* [141] shows up to 95% recovery of the full 14-bit instruction by exploiting the leakage of all bits individually.

3.2.3 Classical Countermeasures

3.2.3.1 Mask

Masking [142–145] is one of the most common countermeasures against side channel analysis. It is often used to protect critical functions of symmetric cryptographic algorithms, like s-boxes. It consists of applying a random mask to the data which has undesirable leakage. This solution requires adapting the algorithm.

The simple case: a linear function $\ell : \mathbb{F}_{2^n} \Rightarrow \mathbb{F}_{2^\nu}$. Let $x \in \mathbb{F}_{2^n}$ be the intermediate value to protect and $\mathbf{m} \in \mathbb{F}_{2^n}$ random mask value. By linearity of ℓ:

$$\ell(x) = \ell(x \oplus \mathbf{m}) \oplus \ell(\mathbf{m}) \quad .$$

The computation is done in parallel with $x \oplus \mathbf{m}$ and $\mathbf{m}$ and the final value is reconstituted only at the last moment. No information could leak on x because $\ell(x)$ is never evaluated. To obtain information on x, the attacker must take into account 2 information leaks (on $x \oplus \mathbf{m}$ and on $\mathbf{m}$): we then speak of an attack of order 2 [116, 146–148]. More generally, the literature speaks about an scheme of order n by using n random masks [149, 150]. But this scheme remains vulnerable to an attack of order $n + 1$.

For a non-linear function $f : \mathbb{F}_{2^n} \Rightarrow \mathbb{F}_{2^\nu}$ (for example the s-boxes), the difficulty increases. In general, a function $f_{\mathbf{m}}$ is generated such that:

$$f_{\mathbf{m}}(x \oplus \mathbf{m}) = f(x) \oplus \mathbf{m} \quad ;$$

or equivalently by change of variable:

$$f_{\mathbf{m}}(x) = f(x \oplus \mathbf{m}) \oplus \mathbf{m} \quad .$$

In the classic case where the non-linear function is an s-box, in software it is thus necessary:

- either recalculate a complete modified s-box before using it,
- either pre calculate all modified s-boxes (one per mask value) and save them in memory.

It is important to ensure that the implementation never evaluates $f(x)$. In hardware, the masking function could be computed with a dedicated circuit.

Masking is a pertinent countermeasure that continues to be studied recently [151–154]. Some new AEAD algorithms, such as the winner of the NIST competition: Ascon [36], have incorporated masking protection into their mathematical design.

3.2.3.2 Noise and desynchronization

Some countermeasures consist in blurring the signal using smoothing techniques, additive noise or desynchronisation effects. They are designed against statistical attacks.

Adding dummy instructions allows modification of execution time or current consumption a portion of embedded code. This added noise desynchronizes the calculations.

A **desynchronization** [155–157] is possible if the program does not always execute the instructions in the same order. Obviously this modification must not impact the semantics of the algorithm.

3.3 BSCA on AEAD

These works are in the APCIL project funded by the Brittany region. In 2020, with Gaël Thomas from DGA-MI, we have started to work on BSCA against AEAD from the NIST competition [35]. At this moment the competition was not finished. That is why the algorithms studied in this section are Elephant [51] and Sparkle [43]. We started with two intership students, Awaleh Houssein Meraneh and Eïd Malouf. Today, we continue with a PhD student, Modou Sarry. Christophe Clavier and Julien Maillard from Xlim joined us for publications targeting the Elephant algorithm [11, 12].

3.3.1 Theoretical attacks

3.3.1.1 Attack on the LFSR of Elephant [11, 12]

LFSRs are used in different lightweight cryptography candidates, and their initial states often depend on both the key and the nonce. As the nonce needs to be changed for each encryption, attacks on such schemes are limited to the decryption algorithm. In the case of Elephant, presented in prerequisites 1.2.3.4.1, the LFSR only depends on the secret key, Consequently, our attack can be applied in an encryption scenario.

3.3.1.1.1 Goal The goal of the presented attack is to retrieve the Elephant LFSR secret initial state. One has to remark three important points.

- Retrieving the initial state of the LFSR which is equal to $\mathsf{mask}_K^{0,0}$ is equivalent to retrieving the secret key. Indeed, the initial state is just the result of the known permutation P applied to the key.
- As the retroaction polynomial is publicly known, it is possible to shift the LFSR backwards: an attacker who knows 20 consecutive bytes of the secret stream is able to recover the initial state.
- The smaller the LFSR is, the more the attack is able to succeed. As a consequence, the Dumbo 1.10 instance is the most vulnerable.

3.3.1.1.2 Attack path

In the LFSR itself In this attack, it is assumed that the HW of all bytes of the LFSR can be obtained by an attacker.

Since the LFSR generates a single new byte at each iteration, let $B_0, \cdots, B_{19}$ be the content of the Dumbo LFSR initialized with $\mathsf{mask}_K^{0,0}$, and extend the notation for $j \geq 20$, by letting B_j be the new byte generated at iteration $j - 20$. In other words, the attacker has the following relation ($(\varphi 1)$) from the LFSR definition (1.5).

$(\varphi 1)$ $B_{j+20} = (B_j \lll 3) \oplus (B_{j+3} \ll 7) \oplus (B_{j+13} \gg 7)$.

The first idea is to use the knowledge of the following HWs:

- HW (B_{j+20}),

- HW $(B_j) = HW(B_j \lll 3)$,
- HW (B_{j+3})
- HW (B_{j+13}).

So with the two equations ($\varphi 1$) and HWs, the attacker has:

$$\text{HW}(B_{j+20}) = \begin{cases} \text{HW}(B_j) & \text{or} \\ \text{HW}(B_j)+1 & \text{or} \\ \text{HW}(B_j)-1 & \text{or} \\ \text{HW}(B_j)+2 & \text{or} \\ \text{HW}(B_j)-2 & \end{cases} \tag{3.3}$$

Looking more precisely at equation ($\varphi 1$), it can be seen that the difference $HW(B_{j+20}) - HW(B_j)$ depends on only four bits. Letting $B_j[i]$ denotes the i-th least significant bit of byte B_j, these four bits are:

$$\{B_{j+3}[0]; B_{j+13}[7]; B_j[4]; B_j[5]\}.$$

The Table 3.2 gives the value of observed difference $HW(B_{j+20}) - HW(B_j)$ depending on the values of these four bits. In the worst case, there are only 6 possibilities left, out of 16.

HW (B_{j+20}) − HW (B_j)		$(B_{j+3}[0], B_{j+13}[7]) =$			
		$(0,0)$	$(0,1)$	$(1,0)$	$(1,1)$
$(B_j[4], B_j[5]) =$	$(0,0)$	0	+1	+1	+2
	$(1,0)$	0	+1	−1	0
	$(0,1)$	0	−1	+1	0
	$(1,1)$	0	−1	−1	−2

Table 3.2: Values of HW (B_{j+20}) − HW (B_j) according to $\{B_{j+3}[0]; B_{j+13}[7]; B_j[4]; B_j[5]\}$.

Link between the different masks The value $\mathsf{mask}_K^{i,1}$ can be expressed in terms of $\mathsf{mask}_K^{*,0}$.

As in the case of $\mathsf{mask}_K^{i,0}$, let B_j^1 denote either the byte j of $\mathsf{mask}_K^{0,1}$ when $0 \leq j \leq 19$, or the new byte obtained after $j-20$ iterations of the LFSR initialized with $\mathsf{mask}_K^{0,1}$. Likewise, let B_j^1 denote either the byte j of $\mathsf{mask}_K^{0,1}$ when $0 \leq j \leq 19$, or the new byte obtained after $j-20$ iterations of the LFSR initialized with $\mathsf{mask}_K^{0,1}$.

Then, two equations, (3.4) and (3.5), can be written:

$$B_j^1 = B_j \oplus B_{j+1}; \tag{3.4}$$

$$B_j^2 = B_j \oplus B_{j+2}. \tag{3.5}$$

The evolution of the LFSR are analogous to ($\varphi 1$):

$$B_{j+20}^1 = (B_j^1 \lll 3) \oplus (B_{j+3}^1 \ll 7) \oplus (B_j^1 + 13 \gg 7). \tag{3.6}$$

$$B_{j+20}^2 = (B_j^2 \lll 3) \oplus (B_{j+3}^2 \ll 7) \oplus (B_{j+13}^2 \gg 7). \tag{3.7}$$

Merge the information Finally, the attacker can thus exploit two attack vectors: on the one hand, equations ($\varphi 1$), (3.6), and (3.7) coming from iterating the LFSR, and on the other hand, equations (3.4) and (3.5) coming from the different masks used for domain separation.

3.3.1.1.3 Attack strategy

The whole search space corresponding to the initial state of the LFSR is represented as a rooted tree. The nodes of depth j correspond to all possible values for the bytes B_0 to B_j of the LFSR. The tested candidates are denoted by $(B_0^\circ, \cdots, B_{19}^\circ)$. The nodes in the graph of the search space are labelled as follows:

- the nodes at depth j correspond to the all possible values of $(B_0^\circ, \cdots, B_j^\circ)$;
- the children of node $(B_0^\circ, \cdots, B_j^\circ)$, are the nodes labelled: $(B_0^\circ, \cdots, B_j^\circ, B_{j+1}^\circ)$ for all values of B_{j+1}°.

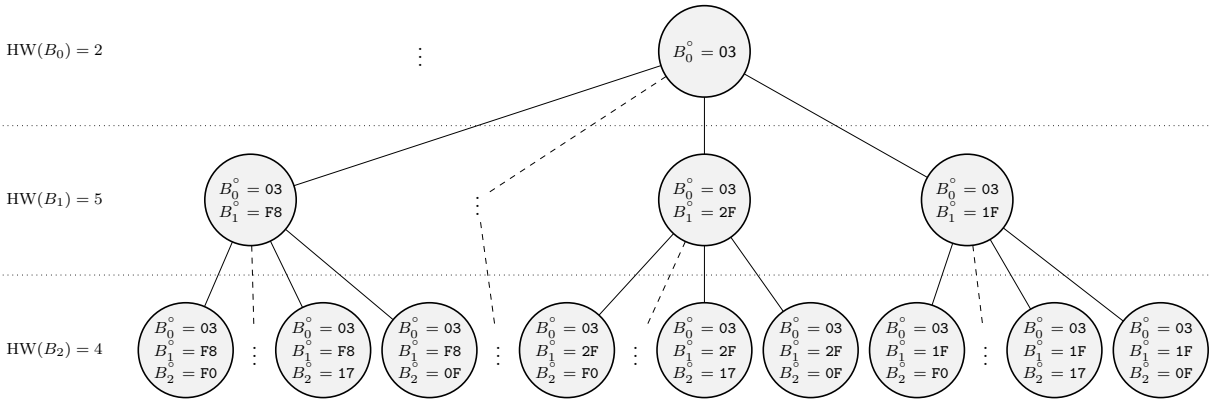


Figure 3.2: Example of the tree representation of the LFSR initial state for the HWs given at the bottom. Only the first three layers of the subtree rooted at $B_0^\circ = 03$ are shown.

In practice, to reduce the number of nodes, only the nodes having the correct HWs are considered. In other words, it suffices to consider nodes with $\mathrm{HW}(B_j^\circ) = \mathrm{HW}(B_j)$. An example of such tree is given on Figure 3.2.

A backtracking algorithm is used. The tree is traversed in a depth-first manner. For each step, the attacker tests whether the current candidate $(B_0^\circ, \cdots, B_j^\circ)$ satisfies the different conditions given by the observed HWs.

One important idea is that the attacker can wait to observe a set of bytes with a minimal complexity of the attack before starting their attack.

3.3.1.1.4 Results

We have simulated the attack on randomly generated Dumbo keys.

The result is that in three quarters of the cases, the key is retrieved in less than two days with desktop computer. The main idea behind this success is the attacker's ability to wait and choose the optimal time to initiate the attack.

On average, for the runs that finished after two days, the number of nodes traversed is $2^{41.92}$, and the number of remaining keys is $2^{36.52}$.

3.3.1.2 Attack on Alzette of Sparkle

Sparkle is another finalist to the NIST lightweight cryptography competition described in 1.2.3.4.2. We work to test the Alzettes $\mathcal{A}_4$ and $\mathcal{A}_5$ of Sparkle against BSCA. This work is not yet published.

3.3.1.2.1 Attack path

K_1 and K_2 from the master key K, are the inputs of $\mathcal{A}_4$ and $\mathcal{A}_5$ respectively.

Therefore, finding the master key K is the same as finding K_1 the input of $\mathcal{A}_4$ and K_2 the input of $\mathcal{A}_5$. As K_1 and K_2 are 64 bits each, so they are divided into two words of 32 bits $(L_0, R_0)_4$ and $(L_0, R_0)_5$ respectively. The method to find $(L_0, R_0)_4$ from the Alzette $\mathcal{A}_4$ is the same to find $(L_0, R_0)_5$ from the Azlette $\mathcal{A}_5$.

It is assumed than the attacker can obtain different leakage measurements on the different internal values L_i and R_i, linked together. More precisely, the measurements are the results of operations in Azlette: xor, rotation and modular addition.

The Alzette is seen as divided in four rounds, and i denotes the round. Finally, the leaking operations in each round of Alzette constitute the attack path. It is illustrated in the Figure 3.3, where the measurement leakage is in red.

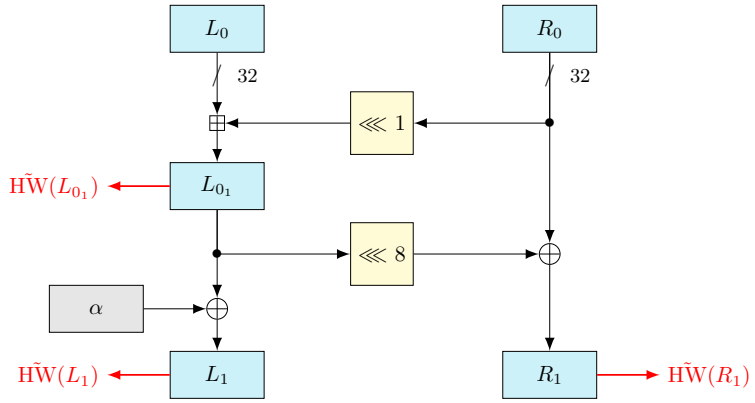


Figure 3.3: Measurement leakage in the first round of Alzette.

The equations (1.20) introduce a strong link between the different points of measurement. That is the chosen attack path that we exploit.

3.3.1.2.2 First results Eïd Maalouf during his work from home intership used the different equation (1.20) to discriminate more guesses on the key. It uses only HW. In his attack, the remaining number of possible 128-bit keys to find is approximately $2^{69.3}$. However, it should be noted that his attack is quite rough and not very strict in practice. So we need to improve the approach.

3.3.2 Attacks in practice [13]

In the real world, it is not possible to obtain HW from the intermediate computation. So the results presented 3.3.1 are not pertinent in practice. In this section, our model for the leakage is a HW with an additive Gaussian noise as explained in 3.2.2.3.2. In this case, another tool is required.

3.3.2.1 Main idea: crossing information with belief propagation (BP)

Information comes in two ways.

- One is the leakage measurement on the different internal values.
- The other is the different equations which link the internal values between them.

An attacker aims to cross the information obtained from the measurements with the information derived from the equations. To this end, a technique known as belief propagation (BP) algorithm (or sum-product algorithm) [158] is used. BP uses an inference Bayesian approach [159].

BP was first used by Gallager [160, 161] for decoding low-density parity-check (LDPC) codes. It was then rediscovered by Tanner [162] and formalized by Pearl [163]. The first time that BP was used in SCA, is the attack [164], then it is studied in [165]. I worked with the same tool in [10] to attack the AES keys.

3.3.2.2 Factor Graph

A BP algorithm relies on a bipartite graph called a factor graph (or Tanner graph). To each node in the factor graph is associated some information.

The nodes of a factor graph are of two kinds:

- variable nodes x;
- factor nodes, representing equations $\mathcal{E}$.

An edge links a variable node with a factor node $\mathcal{E}_j$, when the equation represented by the factor node involves the variable node x_i. A part of a factor graph is illustrated in Figure 3.4.

N denotes a the set of neighbours for a node. Thus the set of $\mathsf{N}(\mathcal{E})$ are composed by variable nodes x; and the set of $\mathsf{N}(x)$ are composed by factor node $\mathcal{E}$.

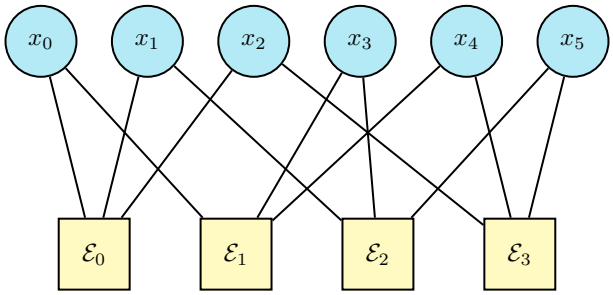


Figure 3.4: Example of a Tanner factor graph part. Circles are variable nodes and squares are factor nodes.

3.3.2.2.1 Use case elephant In the case of Elephant, the Tanner graph illustrated in Figure 3.5 is defined as follows. Variable nodes are the bytes B_j^i of the different LFSRs. There are two types of factor nodes:

- for retroaction equations ($\varphi 1$),(3.6), and (3.7);
- for equations linking the masks (3.4) and (3.5).

We chose not to represent the leakage on the Tanner graph because measurements between bytes are independent.

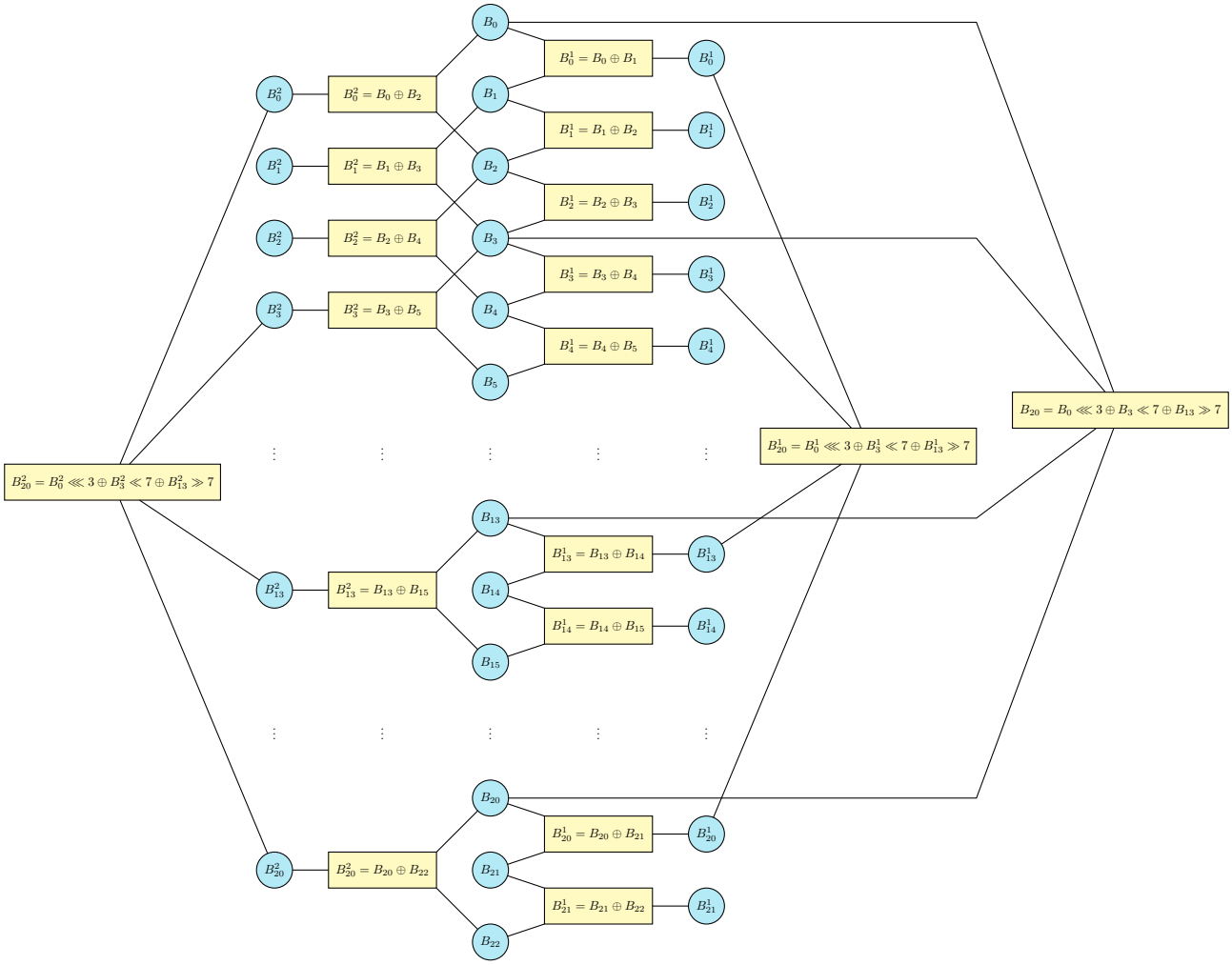


Figure 3.5: Tanner factor graph on LFSRs Dumbo of Elephant

3.3.2.2.2 Use case Sparkle In the case of Sparkle, the Tanner graph is defined on the bits of Alzette blocks (L_0, R_0) to (L_4, R_4). The variable nodes are bits on the different (L_i, R_i) There are two kinds of equations:

- the equations which described the Alzette (1.20),
- the HW equations (sum of bits on a byte).

In this case the HW is represented because it induces a new relation between the different bits. The Figure 3.6 is a part of the Tanner graph for one bit on the first round of Alzette.

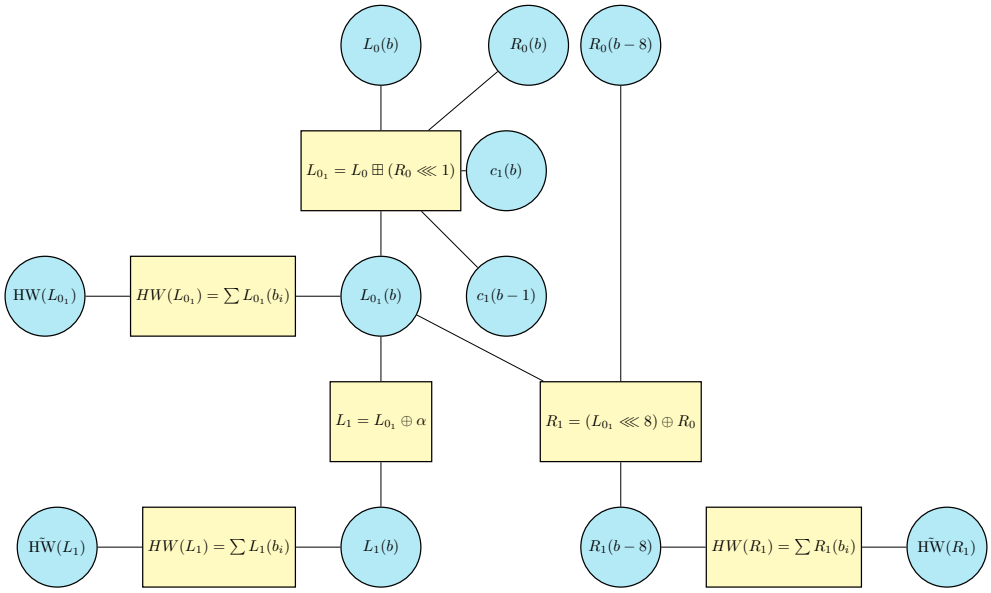


Figure 3.6: Tanner factor graph on first round Alzette, thus only equations (1.8) to (1.11) on one bit b are represented. c represents the carry. Other equation are HW equations on a byte.

3.3.2.3 Generic BP algorithm

BP algorithm inputs are

- the Tanner graph,
- prior probabilities $\mathbb{P}_A$ on the different variable nodes.

BP algorithm returns $\mathbb{P}_P$ a better belief, from the initial value $\mathbb{P}_A$. BP algorithm computes probabilities a posteriori $\mathbb{P}_P$ on the different variable nodes.

$\mathbb{P}_P$ are computed according to the input prior probability $\mathbb{P}_A$ and to the probabilities $\mathbb{P}(x_i = y|\mathcal{E})$ conditional on factor node $\mathcal{E}$ in to be satisfied using the following equation:

$$\mathbb{P}_P \propto \mathbb{P}_A \cdot \sum_{\mathcal{E}_j \in \mathsf{N}(x_i)} \mathbb{P}(x_i = y|\mathcal{E}_j) \tag{3.8}$$

In practice, nodes in the factor graph exchange information messages with their neighbours.
More precisely, since the graph is bipartite, two types of messages are exchanged:

- factor to variable messages between a factor node $\mathcal{E}_j$ and a variable node x_i, denoted $\mu_{\mathcal{E}_j \to x_i}$.

$$\mu_{\mathcal{E}_j \to x_i}(y) = \sum_{y_1 \times y_2 \times \ldots y_{\#(\mathsf{N}(\mathcal{E}_j))-1}} \mathcal{E}_j(y, y_1 \ldots y_{\#(\mathsf{N}(\mathcal{E}_j))-1}) \cdot \prod_{x_l \in \{\mathsf{N}(\mathcal{E}_j) \setminus x_i\}} \mu_{x_l \to \mathcal{E}_j}(y_l) \tag{3.9}$$

- variable to factor messages between a variable node x_i and a factor node $\mathcal{E}_j$, denoted $\mu_{x_i \to \mathcal{E}_j}$;

$$\mu_{x_i \to \mathcal{E}_j}(y) \propto \mathbb{P}_A(x_i = y) \cdot \prod_{\mathcal{E}_l \in \{\mathsf{N}(x_i) \setminus \mathcal{E}_j\}} \mu_{\mathcal{E}_l \to x_i}(y) \tag{3.10}$$

σ	Mean	Standard Deviation	Min	Quartile Q1	Median	Quartile Q3	Max
0.1	3.54	5.97	0	0	3	3	27
0.2	3.67	6.11	0	0	3	3	31
0.3	6.00	10.76	0	0	3	3	97
0.4	9.59	15.72	0	0	3	8	97
0.5	15.97	23.03	0	3	8	31	153
0.6	23.65	31.41	0	3	8	31	157
0.7	33.73	40.11	0	3	27	36	213
1	70.68	61.42	0	31	36	92	246

Table 3.3: Rank on the different bytes of the first LFSR of Elephant for 1000 different keys according different values of noise σ.

The equation (3.9) comes from the law of total probability.

To complete the description of the BP algorithm, an initialization step is done before applying the above equations. The variable to factor messages $\mu_{x_i \to \mathcal{E}_j}(y)$ are initialized with the prior probabilities $\mathbb{P}_A(x_i = y)$. In summary, after an initialization phase, BP works by alternatively applying equations (3.9) then (3.10) for every edge $(x_i, \mathcal{E}_j)$ in the graph. At the end of the execution, the returned value $\mathbb{P}_P$ is computed using equation (3.8). The number of iterations depends on the rapidity to converge.

3.3.2.4 Attack results on simulation

In his thesis, Modou Sarry has developed a generic BP in terms of variable nodes and equations. Thank to this, we have tested BSCA on different AEADs of NIST competition: Elephant [37] and Sparkle [43]. For the moment he has some results in simulation and we project to work on the use case Ascon [36].

A noisy HW is computed, representing the leakage as described in paragraph 3.2.2.3.2. Different values of the noise σ have been tested.

The initial results focus on the LFSR of Elephant. Table 3.3 presents results for the different 20bytes of the first LFSR of Elephant for 100 different keys. This table provides the rank of the correct value.
The second Table 3.4 gives results on the attacks on the Sparkle key of 128 bits.

σ	Mean	Standard deviation	Median	Min	Max	Quartile
0.1	114.06	4.07	114	89	123	(112, 114, 117)
0.2	114.26	3.51	114	97	124	(112, 114, 117)
0.3	113.89	3.67	114	97	125	(112, 114, 116)
0.4	112.92	3.82	113	95	124	(111, 113, 115)
0.5	111.345	3.85	111	92	123	(109, 111, 114)
0.7	108.06	3.85	108	87	120	(106, 108, 110)
1	105.65	4.65	107	85	115	(105, 107, 108)
2	96.33	10.44	100	47	111	(89, 100, 105)
3	88.56	12.54	89	45	111	(82, 89, 100)

Table 3.4: Number of correct Sparkle key bits for 1000 different keys.

In the case where the correct key is not found, the attacker does not know which byte in LFSR Elephant case or bits in Sparkle are incorrect. Therefore, to execute these attacks properly, a backtracking key algorithm can be used. A smart algorithm takes as input the posterior probabilities returned by BP. Doubt is primarily placed on the nodes with the lowest probabilities. One must note that in the case of Sparkle, even with a traditional brute force algorithm, the attacker can retrieve the correct key because they know that only a few bits are wrong on average.

3.3.2.5 Experimental implementation in progress

In his PhD, Modou Sarry has begun testing the attacks with real measurements. The target is a microcontroller `STM32-F411`. He has access to the LHS, including the *EMA* illustrated in Figure 3.1.

The main question to address is: how can we transform an EM trace into exploitable data, such as noisy HW? It is possible that the leakage follows a different model that is more significant. Further investigation is needed to determine this. We hope to obtain experimental evidence to validate the concept of BSCA. Regardless of the specific leakage model, the fundamental idea is that a generic BP can be adapted for analysis.

3.3.3 Impact on the primitive algorithm

3.3.3.1 Elephant use case

The natural question is about what could be done to mitigate this attack path. So these results have been computed in a theoretical attack *i.e* the attacker has HW without noise. Outside of using generic countermeasures, like e.g. boolean masking, there seem to be two possibilities for improvement. Indeed, the attacker gains information from two sources:

- from equations (3.4) and (3.5) used to derive the masks for domain separation;
- from the LFSR state update equation ($\varphi 1$).

Thus either the mask derivation, or the LFSR can be changed, or both.

3.3.3.1.1 Impact of the generation of masks We ran two experiments. In the first experiment, the attacker only know the values of the $HW(B_j)$, and the $HW(B_j^1)$ for enough bytes. In other words, compared to the experiments, they lost the knowledge of the $HW(B_j^2)$. Likewise, in the second experiment, they only know the values of the $HW(B_j)$ for enough bytes. Unfortunately for the attacker, in both cases, none of the 120 runs done has terminated after a week.

From these experiments, it seems that the knowledge of the $HW(B_j)$, $HW(B_j^1)$, and $HW(B_j^2)$ contributed heavily on the success of the attack. It would then seem a good idea to tweak the cryptographic mode of operation by finding another way of generating masks for domain separation.

3.3.3.1.2 Studying different LFSRs This paragraph is dedicated to the study of the influence of the choice of the LFSR. To keep the spirit of the original Elephant, only Fibonacci-like LFSRs, at the byte level, are considered. More specifically, considered LFSRs are: LFSRs where a single new byte is computed from a combination of three bytes using byte-wise shifts and rotations. As usual, the associated feedback polynomial must be primitive to ensure only maximum-length sequences can be generated.

Among all possible candidates, different behaviours can be triggered.

In this work, the **type of an LFSR** is defined as the sequence of number of bits unknown to the attacker at each depth in the tree where a new feedback occurs.

Looking at equation ($\varphi 1$), it can be seen that:

$$|HW(B_{j+20}) - HW(B_j \lll 3)| \leq 2$$

since there are only 2 bits that are modified by:

$$(B_{j+3} \ll 7) \oplus (B_{j+13} \gg 7)\,.$$

Thus, if other feedback functions are used, with more bits involved, it can be expected to have an impact on the attack.

Later in the attack, when at depth 3 in the tree, the same idea can be applied to check whether:

$$|\mathrm{HW}\,(B_{j+17}) - \mathrm{HW}\,((B_{j-3} \lll 3) \oplus (B_j \ll 7))| \leq 1$$

since now only the single bit $B_{j+10} \gg 7$ is unknown. In conclusion, the type of the Dumbo LFSR is $[2, 1]$.

LFSR with different types can be a first criterion when testing our attack.

A second criterion can be: how far apart the feedback bytes are. Indeed, the tighter they are, the faster the attacker can use equation ($\varphi 1$) at its full potential. In the case of Dumbo, the feedback bytes are at indices 0, 3, and 13. We call 13 the **depth**, this is simply the highest index of the feedbacks.

We chose LFSR based on these two criteria. Types is defined from $[2, 1]$ to $[8, 8]$. For types $[2, 1]$, and $[5, -]$, we looked at all the possible LFSRs in order to study the influence of their depth.

The state update function of the different LFSR tested are given by equations ($\varphi 2$) to ($\varphi 21$). Their type and depth are given at the second, respectively third, column of Table 3.5.

(φ2) $B_{j+20} \leftarrow (B_j \lll 3) \oplus (B_{j+1} \ll 7) \oplus (B_{j+11} \gg 7)$

(φ3) $B_{j+20} \leftarrow (B_j \lll 3) \oplus (B_{j+14} \gg 3) \oplus (B_{j+17} \gg 7)$

(φ4) $B_{j+20} \leftarrow (B_j \lll 1) \oplus (B_{j+3} \gg 3) \oplus (B_{j+13} \gg 7)$

(φ5) $B_{j+20} \leftarrow (B_j \lll 1) \oplus (B_{j+9} \gg 3) \oplus (B_{j+15} \gg 7)$

(φ6) $B_{j+20} \leftarrow (B_j \lll 3) \oplus (B_{j+9} \ll 4) \oplus (B_{j+19} \gg 7)$

(φ7) $B_{j+20} \leftarrow (B_j \lll 3) \oplus (B_{j+1} \ll 5) \oplus (B_{j+3} \gg 6)$

(φ8) $B_{j+20} \leftarrow (B_j \lll 1) \oplus (B_{j+4} \gg 3) \oplus (B_{j+19} \gg 5)$

(φ9) $B_{j+20} \leftarrow (B_j \lll 1) \oplus (B_{j+7} \gg 3) \oplus (B_{j+18} \gg 5)$

(φ10) $B_{j+20} \leftarrow (B_j \lll 1) \oplus (B_{j+3} \gg 3) \oplus (B_{j+9} \gg 5)$

(φ11) $B_{j+20} \leftarrow (B_j \lll 3) \oplus (B_{j+1} \gg 7) \oplus (B_{j+17} \ll 4)$

(φ12) $B_{j+20} \leftarrow (B_j \lll 3) \oplus (B_{j+5} \gg 7) \oplus (B_{j+19} \gg 3)$

(φ13) $B_{j+20} \leftarrow (B_j \lll 1) \oplus (B_{j+5} \ll 7) \oplus (B_{j+16} \ll 3)$

(φ14) $B_{j+20} \leftarrow (B_j \lll 1) \oplus (B_{j+1} \gg 7) \oplus (B_{j+9} \gg 3)$

(φ15) $B_{j+20} \leftarrow (B_j \lll 1) \oplus (B_{j+13} \ll 5) \oplus (B_{j+19} \ll 3)$

(φ16) $B_{j+20} \leftarrow (B_j \lll 3) \oplus (B_{j+14} \gg 7) \oplus (B_{j+17} \gg 3)$

(φ17) $B_{j+20} \leftarrow (B_j \lll 3) \oplus (B_{j+4} \ll 1) \oplus (B_{j+5} \gg 6)$

(φ18) $B_{j+20} \leftarrow (B_j \lll 1) \oplus (B_{j+3} \gg 1) \oplus (B_{j+9} \ll 1)$

(φ19) $B_{j+20} \leftarrow (B_j \lll 1) \oplus (B_{j+4} \gg 1) \oplus (B_{j+5} \ll 1)$

(φ20) $B_{j+20} \leftarrow (B_j \lll 3) \oplus (B_{j+1} \gg 1) \oplus (B_{j+8} \lll 7)$

(φ21) $B_{j+20} \leftarrow (B_j \lll 3) \oplus (B_{j+3} \ll 5) \oplus (B_{j+4} \lll 5)$

We ran 120 times the same experiment, for every considered LFSR. For each LFSR, we noted the proportion of runs finished after two days of computations. Results are summarized in Table 3.5.

From these experiments, it seems that the depth has a much more relevant impact than the type. Yet, this seems to be quite tailored to our particular attack. Changing the generation of the different masks is generally more impactful, as it can reduce the amount of information given to the attacker by up to a factor of three.

3.3.3.2 Sparkle use case

The impact of this attack path on Alzette is less pertinent than for Elephant. But a minor result is shown by BP. All the factor node $\mathcal{E}_j$ do not have the same impact to retrieve the secret. Some equations give more information. Indeed, in the following the equation ranked by their order of carried information :

1. the xor with a knowing constant (1.9), (1.12), (1.15) and (1.18).
2. the xor (1.10), (1.13), (1.16) and (1.19).
3. the modular addition with shifting blocks (1.8), (1.11), (1.14) and (1.17).

LFSR	type	depth	finished
(φ1)	[2, 1]	13	53.57%
(φ2)	[2, 1]	11	82.5%
(φ3)	[5, 1]	17	0.83%
(φ4)	[5, 1]	13	94.17%
(φ5)	[5, 1]	15	28.33%
(φ6)	[5, 1]	19	11.67%
(φ7)	[5, 2]	3	100.0%
(φ8)	[5, 3]	19	0.83%
(φ9)	[5, 3]	18	0.0%
(φ10)	[5, 3]	9	95.83%
(φ11)	[5, 4]	17	0.83%
(φ12)	[5, 5]	19	0.0%
(φ13)	[5, 5]	16	0.0%
(φ14)	[5, 5]	9	82.5%
(φ15)	[5, 5]	19	0.0%
(φ16)	[5, 5]	17	0.0%
(φ17)	[8, 2]	5	100.0%
(φ18)	[8, 7]	9	78.75%
(φ19)	[8, 7]	5	100.0%
(φ20)	[8, 8]	8	79.17%
(φ21)	[8, 8]	4	100.0%

Table 3.5: Type, depth, proportion of runs finished after two days of computations, for Dumbo (φ1), and LFSRs (φ2) to (φ21).

3.4 Blind side channel analysis for reverse engineering (BSCARE) [14]

This section presents work with the collaboration of Ronan Lashermes from INRIA. We hope to publish the final results very soon [14].

This work is a new reverse engineering technique that can reconstruct a procedure from inputs, outputs and traces of execution only. The algorithm and the machine is totally unknown. A complete reverse engineering is proposed. So it is another version of BSCA called blind side channel analysis for reverse engineering (BSCARE).

This work intends to explore some of the limits of SCA: what information can be recovered from traces ? Under assumptions, we show that an unknown algorithm can be totally reverse-engineered even executed on an unknown machine, from noisy and lossy traces of execution as the power consumption, for example.

As a reminder of the importance of Kerchoff's principles, this work gives yet another argument that security by obscurity can be bypassed.

The Table 3.6 illustrates the positioning of our work in the existing literature. To our knowledge, there are no similar works.

target \ knowledge	algorithm family	instruction set architecture (ISA)	is a RISC processor
Instructions	×	[140, 166, 167]	×
One or several specific function(s)	[132–135]		×
Generic structure	[136–138]		×
Generic structure + specific function	[139, 168]		×
whole algorithm	×	×	our BSCARE

Table 3.6: Classification of SCARE techniques according to the target of the attacker and its knowledge.

To experiment our attack we have chosen two different block ciphers: AES [46] described in section 1.2.3.1.1 and Twine [47] described in section 1.2.3.1.2.

3.4.1 A simplified description of the attack

The BSCARE technique only relies on the leakage of the data in the targeted algorithm. In this report only a summarised simplification of the easy case without noise is described.

For this work we have the following assumption.

- **8-bit reduced instruction set computer (RISC) software procedure**
  The attacker can only deal with 8-bit RISC software implementations. It means that only 1-to-1 or 2-to-1 elementary operations are allowed with one input-one output or two inputs-one output, operating on 8-bit data.
- **Temporal causality**
  In one trace, the information is present in chronological order. A byte value at time t_0 can be determined by byte values at times $t < t_0$.
- **Synchronicity**
  To perform the analysis, the procedure under test must be repeatedly executed. Traces are required to be synchronous: at the same time (index in the trace), the byte value corresponds to the same underlying procedure sub-operation.

3.4.1.1 Collected data

The objective is to build a function $\mathcal{G}$ that actually computes the same output as the targeted unknown procedure $\mathcal{P}$ for any given input. To do that the attacker uses traces: she repeatedly registers trace and output data for several executions of the same procedure $\mathcal{P}$ for different input data. In the data generating phase, the procedure is executed n times, with inputs of size S_I bytes, outputs of size S_O bytes, and traces length of S_T bytes. All measured data are sampled, giving only values as unsigned bytes: $[\![0, 255]\!]$.
These experiments give three matrices:

- I is the input matrix.
- O is the output matrix.
- $\mathcal{M}$ is the trace matrix.

One row of a matrix corresponds to one execution. For the trace matrix, one column corresponds to all values sampled at the same time t, for all executions. In the rest, it is supposed that inputs and outputs are present in the traces directly, i.e. there is a 1-to-1 function linking an input byte to one trace time *etc.* A trick to make that assumption hold in all cases is to rewrite $\mathcal{M}$ as the concatenation of matrices I, $\mathcal{M}$ and O.

Since the traces are synchronous, then $\exists\ \mathbf{t}_1, \mathbf{t}_2, \mathbf{t}_3, f$ such that $\forall l, \mathcal{M}_{l,\mathbf{t}_3} = f\left(\mathcal{M}_{l,\mathbf{t}_1}, \mathcal{M}_{l,\mathbf{t}_2}\right)$; there is a function f that links times $\mathbf{t}_1$ and $\mathbf{t}_2$ to $\mathbf{t}_3$ as illustrated on Figure 3.7.

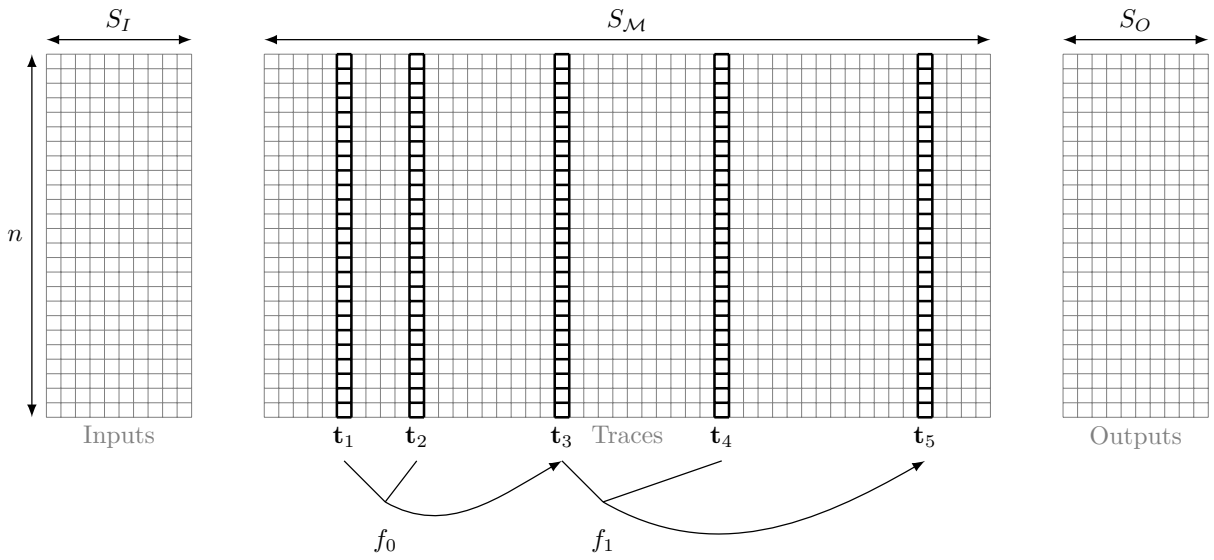


Figure 3.7: Illustrative recapitulation of our notation. Each square represents a byte. The procedure is executed n times. Inputs, trace of execution and outputs are recorded and correspond to a row in the matrices. A time is the index of the column in the traces matrix $\mathcal{M}$. In the illustration, we have identified relations f_0 and f_1 allowing deducing a column from two others. $\mathbf{t}_3 = f_0(\mathbf{t}_1, \mathbf{t}_2)$ and $\mathbf{t}_5 = f_1(\mathbf{t}_3, \mathbf{t}_4)$.

3.4.1.2 Using fractional added information (fai) to identify functions

At numerous points, the attacker needs a measure to detect 1-to-1 and 2-to-1 functions by measuring how much information is in the output, knowing the inputs. The measure that is devised is called **fractional added information (fai)**, it is the estimated normalized conditional entropy.

$$fai(\mathbf{t}_i|\mathbf{t}_j) = \frac{H(\mathbf{t}_i|\mathbf{t}_j)}{H(\mathbf{t}_i)}. \tag{3.11}$$

where H is the Shannon entropy, $\mathbf{t}_i$ the output and $\mathbf{t}_j$ the input of the 1-to-1 or 2-to-1 function ($\mathbf{t}_j$ can be a vector or a two-column matrix).

A fai measure of 1 means that $\mathbf{t}_i$ is independent from $\mathbf{t}_j$. If fai is 0, then $\mathbf{t}_i$ is totally determined by the knowledge of $\mathbf{t}_j$.

3.4.1.2.1 Identifying 1-to-1 functions

To identify a 1-to-1 function, the $fai(\mathbf{t}_2|\mathbf{t}_1)$ measure for all pairs of time $(\mathbf{t}_1, \mathbf{t}_2)$ where $0 \leq \mathbf{t}_1 < \mathbf{t}_2 < S_{\mathcal{M}}$, is computed. If $fai(\mathbf{t}_2|\mathbf{t}_1) = 0$, then $\exists f$ such as $\mathbf{t}_2 = f(\mathbf{t}_1)$.

3.4.1.2.2 Identifying 2-to-1 functions

To identify 2-to-1 functions, the attacker examines fai measures for each $(\mathbf{t}_1, \mathbf{t}_2, \mathbf{t}_3)$ triplets with $0 \leq \mathbf{t}_1 < \mathbf{t}_2 < \mathbf{t}_3 < S_{\mathcal{M}}$. If $fai(\mathbf{t}_3|\mathbf{t}_1, \mathbf{t}_2) = 0$, then $\exists\ f$ such that $\mathbf{t}_3 = f(\mathbf{t}_1, \mathbf{t}_2)$.

3.4.1.3 Generate an information flow hypergraph (IFH)

At this stage the attacker has a complete raw **information flow hypergraph (IFH)** that should link inputs to outputs. The raw to logic transformation consists in the rewriting of the hypergraph in a simpler one called the **logic IFH**, while applying corresponding transformations on the raw traces to obtain the logic traces.

This hypergraph contains only hyperedges with one destination node, and one or two source nodes, for 1-to-1 and 2-to-1 functions respectively. The Figure 3.8 illustrated an example of IFH part.

The attacker transforms the $\mathcal{M}$ matrix into **logic traces**, a new matrix representing the "ideal" traces that can be extracted from the procedure. From these logic traces, they determine the actual functions linking values between them in the same execution.

The last step is a simplify step. Where the attacker merge 1-to-1 functions in the just one as presented in Figure 3.9.

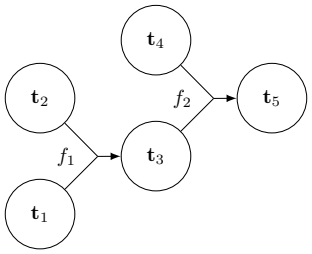


Figure 3.8: Illustration of the information flow hypergraph corresponding to Figure 3.7

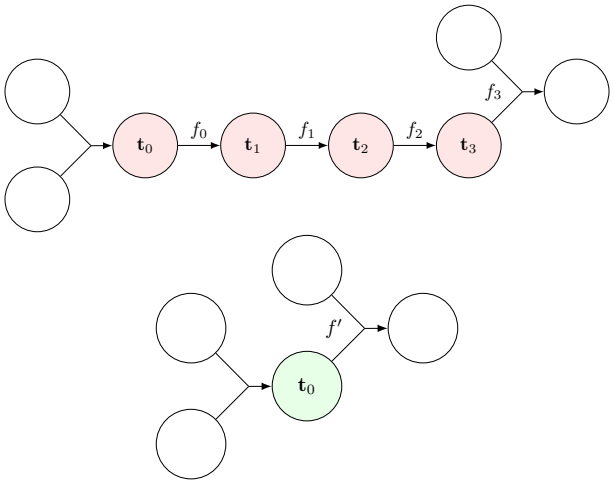


Figure 3.9: Example of simplication of 1-to-1 functions: f_0, f_1, f_2, f_3 in just one: f'.

3.4.1.4 Reconstitution of the function

To finish, functions are needed to link procedure inputs and outputs to their corresponding appearance time in the logic traces. This is simply done, for each input/output byte, by finding then extracting the corresponding 1-to-1 function.

At this point the function $\mathcal{G}$ is built. To execute it, the attacker has to generate a logic trace corresponding to the provided fresh inputs.

3.4.2 Use cases

Even if the technique is not specifically designed to target symmetric cryptographic primitives, it is particularly interesting to use in this case for an attacker. We have chosen to opt to validate the techniques by applying it to reverse engineer simulations of AES [46] and Twine [47], respectively described in 1.2.3.1.1 and 1.2.3.1.2. This paragraph enumerates potential malicious applications that an attacker could execute with this capacity in symmetric cryptography context.

- **Forging encrypted data.**
  Forging encrypted data can allow the adversary to feed their own information to the system. A man-in-the-middle attack could be mounted using BSCARE by observing the sender's encryption process, and then replacing the encrypted data by the forged ones.

- **Decrypting data with specific modes of operation.**
  Many modes of operation most notably counter based such as CTR, use the same block cipher for encryption and decryption. Thus, depending on the precise mode, it becomes possible to use the block cipher learnt with BSCARE to decrypt other data that have been encrypted with the same cipher and key. In this regard, BSCARE it is particularly powerful against these modes.

- **Forging message authentication codes**.
  It may be possible to clone a MAC function using BSCARE in some circumstances. For example, in the case of the Pelican MAC function [49] described in section 1.2.3.2, there are several possible ways to reverse engineer this function. Either the attacker is able to observe the execution of the Pelican MAC,

with the same secret key, numerous times for messages of the same size (respecting the constant-time constraint). Or they can try to reverse engineer subparts of the algorithm if they can observe some intermediate values. For example, since the key is used only at the start and the end of the procedure, if the attacker can observe the inputs and outputs of the last AES application it is enough to reverse engineer the subparts using the secret key. This particular example illustrates that BSCARE can also be used to reverse engineer only unknown parts of otherwise known schemes.

- **Getting the decryption from the observation of encryptions.**
  Similarly to CTR modes being particularly vulnerable, some block cipher structures may increase the power of this technique. In particular, GFN like Twine could be inferred by observing the encryption procedure. But preliminary experiments showed that it implies to modify our structure recovery method to properly deal with reordering. In particular, it is not possible to just switch inputs and outputs and flip the traces. The resulting structure still breaks our *temporal causality* assumption.

3.5 Conclusion

In this chapter, two physical cryptanalysis techniques targeting AEAD have been presented, specifically against Elephant and Sparkle. These attacks, known as blind side channel analysis (BSCA), employ a belief propagation (BP) algorithm, a powerful tool in this context.

Moreover, BSCA is emerging as a compelling tool to evaluate symmetric cryptographic primitives. In the case of Elephant, it highlights that the choice of incrementation in different rounds presents a vulnerability. The results highlight that the selection of LFSRs is far from trivial. Potentially, this approach could be extended to other security algorithms.

Ascon emerged as the winner of the NIST competition, so a future direction of my work involves assessing the resilience of Ascon against BSCA.

Additionally, this chapter introduced the concept of BSCARE. Most work related to SCARE possesses some level of information on the targeted algorithm. The results of BSCARE are significant as they indicate an attacker could potentially use traces from a completely unknown algorithm.

In these attacks, the assumption that an attacker can obtain a HW or noisy HW is substantial. Future experimentation will determine whether we can validate this hypothesis.

To conclude, I believe we are only at the beginning of these novel approaches, namely BSCA and BSCARE.

Chapter 4

Protecting instructions: toward a cryptographic solution.

"En essayant continuellement on finit par réussir. Donc : plus ça rate, plus on a de chance que ça marche."
Les shadocks
de Jacques Rouxel.

Contents

4.1 Introduction

Physical attacks are attacks that rely on the interaction of the computing unit with its physical environment. Side channel analysis (SCA), presented in the previous chapter 3 are based on the observation of the circuit's behaviour during the computation: e.g. the time taken by a computation or the device's power consumption. Fault injection attack (FIA) consists in disturbing the computation in order to alter the correct progress of the application. Clock glitches, power glitches, electromagnetic pulses or laser pulses are commonly used to obtain these perturbations.

Embedded systems are targets of choice for physical attacks since the attacker can get physical access. Unfortunately, most devices are not protected against physical attacks. There are multiple reasons for this: economics since hardened devices are more expensive, technical because the security is achieved at the cost of both performances and energy efficiency. The last barrier is cultural since for a long time such attackers were dismissed as not realistic by all but one industry: the smart card designers. In practice, there is a real need to protect devices against physical attacks.

To protect against the fault injections attacks, the execution of a program must enforce guarantees. We define the following guarantees.

- **Control flow integrity (CFI)**: the control flow cannot be modified (no arbitrary jumps). The control flow follows the valid control flow graph (CFG). One has to remark that while it's possible to prove that a path is one of the possible path in the CFG., until now, it is not possible to definitively prove that it's the correct one.
- **Instructions integrity (II)**: the instruction values must not be altered outside of it.
- **Data integrity (DI)**: the data handled by the program must not be altered.
- **State integrity (StI)**: the processor state (configuration, registers, program counter, ...) must not be altered.

Often, DI, II and StI are considered together under the name of system integrity (SI).

4.1.1 Motivation

Ensuring resistance to physical attacks is crucial, and integrity is a prerequisite for many critical infrastructures. A large literature exists on the protection of instructions and the works target predominantly the integrity, rarely confidentiality. One has to remark the major propositions [169, 170] protect the control flow against error or code injection attack (CIA) and code-reuse attack (CRA), but not against a hardware attack like fault injection attack (FIA).

In the literature, a countermeasure is developed for each attack on a specific algorithm or device. However, the idea of creating a patch for each new attack is insufficient. There is a real motivation to find general, rather than specific, countermeasures.

The main countermeasures in the literature to resist physical attacks aim at protecting specific computations for some critical algorithms [171]. For examples, masks are used to protect cryptographic primitives (e.g. AES S-box) or redundancy is used to protect the critical verification (e.g. verify `PIN`).

But resistance to physical attacks should be ensured in all contexts. For example, a fault in a non-critical part of an application can be enough to allow the attacker to take control of the device, via a return-oriented programming (ROP) attack. In consequence, recently, a new trend appeared where the designers protect all instructions of an application and not only a specific algorithm.

This chapter is focused on instructions protection in the context of physical attacks, with specific emphasis on fault injection attacks. Hardware fault injection can modify a program at runtime by skipping arbitrary instructions [172] or by altering the instruction encoding. Consequently the problem of integrity in the microarchitecture is a rising concern today. When a strong integrity guarantee is needed from a processor, one of the leading solution is to use lockstep processors. Such a processor contains two or more identical cores that reproduce the same computation on each core. Then a monitor ensures that they provide identical results. Lockstep processors are difficult to implement, are not energy efficient and often provide poor performances since we need simple cores in order to synchronize them in the system. Additionally, they do not resist an attacker able to inject identical fault on all the cores with the correct timing. A feat hardly feasible today, but with advancing injection techniques this does not seem too far-fetched in a near future.

4.1.2 Contributions

I have worked on the project `Cogito` [173], which presents a significant improvement to the community by providing automatic countermeasures against various attacks. `Cogito` implements classical countermeasures through polymorphic code.

After this professional experience, I believe that stronger countermeasures are necessary. One idea to protect instruction integrity against faults was published in HAPEI [16]. While this work has some limitations and practical challenges, it can be considered a starting point. I strongly believe that finding effective and practical cryptographic solutions to defend instructions against physical attacks is a new area of research.

4.1.3 Organization

The context of fault injection attacks is provided in section 4.2. Results of the `Cogito` solution are presented in section 4.3. The solution HAPEI is presented in section 4.4. Then, section 4.5 discusses new ideas that I want to explore. A conclusion is drawn in section 4.6.

4.2 Fault injection attack context

4.2.1 Definition

The concept of injecting physical faults originates from the field of space exploration, where cosmic rays generate errors in the circuits of computers sent into space. A similar problem arises in the context of nuclear technology. As a result, the first physical faults were accidental and not intentional or malicious.

In this manuscript, **fault injection attack (FIA)** is the idea of disrupting the normal operation of a device, with the aim of obtaining information or hijacking it [174, 175].

4.2.2 Technical injection

4.2.2.1 Laser fault injection

Light radiation emitted by a laser or a focused light source can be used to inject faults into a device. Such fault injection devices require the package to be open, these are therefore invasive attacks. This method is a subdomain of fault injection techniques [176–181].
This method uses the photoelectric effect, which is the emission of electrons when electromagnetic radiation, such as light, hits a material. In practice, the energy of the light radiation used is absorbed by the silicon of the circuit, which induces a voltage spike. The induced voltage spike can cause the appearance of a fault if it is sampled by a memory element such as a register.

4.2.2.2 Electromagnetic fault injection

Another method is the injection of faults by electromagnetic pulse [182–184]. The circuit attacked is placed under a near-field antenna. The attacker then causes a very strong current to flow for a very short time (the exact parameters depend on the targeted chip). By electromagnetic coupling, part of this energy is transmitted in the metal at the surface of the chip (power supply tracks, clock tracks, data bus, etc.). The current generated in the chip can therefore generate errors.

Figure 4.1: *Faustine*: fault injection bench of the LHS INRIA Rennes. ©INRIA / Photo C. Morel

Figure 4.1 depicts a platform for injecting faults into a device, using electromagnetic radiation. This platform is owned by INRIA Rennes [185]. As I have access to this platform, I can provide additional details about it.

Faustine is made of a Keysight `33509B` pulse generator, a Keysight `81160A` signal generator and a Milmega `80RF1000-175` power amplifier, connected in sequence to generate a signal. This signal then passes through a Langer RF probe `RF B 0.3-3` located on the targeted chip to generate an electromagnetic fault injection. In order to launch a fault injection, a synchronization signal (a trigger) is sent by the targeted chip General-Purpose I/O (controlled from the code) directly to the `33509B` pulse generator. This experimental trick, possible when the attacker has control of the code (i.e. only for vulnerability assessment) is not mandatory. Other synchronization possibilities include sniffing communications with the target or measuring its EM to find a relevant pattern. The location of the probe on the chip is chosen after a scan that determined the most sensitive area on the targeted chip.

4.2.2.3 Clock glitch

A synchronous circuit is composed of one or more logic gates, framed by registers managed by a common internal clock, denoted `clk`. The logic blocks correspond to the functions computed by the theoretical algorithm. The registers are memory elements of the device. The internal clock gives at regular intervals, an electrical pulse, simultaneously to all processor components. At each rising edge of the clock, data are updated at the output of the registers. The clock period must be long enough to ensure that the calculations run as shown in Figure 4.2.

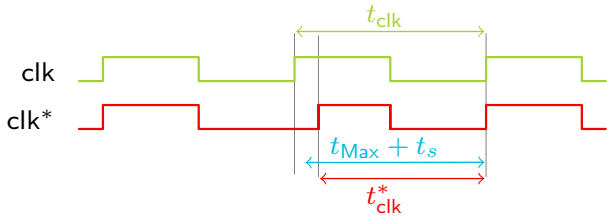


Figure 4.2: Clock glitch

More precisely, the following notations are used.

- t_{clk}: the clock period.
- t_{Max}: the propagation time of the bit that has the longest path to go from one register to another.
- t_{s}: the setup time during which the data must remain stable at the input of a register to be sampled correctly.

The following equation must always hold:

$$t_{\text{clk}} > t_{\text{Max}} + t_{\text{s}}.$$

In the case of a glitch on the clock, if:

$$t^*_{\text{clk}} < t_{\text{Max}} + t_{\text{s}},$$

an error can appear. This technique can be used to generate errors/faults to build attacks [186–188].

4.2.3 Effects

The consequences of a fault injection are multiple. In case of failure, the device can be destroyed. It is the undesirable effect of a bad adjustment parameter for a laser or a electromagnetic pulse. A successful fault disrupts the computation.

As in SCA the process to obtain a fault is a physical experiment thus not a mathematical function. But a mathematical model can be used. The choice of the model depends the chosen different abstraction levels. The different abstraction levels 1 to 5 are presented in chapter 1. In this section, main models at different levels are presented.

4.2.3.1 Algorithm and software model

The fault can be modelled directly at the algorithm level (1). A classic model in this case is the xor function:

$$x^* = x \oplus e \tag{4.1}$$

The faulty value x^* is just the correct value x xored with an error value e.

A classical exploitation example in physical cryptanalysis is the **differential fault analysis (DFA)**, which are attacks that exploit faulty results obtained through fault injection. Cryptographic primitives are prime targets for these attacks, as demonstrated by the extensive literature [124, 175, 189–193].

At the software level (2), the model is very similar to the algorithm model (1). The model can just be more specific to the language.

4.2.3.2 Microarchitecture model

This section examines the fault from the processor's perspective: what are the effects of the injection on the operation of the processor and therefore on the software running on it. One has to remark that most of the work characterizing fault models on the targeted microarchitecture focus on microcontrollers. Thus, the details of the effects of a fault injection on a complex architecture are largely unknown. Here is a non-exhaustive list of classic model used:

- **Corruption of the cache memory** [194].
- **Instruction skip** or no operation (NOP) [195–198]. This fault model has a direct exploitable impact at algorithm level. For example, the verification of a `PIN` code can be skipped as in [188].
- **Phantom edges** refer to instances of branching or jumping to uncontrolled destinations. The term derives from the effect of the injection adding an edge, which is normally absent, to the CFG.
- The **replay** is an instruction that is overwritten and replaced by the previous instruction [199, 200].

4.2.3.3 RTL model

The effect from the device's point of view, at the RTL level (4), corresponds to its impact on the logical values of the signals.

- A **bit flip** is the inversion of a bit value on a wire. Depending on the wire's location in the circuit, this error may be propagated to a register, leading to a fault, *i.e.*, an incorrect result. However, it's also possible that the error is logically absorbed by a combinator circuit: no fault then manifests.
- A **stuck at 0 or stuck at 1** fault is the consequence of an induced current having a memory effect due to feedback: a latch-up. The wire then no longer moves for a certain time, remaining at the value 0 or 1. This time may be reduced or infinite: it is then necessary to stop the circuit's power supply to erase the disturbance.

4.2.3.4 Physical model

The physical models are at the transistor level (5). A **timing fault model** [182] is about a fault generated in a register due to the violation of temporal constraints.

It is also possible to directly influence the sampling process of a register, or by directly disrupting the clock. This is the **sampling fault** model [201]. This is a model that corresponds, in principle, more to EM injection.

In reality, a fault injection by disturbance or EM is likely to cause one of the two models, or both at the same time, depending on the bench and the target.

4.2.4 Countermeasures

Countermeasures such as noise and desynchronisation, which are used against SCA and introduced in section 3.2.3.2, can also be employed against FIA. Indeed, it becomes more challenging for an attacker to target a specific instruction if that instruction doesn't consistently occur at the same moment during execution.

4.2.4.1 Redundancy

In the case of fault injection attacks, most countermeasures mainly aim to add **redundancy** to the code to be executed [202–204]. This redundancy can be implemented at different levels. It can be as follows.

- **Spatial redundancy**: several distinct part or element perform the same operation in parallel. For example, at the microarchitecture level (3): two cores are placed together, expected to execute the same tasks synchronously. This is called processors in lockstep.
- **Temporal redundancy**: the same operation is repeated several times; or the inverse of the operation is computed. Sometimes another possibility is to compute the reciprocal of the computation and check the result with the initial input.

One has to remark that today an attacker is able to repeat the same fault several times during the same execution [188, 205]. So just adding redundancy is not a pertinent countermeasure.

Another form of redundancy is the use of **error detecting codes** or **better error correcting codes** (to protect against denial of service) [206–208]. In this case, sometimes, it is a question of modifying the algorithm to integrate these codes. It is for example possible to modify the sub-functions of the AES to calculate the modification of the state in parallel with a parity bit on this state.

4.2.4.2 Shield

To protect a device at the transistor level (5), it is possible to use a **shield**. It consists of metal tracks positioned above and below the chip [209, 210] These tracks provide passive electromagnetic shielding for the tracks below. Additionally, to prevent attackers from penetrating the shield (e.g., using a focused ion beam), pseudo-random data can be transmitted through the tracks of the shield. The integrity of the shield is ensured by verifying that the data is still present and correct.

4.2.4.3 Instruction set randomization (ISR): a symmetric cryptography solution

In software and control flow integrity architecture (SOFIA) [211], the authors want to ensure CFI with instruction set randomization (ISR). It is one of the first solutions which uses cryptography to protect the instruction at the microarchitectural level.

The proposed solution is to mask the instructions with a key stream depending on the program counter (PC) and the previous PC denoted PC_{prev} (for the previous instruction). Let $\mathcal{I}$ be an instruction and $\mathcal{I}'$ the corresponding encrypted instruction.

$$\mathcal{I}' = En(PC_{prev}||PC, K) \oplus \mathcal{I}$$

This elegant solution ensures that an instruction can be correctly decrypted only if PC and PC_{prev} are correct. Effectively, it encrypts all the possible successions of instructions and the correct instruction can be decrypted only upon correct PC and PC_{prev} values.

In order to ensure instructions integrity (II), a tag with a MAC is computed and verified per batch of up to 6 instructions. The tag value is stored as two words at the beginning of each block. If an instruction has two

predecessors, a special case must be made: the multiplexor block. In this block, the two first words correspond to the MAC values for the two possible predecessors:

$$\begin{cases} T_1 = & \text{MAC}(PC^1_{prev}||PC, K), \\ T_2 = & \text{MAC}(PC^2_{prev}||PC, K). \end{cases}$$

The tag not used (corresponding to the wrong predecessor) must be skipped in a software transparent way.

In a recent publication [170], the authors show how encryption can be achieved at the instruction level using a stream cipher. The decryption is so fast and lightweight that it can be performed very deep within the processor, ensuring that plain instructions remain only within the processor's execution logic. The method requires slight hardware and software modifications.

4.3 Instructions randomization by `Cogito` [15]

`Cogito` is an ANR project [17] that funded my post-doctorate. The coordinator was Damien Couroussé from CEA. The partners were INRIA, ENSMSE and CEA. My use case was to test the countermeasures of `Cogito`, protecting `PIN` codes against SCA. Different works have been published [15, 173, 212, 213].

It is important to clarify that in `Cogito`, it is indeed the randomization of the instructions and not the randomization of the instruction set, so it is a software solution.

4.3.1 The project

Security in embedded devices and runtime code generation are, a priori, two technological fields that hardly combine together. On the one hand, secure elements must target small production costs, silicon and energy consumption, and as such, offer very limited computing and memory resources. On the other hand, compilation is a computation-intensive process, and dynamic compilation techniques require a fair amount of computing power and of memory resources at runtime. In `Cogito`, the authors focused on the use of runtime code generation to introduce behavioural variability in embedded systems. Indeed, behavioural variability is often used as a protection against physical attacks. So, the objective of `Cogito` is to demonstrate the applicability and the effectiveness of code generation techniques applied at runtime and on board for security purposes in embedded devices.

The `Cogito` solution is an adaption of `deGoal`, a technology for runtime code generation, developed by the CEA [214]. In `deGoal` code segments called kernels are generated and tuned at runtime by ad hoc runtime code generators, called compilettes. A compilette is an ad hoc code generator focusing on a single kernel, with the intention of being invoked at runtime.

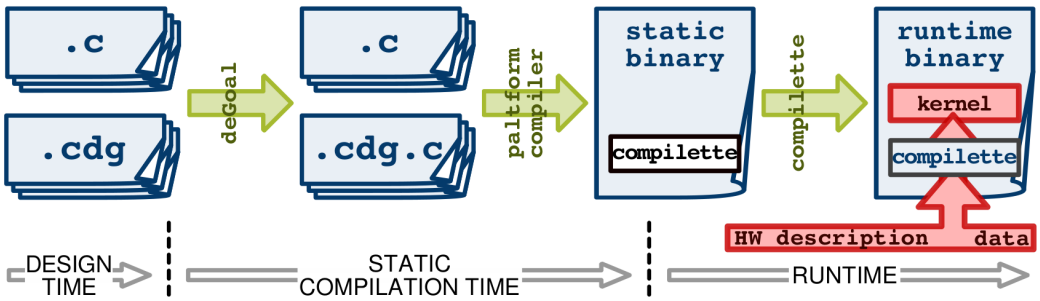


Figure 4.3: `deGoal` workflow: from the writing of the application's source code to the execution of a kernel generated at runtime. Figure is extracted from [173] HW means hardware.

The building and the execution of an application using `deGoal`, illustrated in Figure 4.3, consist in the following steps :

- writing the source code using a mix of C source code and of a dedicated `cdg` language;
- compiling the binary code of the application and the binary code of compilettes;

- at runtime, generating the binary code of kernels by compilettes and in the end running the kernels.

At the end of the project, polymorphism is implemented with runtime code generation driven by random data. The main result is to regularly generate new versions of the secured binary code on the target. Each program version is functionally equivalent but has a different implementation, so that each execution would lead to a different observation. The polymorphic code generator is produced by a framework for runtime code generation adapted to the constraints of embedded systems.

The project was carried out by partners including INRIA, ENSMSE, and CEA, each with different responsibilities. As part of my post-doctoral work, INRIA was responsible for testing the countermeasure `Cogito` in a use case involving `Verify PIN`.

- The CEA was responsible for proposing the solution.
- The ENSMSE tested the solution in a use case involving encryption with AES.
- As a post-doctoral researcher at INRIA, I tested the `Cogito` countermeasure in a second use case involving `Verify PIN`.

4.3.2 How `Cogito` works

In simple words, `Cogito` is a way to automatically introduce classical countermeasures against physical attacks. It is a software solution at the level (2).
The idea is to introduce noise, desynchronization and random masks as explained in sections 3.2.3.2 and 3.2.3.1 with a unique tool. `Cogito` can be summarized as follows.

- `Cogito` introduces randomness during runtime code generation to produce polymorphic application components.
- `Cogito` uses semantic equivalences at the instruction level to produce different instances of code sequences.
- `Cogito` shuffles, at runtime, the machine instructions and randomizes the mapping to physical registers.
- `Cogito` can be combined with other hardware protections.
- `Cogito` limits memory consumption and fast code generation, so that it is applicable to very small computing units such as secure elements

The protections are presented in detail in the following.

4.3.2.1 Register allocator (RA)

The role of the register allocator (RA) is to map the endless number of single static assignment virtual registers to a limited number of physical registers. Thereby the RA attempts to reuse as much as possible each register in order to avoid assigning variables into memory locations (spill slots). For that, the RA implements an analysis pass called liveness analysis. This pass computes for each variable an interval in which the variable is considered alive. The liveness interval is a pair of program points, start and end, which starts when the value of the variable is defined and ends at the last point where the value is used (read). Thus, if two variables have disjoint liveness intervals, it means they can be assigned to the same physical register.

In dynamic compilers, instruction selection and instruction scheduling are usually performed before register allocation [215]. In `Cogito`, register allocation is done first, before instruction scheduling, by using a greedy algorithm. The purpose is to lighten the pressure on instruction selection and instruction scheduling: if register allocation is done first, it becomes possible to perform instruction scheduling from a much simpler intermediate representation, the allocator simply needs to maintain a list of the free registers available.

With `Cogito`, the same program changes the register allocation at each execution, which **introduces randomness**. The idea is to limit the fault injection against a specific register.

4.3.2.2 Instruction selection

Instruction selection is performed after RA once the runtime constants have been evaluated by the compilette. Instruction selection is done at the level of `cdg` instructions. Each expression can be mapped to one or more machine instructions depending on the target processor architecture and code generation options. For the purpose of achieving code polymorphism, supplementary variants are introduced to provide more opportunities for polymorphism. The semantic equivalences are used in the context of the experiment.

The semantic equivalences introduce **noise** and **desynchronization**.

4.3.2.3 Instruction scheduling

In traditional compilers, instruction scheduling aims at improving the performance of code execution: machine instructions are ordered in program memory in order to minimize execution time. The difficulty of scheduling lies in finding a possibly optimal ordering of machine instructions without breaking the semantics of the original source program.

In the case of code polymorphism, the aim is to exploit scheduling opportunities to generate many variants of the same source program, all functionally equivalent and semantically correct. Hence, code performance is only a secondary matter, and `Cogito` project considers resource constraints that impact code correctness, but not those that only impact program performance.
Instruction scheduling is performed in one pass: each time a new instruction is inserted in the instruction buffer. First the list of possible insertion slots is computed, by comparing the definitions and uses of the inserted instruction and of the instructions already stored in the buffer. Next, the insertion position is randomly selected among the list of insertion slots previously identified. If no insertion slot was found, the new instruction is appended at the end of the instruction buffer. If the instruction buffer is full, its first instruction is emitted in program memory to free one instruction slot.

The main idea behind this instruction scheduling is to introduce additional **desynchronization** and **redundancy** to limit fault injection against a specific instruction.

4.3.2.4 Insertion of noisy instructions

Noise instructions are appended to the end of the instruction buffer before the insertion of a new instruction. The noise instructions are inserted with a chosen probability. If needed, extra registers are randomly selected in the list of free registers. If no register is available, registers are randomly selected, pushed and popped on the stack before and after use. Several kinds of instructions can be inserted: core arithmetic instructions that execute in one processor cycle (*e.g.* integer operations add or sub) or in several processor cycles and memory accesses to a data table possibly provided by the user.

4.3.3 Results and limitations

In this section the results and the limitation of this project are presented.

4.3.3.1 Protection against side channel attacks

The results of `Cogito` against side-channel attacks are presented in [213]. The main results are:

- the number of side-channel observations necessary to perform a successful attack is increased (50 to 10000 traces);
- polymorphism can reduce information leakage.

The attack tested in [213] was limited to a CPA against unprotected AES. There is no proof that an attacker with greater computing power cannot analyse more traces and successfully launch the CPA.

During my post-doctorate, I was able to execute a successful template attack against `Verify PIN` even when `Cogito` was implemented. This implies that `Cogito` is not effective against template attacks.

The `Cogito` solution significantly slows down attackers, but does not entirely stop them. While it may be sufficient to deter amateur attackers, it is not a foolproof solution for strong security.

4.3.3.2 Protection against fault injection attacks

The scheduler rearranges the execution order of instructions to improve execution time while maintaining the original behaviour of the program. An analysis of this protection is presented in [212]. The authors use countermeasures included in `Cogito`, but not `Cogito` itself.

This countermeasure is designed to defend against fault injection attacks targeting a specific instruction. It could be a good protection against fault injection attacks aimed at skipping the call to compare arrays or the decrement counter in a `Verify PIN` scenario. However, it may not be effective against the new multi-fault attacks [188]. In cases where the attack does not target a specific instruction, such as in many DFA attacks, the attacker can simply execute the algorithm multiple times, until a desired fault is achieved.

4.3.3.3 Limitation of implementation

The problem I encountered with the `Cogito` solution stems from its conception. It is specifically designed with the idea of protecting symmetric cryptographic algorithms against CPA and DFA. To protect the `Verify PIN`, the use case slightly differs and two problems occur.

1. In a `Verify PIN`, an important protection is the constant time execution, independent of the `PIN` code candidate proposed by the user. Unfortunately, the random noise addition in `Cogito` does not respect this rule.
2. In a `Verify PIN` algorithm different flags are used to the protection of the algorithm. At the end of the computation, all the flags have to be true. The place of the flag is crucial. The idea is to check if all important computation such as, for example the decrement of the trial counter, is made in good time. Unfortunately, the instruction scheduling of `Cogito` breaks the semantics of the original flag `Verify PIN`. In practice, `Cogito` mixes the flag instructions with the other ones and thus nullifying their security effect.

A strong implementation of `Verify PIN` needs flags and a constant time execution. To keep that, two different protection of `Cogito`: noise insertion and instruction scheduling should turned off.

4.3.4 Conclusion

The `Cogito` project's best result is the automated inclusion of countermeasures (noise, redundancy, desynchronization, and masking) against physical attacks. The solutions presented in `Cogito` are intuitive, but they are imperfect patchwork solutions that randomly apply classical countermeasures. In practice, the countermeasures are not always effective at the moment of implementation, and there is no mathematical proof of security.

4.4 HAPEI [16]

I have worked with Ronan Lashermes from INRIA and Gaël Thomas from DGA-MI on hardware-assisted program execution integrity (HAPEI) published in [16]. It is important to note that this work was conducted prior to the publication of [170].

The main objective is to protect instructions from fault injection attacks by introducing integrity in code execution with a cryptographic tool, rather than relying on redundancy-based countermeasures.

We recognize the effectiveness of the SOFIA [211] solution, so we built our own solution based on it. A major issue with SOFIA is the separation between CFI and II. Since the CFI mechanism relies on the PC rather than the instruction value, an additional mechanism is necessary to ensure II. Finally, the multiplexor block must deal with possible predecessors in a non-trivial way. It modifies the control flow according to the actual predecessor where it should be predecessor agnostic (all legitimate predecessors must be dealt with in the same way).

HAPEI proposes a solution to these problems, by relaxing the efficiency requirements at a microarchitecture level.

4.4.1 First step

In order to harden the application, a preparatory step is required to encode the instructions. Only then, the application can be executed.

- **Packing** is required to create the encoded program, enriched with the necessary metadata, following the scheme detailed below. It must be done on the final device, since it requires a device-specific secret key K.
- During **execution**, the encoded instructions are decoded on-the-fly and executed. The decryption can only occur if the program state is correct.

4.4.2 The program execution integrity (PEI)

Here program execution integrity (PEI) denotes the combination of CFI and II.

SOFIA encodes the state of the program as the succession of PC_{prev} and PC. HAPEI proposes instead to encode the state of the program as the history of all previous executed instructions. This proposal does not depend on the PC value (apart when encoded in an instruction value). As such, the machine code is ensured to be executed correctly: instructions integrity is ensured together with CFI.

Secondly, it becomes easy to check at any time during execution that the current state of the program is valid.

It supposes that the CFG of our program is perfectly defined at compile time. There is no ambiguity on the destination address of jumps and calls. This condition is trivially satisfied if there are no indirect jumps or calls in our program. In the other case, it can be trickier.

Let acc_j (standing for accumulator) be a value representing the state of the program when instruction $\mathcal{I}_j$ is about to be executed (j uniquely identify one instruction). $\mathcal{I}_j$ and acc_j can be seen as values in $\mathbb{F}_{2^n}$ and $\mathbb{F}_{2^\nu}$, with n is the instruction size (considered fixed) and ν is the security parameter.

4.4.2.1 Starting step

To bootstrap the encoding, one has to use an initialization vector IV as input for the first executed instruction.

$$acc_{init} = \text{HMAC}_K(IV)$$

acc_{init} is considered as a predecessor program state to the entry instruction. It may be used in a multi-predecessor scheme or in the 1-predecessor one. HMAC is chosen because it is a pertinent cryptographic tool to guarantee the integrity.

4.4.2.2 1-predecessor

The easy case is the 1-predecessor case. The program snippet is a succession of instructions $\{\mathcal{I}_1, \mathcal{I}_2, \cdots, \mathcal{I}_j, \cdots\}$ where all instructions are executed in order.

The instructions are encoded as

$$\mathcal{I}'_j = \mathcal{C}\left(acc_j\right) \oplus \mathcal{I}_j,$$

where $\mathcal{C}$ is a compression function: a projection from $\mathbb{F}_{2^\nu}$ to $\mathbb{F}_{2^n}$. $\mathcal{C}$ must ensure that x cannot be deduced from $\mathcal{C}(x)$.

Obviously the state of the program must be updated, using secret key K:

$$acc_{j+1} = \text{HMAC}_K(acc_j || \mathcal{I}_j).$$

The state of the program is encoded with a hash chain depending on all previous instructions. The encoded instruction $\mathcal{I}'_j$ can only be decoded when the previous state of the program acc_j is correct. This is possible only when instruction $\mathcal{I}_j$ is due. Decoding necessitates the same operations:

$$\begin{aligned} acc_j &= \text{HMAC}_k(acc_{j-1} || \mathcal{I}_{j-1}), \\ \mathcal{I}_j &= \mathcal{C}\left(acc_j\right) \oplus \mathcal{I}'_j. \end{aligned}$$

4.4.2.3 A first solution with p-predecessors

It is possible to generalize in order to allow up to p predecessors for instruction and for a control flow with cycles.

In order to allow cycles, we must "rebase" our program state for all instructions having several predecessors. In this case, the program state is a new uniformly random value. The problem is now to map valid predecessor states to this same new state.

Let σ be a random value in $\mathbb{F}_{2^n}$. Let $acc_j^i, i \in [\![1, p]\!]$ be the allowed previous accumulator values for current instruction $\mathcal{I}_j$. A polynomial P can be defined such that $\forall i \in [\![1, p]\!], P(acc_j^i) = \sigma$ using Lagrange interpolation. Since the generated polynomial is minimal, it is constant if we do not define an additional point. $P(0) = \alpha$ for a random data such as $\alpha \neq \sigma, \in \mathbb{F}_{2^\nu}$.

The p coefficients of P are stored as program metadata with the corresponding instruction $\mathcal{I}_j$. At packing, HAPEI encodes with $\mathcal{I}'_j = \mathcal{C}(\sigma) \oplus \mathcal{I}_j$. To decode instruction $\mathcal{I}j$, the polynomial evaluation $acc_j = P(acc_{j-1})$ is used. Note that the polynomial evaluation replaces the HMAC call. So the strong integrity propriety cannot be guaranteed in a cryptographic definition.

4.4.2.4 A second solution with p-predecessors

$\mathbb{F}_{2^\nu}^*$ can be decomposed in different subgroups $\mathbb{G}_p$ where

$$\forall x \in \mathbb{G}_p, x^p = 1.$$

(subgroup of pth-root of unity). Such subgroups exist for all p such that $p \mid 2^n - 1$.

For all valid p (depending on n), a scheme that allows p predecessors is defined. For example, $n = 16$ allows a scheme with 5 predecessors ($p = 5$ divides $65535 = 2^{16} - 1$).

Let $acc_j^i, i \in [\![1, p]\!]$ be the allowed previous accumulator values for current instruction $\mathcal{I}_j$. Let σ be a random value in $\mathbb{F}_{2^\nu}$. Let $g \in \mathbb{G}_p$ be a generator value for the subgroup. We construct a polynomial P (in $\mathbb{F}_{2^\nu}$) using Lagrange interpolation such that $\forall i \in [\![1, p]\!], P(acc_j^i) = \sigma \cdot g^i$. The $p-1$ coefficients of P are stored as program metadata with the corresponding instruction $\mathcal{I}_j$. At packing, HAPEI encodes with $\mathcal{I}'_j = \mathcal{C}(\sigma^p) \oplus \mathcal{I}_j$. To decode instruction $\mathcal{I}_j$, $acc_j = P(acc_{j-1})^p$ is used. Indeed, by construction

$$\forall i \in [\![1, p]\!], P(acc_j^i)^p = \left(\sigma \cdot g^i\right)^p = \sigma^p \cdot (g^p)^i = \sigma^p.$$

In this scheme, polynomials have degree $p-1$ instead of p in first solution 4.4.2.3: the memory overhead is lower.

4.4.2.5 Ensuring instruction integrity

To verify the II, it is sufficient to compare the value of acc_j to a pre-generated truth value during the packing stage. The more often the check is performed, the sooner any tampering is detected, but this requires a larger amount of metadata.

A second strategy is to have valid instructions forming a set $\mathbb{I}_{valid} \subset \mathbb{F}_{2^\nu}$. If $\#(\mathbb{I}_{valid}) << \#(\mathbb{F}_{2^\nu})$ a wrongly decoded instruction has a very low probability of belonging to $\mathbb{I}_{valid}$, of being valid.

4.4.3 Key management

In this scheme, the component responsible for managing the secret key K is critical. In most cases, the binary encoding cannot be performed at compilation on the binary provider machine as it would require shipping the secret (which is then shared) along with the binary. As a result, any ISR scheme using a secret key must have a packing phase that transforms an unmodified binary (or extended with the CFG information) into a hardened one.

The other possibility is for the binary provider to encrypt the application for each intended recipient, then to use public-key cryptography to share the corresponding secret key with the targeted hardware.

4.4.4 Limitations and conclusion

This solution uses the program state, a hash chain of all previously executed instructions, in order to encode the program. Correct decode can only be achieved if the program state is correct. The difficulty lies in the multi-predecessors case. Here, the program state is reinitialized with a random value and a polynomial is computed that maps all previous program states to this new value. Two solutions have been explored 4.4.2.3 and 4.4.2.4.

- The problem with the first solution 4.4.2.3 is that P is constructed in a very specific way: Lagrange interpolation ensures that $P(X) - \sigma$ has p distinct roots. An attacker can use the peculiarity to gain information on σ and acc_j^i. Given a random polynomial Q of degree p, the probability that Q has p distinct roots corresponds to the number of combinations to distribute p roots over 2^n values divided by the total number of polynomials of degree p:

$$\frac{\binom{2^n}{p}}{2^{n\cdot(p+1)}}. \tag{4.2}$$

  Since in our case where $p \ll 2^n$, equation (4.2) becomes:

$$\frac{1}{p! \cdot 2^{n\cdot(p+1)}} \tag{4.3}$$

  As a conclusion, a proportion of $p!$ random polynomials have p roots. The greater the p, the better for the attacker that becomes able to discriminate σ. In most cases, p is low and 2^n is high ($n \geq 128$) so the security should not be compromised since the attacker cannot possibly enumerate all possible σ.

- The possibility for the attacker to discriminate σ given P is the main motivation for the alternative solution 4.4.2.4. In this proposition, σ is not a special value with respect to $P(X) - \sigma \cdot g^i$ may have any number of roots (≥ 1). But then, it means that additional roots may be considered valid program states. Some random illegitimate accumulator values could be mapped to the legitimate one. Since the attacker should not be able to control the accumulator value, the security is not compromised.

Thus the choice between solutions proposed in 4.4.2.3 and 4.4.2.4 depends on the attacker model. If they can control acc_j, then first solution must be chosen. If not but they have a huge computation power, second solution should be preferred.

Even though HAPEI appears to be an original mathematical concept, it has significant limitations.

The first major drawback is that these solutions for p-predecessors, which use a polynomial approach, consequently lose the cryptographic property of integrity that is guaranteed by an HMAC in the solution for a 1-predecessor.

The second major issue with the HAPEI solution is its impracticality. Unfortunately, this solution is utopian. Indeed its polynomial computations cannot be practically implemented. Computing a polynomial at each execution is too time-consuming. The performances are simply too poor.

Moreover, since the CFG must be perfectly known at packing time, indirect jumps and calls should be avoided. In particular, the scheme is not compatible with virtual method tables required for runtime polymorphism in several languages (`C++`, `java`, ...). Additionally, the scheme is tailored for self-contained applications. If the program must call external code (shared libraries, OS call), things do not play well.

4.5 Future works

4.5.1 Definition of the problem

Protecting programs against physical attacks is not trivial. The first question is: what kind of attack does it need to guard against? Previous works such as BSCARE introduced in 3.4 suggest that protecting a program's confidentiality is an extremely challenging problem. The control graph can always be retrieved. On the other hand, the problem of integrity in a program arises because FIA can alter it.

The solutions proposed in the previous sections of this chapter are not satisfactory.

`Cogito` is a solution at software level against all physical attacks. Unfortunately it is not enough to protect correctly against physical attacks. To find a real pertinent solution, it needed to descend into lower levels of abstraction.

HAPEI is a first solution at microarchitecture level. But it suffers from inefficient cryptographic schemes leading to inefficient designs in terms of performances, silicon area and energy consumption. Moreover it is not a protection against SCA.

I want to continue to study the problem of integrity in the microarchitecture. The solution HAPEI is clearly not adapted. But the idea to use cryptography to protect instructions stay pertinent as proven the recent works of Savry *et al.* [170, 216].

To precise the problem, I have to zoom on the different levels at the pipeline. A basic pipeline is composed of five stages as follows.

1. Fetch the instruction in instruction cache memory $I\$$.
2. Decode the instruction $\mathcal{I}$ and address the register in the register file;
3. Execute the instruction $\mathcal{I}$ with data in the register. Change the PC if necessary (branch).
4. Memorize the output of the instruction in the cache data memory $D\$$.
5. Write back in the register file.

The Figure 4.4 illustrates the five stages pipeline.

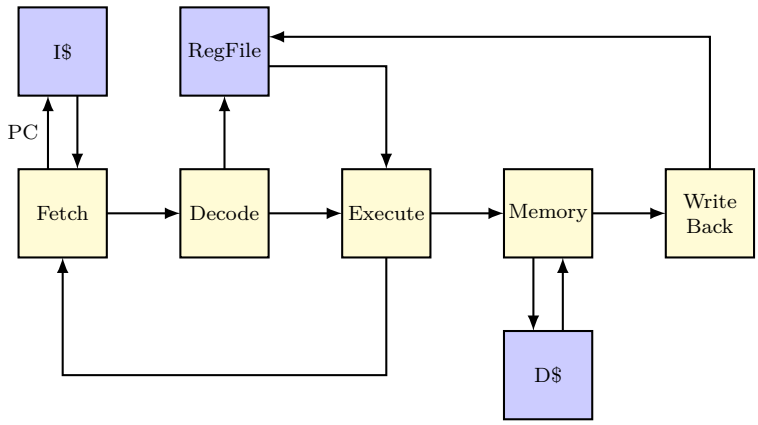


Figure 4.4: 5 stages pipeline

The initial set of open questions can be enumerated as follows.

- What is the model of an attacker? Where exactly are they able to induce the fault? Thus where in these stages should a future solution be placed?
- What tools should be used: MAC or encryption?
- Once instructions are encoded (with a cryptographic primitive), when should they be decoded?

I plan to launch a new project with Gaël Thomas from DGA-MI and Ronan Lashermes from INRIA on this subject to try to solve some of these problems.

4.5.2 Ideas

In this section the first research track is presented. In HAPEI, MAC and polynomial functions have been used but they have been shown to be the limiting factor in the applicability of the solution. We want to improve it to make it actually work in a real processor and have cryptographic guaranties. Our objective is to propose suitable solutions to protect the instructions in the aspects of both the control flow graph and the execution cycle.

We need to replace interpolating polynomials with a scheme based on MAC, purposely designed to meet our security (fault attack resilience) and performance criteria. We need a novel cryptographic primitive with a particular constraint: low latency. So we want to examine the different primitives for MAC with low-latency in mind. A first list of algorithms to study can be: Prince [217], Midori [218] or Mantis [219], Orthros [220], Qarma [221] and probably the most pertinent: Speedy [222].

The simple case in the control flow graph is when an instruction has just one predecessor. In this simple case, applying a MAC is enough. When an instruction has multiple predecessors, a more complex protocol is required. The previous schemes in the literature [16, 211, 216] suffer from inefficient cryptographic schemes leading to inefficient designs in terms of performances, silicon area and energy consumption. Helped with our new custom MAC, we should devise a new scheme relying on it to ensure program execution integrity. It should be an ISR scheme: using a MAC integrity scheme even for multipredecessors instructions. The verification

should be possible only when the initial control flow has been respected and if the protected program has not been tampered with.

We will have to integrate the specially crafted MAC and protocol into a hardened processor, starting from an unsecured one for comparison. At first, we will consider that programs have no indirect control flow redirection (indirect jumps) since they are notably hard to protect with an ISR scheme. Several trade-offs have to be explored, between security and performances notably. We project to use the LHS from INRIA to evaluate the resistance of the implementation under clock glitches and electromagnetic fault injection attacks. The solution should have an impact as low as possible on the actual speed of the protected CPU compared to an unprotected one.

This is a multidisciplinary project involving both cryptography and microarchitecture. Our solution, by being integrated directly in the pipeline, could be used in complex cores (even Out-of-Order cores), offer a strong, cryptographic, guarantee on the integrity of instructions but also of the control flow. This solution is future proof in the context of injection techniques getting more precise and more powerful.

4.6 Conclusion

Protecting devices, especially the instructions in a program, against physical attacks is a crucial necessity. In this chapter, various attempts to tackle this challenge and tracks for a future project are presented. The problem of integrity in the microarchitecture is a rising concern today.

The `Cogito` project's best outcome is the automation of including countermeasures such as noise and desynchronization against physical attacks. However, this is not an ultimate solution as instructions need real integrity to be protected against faults.

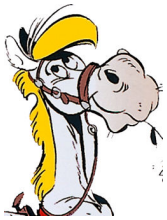
Symmetric cryptography can be a viable solution to address this problem to some extent. HAPEI is a work that encodes and decodes instructions on the fly. In practice, only in the case of instructions with just one predecessor, a cryptographic solution is proposed: using an HMAC. In the other cases, polynomial functions have been used.

To make instructions secure, cryptographic integrity is essential. In a future collaboration project, I aim to improve HAPEI solution by using low latency MAC. However, many questions still need to be addressed, and I hope to obtain funding to work on this significant challenge.

Chapter 5

Conclusion and Future works

"*Je me demande de quelle façon loufoque nous allons nous en sortir...*"
Jolly Jumper dans *Lucky Luke Le fil qui chante*
de Morris et René Goscinny.



Contents

5.1 Different views of symmetric cryptography

Symmetric cryptography remains a useful tool for security. It must adapt to current needs and constraints as for example in the IoT context. Consequently, the NIST has recently standardized new lightweight AEAD algorithms, which are well-suited for constrained environments where the performance of traditional symmetric cryptographic standards proves to be inadequate. Moreover, AEAD guarantee both confidentiality and integrity, thus replacing both encryption and MAC with just a single tool.

My work in this field is not about classical cryptanalysis or developing new primitives. I study physical cryptanalysis and various other original aspects of symmetric cryptography algorithms.

The chapter 2 presents my work on ransomware. Ransomware uses symmetric cryptography to take user data hostage. So in case of ransomware, symmetric cryptography is a threat. Since the start of the pandemic, there has been an evolution in the use of cryptography by ransomware. Indeed initially, the first ransomware programs did not use the tools correctly, as for example the encryption mode. Then, ransomware utilized the victims' cryptographic tools. Today, ransomware is proficient in cryptography. Ransomware ceased making significant mistakes in cryptography. Ransomware programs employ their own cryptographic protocols, which utilize various standards. Consequently, conducting cryptanalysis on ransomware cryptography is equivalent to performing cryptanalysis on the standards themselves. An alternative solution involves detecting the encryption process. In this approach, Shannon's properties of diffusion and confusion are considered as leakages. Indeed, these properties increase the entropy of data. It is likely that future ransomware will utilize cryptosystems based on the Hilbert matrix to avoid impacting entropy during encryption.

In the chapter 3, physical cryptanalysis targets new lightweight AEAD primitives. This chapter presents two new BSCA for retrieving the secret key. Moreover, these attacks give clues to improve the design of the different primitives. In the case of Elephant, we demonstrated that the different designs of a LFSR are not equivalent. For Sparkle, we highlighted the leakage of different basic functions.
Still within chapter 3, a BSCARE is presented against MAC and decryption primitives. The goal of this attack is to recompute the inputs using only the known entries, while the key and the primitive algorithm remain unknown.

In the final chapter 4, we aim to protect the instructions of security algorithms, such as cryptographic algorithms. To achieve this, various initial approaches have been tested. The results were not significant. However, these efforts have opened doors to a new field of research. How to use cryptography at a microarchitecture level to protect instructions? A future area of study involves examining low-latency primitives for MAC.

5.2 Another perspective for physical leakages

With physical attacks, the involuntary physical leakage of a device helps an attacker. However, always with the intention of reversing approaches, these leakages can be viewed as a valuable source of information to verify if a device is performing the computations it is supposed to. Indeed, physical leakages do not lie.

I am the advisor for Awaleh Houssein Meraneh's PhD thesis. In his thesis, Awaleh is working to find new supervision tools for industrial cyber-physical systems (ICPS). ICPS are intelligent systems that integrate engineered networks of physical and computational components such as automata, robots, *etc.* They are the prime targets for cyber attacks.
To counter these threats, proactive measures such as anomaly detection systems are deployed. However, some attacks can bypass network security, necessitating alternative detection methods. In this context, physical leakages are pertinent to detect abnormal behaviour.
More specifically, Awaleh has chosen to focus on sound. Sound-based anomaly detection (SAD) involves determining whether the sound emitted by a target machine deviates from the norm [223]. This system's sound data can be easily captured using low-cost equipment, only a microphone is needed. Furthermore, the typical behaviour of systems can be accurately fingerprinted using sound.
I plan to recruit another PhD student to cross sound with power consumption information to improve the results obtained during Awaleh thesis.

Just to give another use case, today ransomware does not attack only computer but small devices too. At

the crossroads between ransomware detection and side channel analysis, it is possible to detect a ransomware on a device by physical leakage, as shown the recent work of Pham *et al.* in [224].
With the same philosophy, others detect hardware trojan or backdoor [225–227].

To finish, in future projects, I would like to explore an intuition. A cryptographic algorithm is a product of rigorous research, and every detail in its design is meticulously considered. It is conceivable that even slight modifications could render these algorithms vulnerable. A properly performed fault injection could trigger a backdoor in a cryptographic primitive. This modification could be performed using a laser or electromagnetic radiation fault injection. Thus, a secure algorithm would become weak under faults. These works would allow collaboration with other research institutes that have laser fault injection benches available, like ENSMSE.

Acronyms

fai fractional added information. 7, 45, 64

PIN personal identification number. 6, 11, 12, 14, 24, 45, 50, 51, 69, 72, 74, 77

AEAD authenticated encryption with associated data. 5, 7, 10, 14–16, 20, 22, 45, 47, 53, 59, 66, 85

AES advanced encryption standard. 5, 10, 16, 18, 29, 31, 47, 52, 56, 62, 65, 66, 69, 73, 75, 76

ALU arithmetic logic unit. 13

ANR Agence Nationale de la Recherche. 11, 74

ANSSI Agence Nationale de la Sécurité des Systèmes d'Information. 24, 43

APCIL Attaques Physiques sur les Chiffrements Intègres Légers. 11, 47, 53

API application programming interface. 6, 26, 29–32

BP belief propagation. 7, 45, 56, 58–61, 66, 92, 94

BSCA blind side channel analysis. 7, 45, 47, 51–53, 55, 59, 60, 62, 66, 85

BSCARE blind side channel analysis for reverse engineering. 7, 45, 47, 62, 63, 65, 66, 80, 85

CBC cipher block chaining. 5, 6, 10, 19, 26, 31, 33, 34

CEA Commissariat à l'Énergie Atomique. 24, 74, 75

CFG control flow graph. 69, 72, 78–80

CFI control flow integrity. 69, 73, 77, 78

CIA code injection attack. 69

CNG cryptography API next generation. 31

CPA correlation power analysis. 51, 76, 77

CPU central processing unit. 13, 82

CRA code-reuse attack. 69

CSP cryptographic service provider. 31, 32

CTR counter. 5, 10, 19, 20, 65, 66

DaD data aware defence. 41, 42

DFA differential fault analysis. 72, 77

DGA-MI Direction Génerale de l'Armement Maîtrise de l'Information. 27, 35, 42, 47, 53, 77, 81

DI data integrity. 69

DLL dynamic link library. 31, 38

DPA differential power analysis. 51

ECB electronic code book. 5, 6, 10, 18, 26, 31, 33

ECC elliptic curve cryptographic. 31

EM electromagnetic. 6, 45, 48–51, 60, 71, 73

ENSMSE École Nationale Supérieure des Mines de Saint-Étienne. 11, 24, 74, 75, 86

FIA fault injection attack. 13, 47, 69, 70, 73, 80

FIPS federal information processing standard. 31

FP false positive. 35, 38

GCM Galois counter mode. 5, 10, 20, 31, 34

GF Galois field. 16, 92

GFN generalized Feistel network. 17, 66

HAPEI hardware-assisted program execution integrity. 8, 68, 70, 77–82

HD Hamming distance. 51

HDR Habilitation à Diriger des Recherches. 5, 10, 11

HMAC hash-based message authentication code. 15, 78–80, 82

HW Hamming weight. 7, 45, 51, 53–56, 58–60, 66, 92

I/O input/output. 41, 71

ICPS industrial cyber-physical systems. 85

IDS intrusion detection system. 39

IFH information flow hypergraph. 7, 46, 64

II instructions integrity. 69, 73, 77–79

IID independent and identically distributed. 34

INRIA Institut national de recherche en sciences et technologies du numérique. 4, 11, 24, 27, 37, 42, 47–49, 62, 71, 74, 75, 77, 81, 82

IoT internet of things. 11, 12, 24, 50, 85

IRISA Institut de Recherche en Informatique et Systèmes Aléatoires. 4, 11

ISA instruction set architecture. 62

ISR instruction set randomization. 8, 68, 73, 79, 81, 82

LDPC low-density parity-check. 56

LFSR linear feedback shift register. 7, 20, 21, 45, 53–55, 57, 59–62, 66, 85, 94

LHS laboratory high security. 11, 14, 40, 42, 48, 49, 59, 71, 82

MAC message authentication code. 15, 18, 47, 65, 73, 74, 81, 82, 85

MC MixColumns. 16

MoM malware-o-matic. 14, 40, 41, 43

MOOC massive open online course. 24

MS CAPI Microsoft's cryptographic application programming interface. 30–32

MSCS Mastère Spécialisé en Cyber-Sécurité. 11, 24

NIDS network intrusion detection systems. 30

NIST national institute of standards and technology. 11, 15, 16, 20, 31, 47, 53, 55, 59, 66, 85

NOP no operation. 72

OCIF “Objets Communicants pour l’Internet du Futur”. 4, 11, 47

OS operating system. 41, 80

PC program counter. 73, 77, 78, 81

PEI program execution integrity. 8, 68, 78

RA register allocator. 8, 68, 75, 76

RISC reduced instruction set computer. 62, 63

ROP return-oriented programming. 69

RTL register transfer logic. 8, 13, 68, 72

SAD sound-based anomaly detection. 85

SB SubBytes. 16

SCA side channel analysis. 6, 13, 14, 45, 47, 48, 50–52, 56, 62, 69, 72–74, 80

SCARE side channel analysis for reverse engineering. 7, 45, 47, 52, 62, 66

SI system integrity. 69

SOFIA software and control flow integrity architecture. 73, 77, 78

SPA simple power analysis. 6, 45, 49

SR ShiftRow. 16

StI state integrity. 69

TAMIS threat analysis and mitigation for information security. 11

TF-IDF term frequency-inverse document frequency. 6, 26, 38

Notations

- A denotes data associated.
- $\mathcal{A}$ denotes Alzette.
- B denotes a byte.
- $\mathcal{B}$ is the better belief.
- b denotes a bit.
- C is a ciphertext.
- $\mathcal{C}$ denotes compression function.
- c is a carry.
- clk means clocks.
- Dn means decryption.
- $\$D$ is a the data cash.
- $\mathcal{E}$ an equation or a factor node in a BP.
- En means encryption.
- e denotes an error.
- F is a distribution function.
- $\mathcal{F}$ is a round or Feistel function.
- $\mathbf{F}$ is a probability density function.
- f denotes a function.
- GF means Galois field (GF).
- $\mathbb{G}$ is a group or subgroup.
- $\mathcal{G}$ is function.
- g denotes a group or subgroup generator.
- $\mathcal{H}$ is a hash function.
- H is the Shannon entropy.
- $H0$ is the null hypothesis.
- HW is a Hamming weight (HW).
- $\tilde{\text{HW}}$ is a noisy HW.
- I is an input matrix.
- $\mathcal{I}$ is an instruction.
- $\mathbb{I}$ is a set of instructions.
- $\$I$ is a the instruction cache.
- i is used for index.
- IV is an initial vector.
- j is used for index.
- K denotes a key.
- L is for left.

- ℓ is a linear application.
- l is an index
- M is a plaintext or a file.
- $\mathcal{M}$ is a trace matrix.
- m is a memory.
- $\mathbf{m}$ is a mask.
- N is a nonce.
- $\mathcal{N}$ is a Gaussian.
- N denotes a set of neighbours.
- n denotes a size.
- O is an output matrix.
- o is an occurrence count.
- P is a permutation.
- $\mathbb{P}$ is a probability.
- $\mathcal{P}$ is a procedure.
- P is a polynomial.
- p denotes a number of predecessors.
- R is for right.
- r is a round.
- $\mathbb{S}$ is the set of the possible successor s.
- S_I, S_O and $S_{\mathcal{M}}$ are size of respectively matrix I, O and $\mathcal{M}$.
- s is a successor or a sample.
- T is a tag.
- $\mathcal{T}$ is a transition matrix.
- t is a time.
- $\mathbf{t}$ is a trace.
- v is a binary vector.
- $\mathbf{v}$ is model of vector.
- $\mathcal{W}$ is a whitening layer.
- x is a value or a variable.
- X denotes a random variable or a Markov chain according to the context.
- $\mathbb{X}$ is the set of definition of the variable X.
- y is a random value.
- z is used for index.
- α is a constant.
- β denotes a bloc of bits.

- χ^2 is the test of χ^2.
- Γ_B is the number of hits observed for the byte B.
- γ_B is the expected number according to a known distribution function.
- Λ denotes the likelihood ratio.
- ν is a size or a number of bits.
- μ a message in a BP algorithm.
- φ is a LFSR.
- (φ) denotes a retroaction equation of one LFSR.
- σ is a noise or a random data.
- $*$ means faulty.
- $\oplus$ denotes a xor.
- $\boxplus$ denotes a modular addition.
- $\perp$ is returned when the integrity of the message is not guarantee.
- $\ll$ is a shift left.
- $\lll$ is a rotation left.
- $^\circ$ denotes tested.
- $||$ means concatenation.
- $\#$ means cardinal.

Bibliography

[1] R. Moussaileb, N. Cuppens, J.-L. Lanet, and H. Le Bouder, "A survey on windows-based ransomware taxonomy and detection mechanisms," *ACM Computing Surveys (CSUR)*, vol. 54, no. 6, pp. 1–36, 2021.

[2] A. Palisse, H. Le Bouder, J.-L. Lanet, C. Le Guernic, and A. Legay, "Ransomware and the legacy crypto api," in *International Conference on Risks and Security of Internet and Systems*. Springer, 2016, pp. 11–28.

[3] A. Palisse, A. Durand, H. Le Bouder, C. Le Guernic, and J.-L. Lanet, "Data Aware Defense (DaD): Towards a Generic and Practical Ransomware Countermeasure," in *NordSec: Nordic Conference on Secure IT Systems*. Springer, 2017, pp. 192–208.

[4] N. Bailluet, H. Le Bouder, and D. Lubicz, "Ransomware detection using markov chain models over file headers," 2021.

[5] Y. Lemmou, H. Le Bouder, and J.-L. Lanet, "Discriminating Unknown Software Using Distance Model," in *ICACSIS: 11th International Conference on Advanced Computer Science and Information Systems*, *Bali*, Indonesia, 2019.

[6] R. Moussaileb, B. Bouget, A. Palisse, H. Le Bouder, N. Cuppens, and J.-L. Lanet, "Ransomware's early mitigation mechanisms," in *Proceedings of the 13th International Conference on Availability, Reliability and Security*. ACM, 2018, p. 2.

[7] R. Moussaileb, N. Cuppens, J.-L. Lanet, and H. Le Bouder, "Ransomware Network Traffic Analysis for Pre-Encryption Alert," in *FPS: 12th International Symposium on Foundations & Practice of Security*, 2019.

[8] S. K. Bukasa, R. Lashermes, H. Le Bouder, J.-L. Lanet, and A. Legay, "How trustzone could be bypassed: Side-channel attacks on a modern system-on-chip," *Information Security Theory and Practice*, p. 93, 2017.

[9] H. Le Bouder, T. Barry, D. Couroussé, J.-L. Lanet, and R. Lashermes, "A template attack against VERIFY PIN algorithms," in *SECRYPT*, 2016, pp. 231–238.

[10] H. Le Bouder, R. Lashermes, Y. Linge, G. Thomas, and J.-Y. Zie, "A multi-round side channel attack on aes using belief propagation," in *International Symposium on Foundations and Practice of Security*. Springer, 2016, pp. 199–213.

[11] A. Houssein Meraneh, C. Clavier, H. Le Bouder, J. Maillard, and G. Thomas, "Blind Side Channel On The Elephant LFSR," in *SECRYPT*, 2022.

[12] J. Maillard, A. Houssein Meraneh, M. Sarry, C. Clavier, H. Le Bouder, and G. Thomas, "Blind side channel on the Elephant LFSR Extended version," in *Springer Book of SECRYPT*, 2023.

[13] M. Sarry, H. Le Bouder, E. Maalouf, and G. Thomas, "Blind Side Channel Analysis against AEAD with a Belief Propagation Approach," in *Cardis*. Springer, 2023.

[14] R. Lashermes and L. B. Hélène, "Generic SCARE: reverse engineering without knowing the algorithm nor the machine." in the process of submission.

[15] D. Couroussé, T. Barry, B. Robisson, N. Belleville, P. Jaillon, O. Potin, H. L. Bouder, J. Lanet, and K. Heydemann, "All paths lead to rome: Polymorphic runtime code generation for embedded systems," in *Proceedings of the Fifth Workshop on Cryptography and Security in Computing Systems, CS2*. ACM, 2018, pp. 17–18.

[16] R. Lashermes, H. Le Bouder, and G. Thomas, "Hardware-Assisted Program Execution Integrity: HAPEI," in *NordSec Nordic Conference on Secure IT Systems*. Springer, 2018.

[17] "Runtime Code Generation to Secure Devices COGITO." [Online]. Available: https://anr.fr/Project-ANR-13-INSE-0006

[18] A. Palisse, "Analyse et détection de logiciels de rançon," Ph.D. dissertation, Rennes 1, 2018.

[19] L. Ouairy, "Analyse des vulnérabilités dans des systèmes embarqués face à des attaques par fuzzing," Ph.D. dissertation, Rennes 1, 2020.

[20] L. Ouairy, H. Le Bouder, and J.-L. Lanet, "Normalization of Java source codes," in *SECITC*, 2018.

[21] ——, "Protection of systems against fuzzing attacks," in *FPS*, 2018.

[22] ——, "JavaNeighbors: Improving ChuckyJava's neighborhood discovery algorithm," in *EUSPN - 10th International Conference on Emerging Ubiquitous Systems and Pervasive Networks*, 2019.

[23] L. Ouairy, H. L. Bouder, and J.-L. Lanet, "Confiance: detecting vulnerabilities in java card applets," in *Proceedings of the 15th International Conference on Availability, Reliability and Security*, 2020, pp. 1–10.

[24] H. Le Bouder, E. Thomas, Gaëland Bourget, M. Graa, N. Cuppens, and J. Lanet, "Theoretical security evaluation of the human semantic authentication protocol," in *SECRYPT,*, 2018, pp. 498–505.

[25] S. Blanc, A. Lahmadi, K. Le Gouguec, M. Minier, and L. Sleem, "Benchmarking of lightweight cryptographic algorithms for wireless iot networks," *Wireless Networks*, vol. 28, no. 8, pp. 3453–3476, 2022.

[26] F.-X. Standaert, T. G. Malkin, and M. Yung, "A unified framework for the analysis of side-channel key recovery attacks," in *Advances in Cryptology-EUROCRYPT: 28th Annual International Conference on the Theory and Applications of Cryptographic Techniques*. Springer, 2009, pp. 443–461.

[27] S. Picek, A. Heuser, G. Perin, and S. Guilley, "Profiling side-channel analysis in the efficient attacker framework," *Cryptology ePrint Archive*, 2019.

[28] A. Ito, R. Ueno, and N. Homma, "Perceived information revisited: New metrics to evaluate success rate of side-channel attacks," *IACR Transactions on Cryptographic Hardware and Embedded Systems*, pp. 228–254, 2022.

[29] A. Gangolli, Q. H. Mahmoud, and A. Azim, "A systematic review of fault injection attacks on iot systems," *Electronics*, vol. 11, no. 13, 2022.

[30] H. Le Bouder, G. Thomas, R. Lashermes, Y. Linge, B. Robisson, and A. Tria, "An evaluation tool for physical attacks," in *International Conference on Ad-Hoc Networks and Wireless*. Springer, 2018, pp. 112–119.

[31] B. Robisson and H. Le Bouder, "Physical functions: the common factor of side-channel and fault attacks?" *Journal of Cryptographic engineering*, vol. 6, no. 3, pp. 217–227, 2016.

[32] H. Le Bouder, R. Lashermes, Y. Linge, B. Robisson, and A. Tria, "A Unified Formalism for Physical Attacks," *IACR Cryptology ePrint*, 2014.

[33] R. Moussaileb, "Log analysis for malicious software detection," Ph.D. dissertation, Ecole nationale supérieure Mines-Télécom Atlantique Bretagne Pays de la Loire, 2020.

[34] C. E. Shannon, "A mathematical theory of communication," *The Bell system technical journal*, vol. 27, no. 3, pp. 379–423, 1948.

[35] NIST, *Lightweight Cryptography Standardization Process*, 2018. [Online]. Available: https://csrc.nist.gov/projects/lightweight-cryptography

[36] C. Dobraunig, M. Eichlseder, F. Mendel, and M. Schläffer, "Ascon," *Submission to the CAESAR competition*, 2014.

[37] T. Beyne, Y. L. Chen, C. Dobraunig, and B. Mennink, "Dumbo, jumbo, and delirium: Parallel authenticated encryption for the lightweight circus," *IACR Transactions on Symmetric Cryptology*, pp. 5–30, 2020.

[38] S. Banik, A. Chakraborti, A. Inoue, T. Iwata, K. Minematsu, M. Nandi, T. Peyrin, Y. Sasaki, S. M. Sim, and Y. Todo, "Gift-cofb," *Cryptology ePrint Archive*, 2020.

[39] M. Hell, T. Johansson, A. Maximov, W. Meier, and H. Yoshida, "Grain-128aead, round 3 tweak and motivation," 2021.

[40] C. Dobraunig, M. Eichlseder, S. Mangard, F. Mendel, B. Mennink, R. Primas, and T. Unterluggauer, "Isap v2. 0," 2020.

[41] Z. Bao, A. Chakraborti, N. Datta, J. Guo, M. Nandi, T. Peyrin, and K. Yasuda, "Photon-beetle authenticated encryption and hash family," *NIST Lightweight Compet. Round*, vol. 1, p. 115, 2019.

[42] M. Khairallah, "Romulus: Lighweight AEAD from Tweakable Block Ciphers," in *Hardware Oriented Authenticated Encryption Based on Tweakable Block Ciphers*. Springer, 2022, pp. 115–134.

[43] C. Beierle, A. Biryukov, L. C. Dos Santos, J. Großschädl, L. Perrin, A. Udovenko, V. Velichkov, Q. Wang, and A. Biryukov, "Schwaemm and esch: lightweight authenticated encryption and hashing using the sparkle permutation family," *NIST round*, vol. 2, 2019.

[44] H. Wu and T. Huang, "Tinyjambu: A family of lightweight authenticated encryption algorithms (version 2)," *Submission to the NIST Lightweight Cryptography Standardization Process*, 2021.

[45] J. Daemen, S. Hoffert, M. Peeters, G. V. Assche, and R. V. Keer, "Xoodyak, a lightweight cryptographic scheme," 2020.

[46] NIST, "Specification for the Advanced Encryption Standard," *FIPS PUB 197*, vol. 197, 2001.

[47] T. Suzaki, K. Minematsu, S. Morioka, and E. Kobayashi, "TWINE: A Lightweight Block Cipher for Multiple Platforms," in *International Conference on Selected Areas in Cryptography*. Springer, 2012, pp. 339–354.

[48] T. Suzaki and K. Minematsu, "Improving the generalized feistel," in *International Workshop on Fast Software Encryption*. Springer, 2010, pp. 19–39.

[49] J. Daemen and V. Rijmen, "The mac function pelican 2.0," *Cryptology ePrint Archive*, 2005.

[50] M. J. Dworkin, *Sp 800-38d. recommendation for block cipher modes of operation: Galois/counter mode (GCM) and GMAC*. National Institute of Standards & Technology, 2007.

[51] T. Beyne, Y. L. Chen, C. Dobraunig, and B. Mennink, "Elephant v2," *NIST lightweight competition*, 2021.

[52] D. J. Bernstein, "How to Stretch Random Functions: The Security of Protected Counter Sums," *J. Cryptol.*, 1999.

[53] A. Luykx, B. Preneel, E. Tischhauser, and K. Yasuda, "A MAC Mode for Lightweight Block Ciphers," in *Fast Software Encryption FSE*, T. Peyrin, Ed. Springer, 2016.

[54] R. Granger, P. Jovanovic, B. Mennink, and S. Neves, "Improved Masking for Tweakable Blockciphers with Applications to Authenticated Encryption," in *EUROCRYPT*. Springer, 2016.

[55] A. Bogdanov, M. Knezevic, G. Leander, D. Toz, K. Varici, and I. Verbauwhede, "Spongent: a Lightweight Hash Function," in *Cryptographic Hardware and Embedded Systems - CHES*, vol. 6917. Springer, 2011, pp. 312–325.

[56] G. Bertoni, J. Daemen, M. Peeters, and G. van Assche, "The Keccak Reference," 2011.

[57] NIST, "SHA-3 Standard: Permutation-Based Hash and Extendable-Output Functions," *FIPS 202*, 2015.

[58] A. L. Young and M. Yung, "Cryptovirology: Extortion-Based Security Threats and Countermeasures," in *IEEE Symposium on Security and Privacy*, 1996.

[59] Z. A. Genç, G. Lenzini, and P. Y. Ryan, "Next generation cryptographic ransomware," in *Nordic Conference on Secure IT Systems*. Springer, 2018.

[60] B. A. S. Al-rimy, M. A. Maarof, and S. Z. M. Shaid, "Ransomware threat success factors, taxonomy, and countermeasures: A survey and research directions," *Computers & Security*, 2018.

[61] S. Aurangzeb, M. Aleem, M. A. Iqbal, M. A. Islam *et al.*, "Ransomware: a survey and trends," *J. Inf. Assur. Secur*, 2017.

[62] J. Castiglione and D. Pavlovic, "Dynamic distributed secure storage against ransomware," *IEEE Transactions on Computational Social Systems*, 2019.

[63] M. Baykara and B. Sekin, "A novel approach to ransomware: Designing a safe zone system," in *ISDFS*. IEEE, 2018.

[64] M. M. Ahmadian, H. R. Shahriari, and S. M. Ghaffarian, "Connection-monitor & connection-breaker: A novel approach for prevention and detection of high survivable ransomwares," in *(ISCISC)*, 2015.

[65] K. Cabaj, M. Gregorczyk, and W. Mazurczyk, "Software-defined networking-based crypto ransomware detection using http traffic characteristics," *Computers & Electrical Engineering*, 2018.

[66] A. O. Almashhadani, M. Kaiiali, S. Sezer, and P. OKane, "A multi-classifier network-based crypto ransomware detection system: a case study of locky ransomware," 2019.

[67] C. G. Akcora, Y. Li, Y. R. Gel, and M. Kantarcioglu, "Bitcoinheist: Topological data analysis for ransomware detection on the bitcoin blockchain," *arXiv preprint*, 2019.

[68] Z.-G. Chen, H.-S. Kang, S.-N. Yin, and S.-R. Kim, "Automatic ransomware detection and analysis based on dynamic api calls flow graph," in *Proceedings of the International Conference on Research in Adaptive and Convergent Systems*, 2017.

[69] B. A. S. Al-rimy, M. A. Maarof, Y. A. Prasetyo, S. Z. M. Shaid, and A. F. M. Ariffin, "Zero-day aware decision fusion-based model for crypto-ransomware early detection," *International Journal of Integrated Engineering*, 2018.

[70] B. A. S. Al-rimy, M. A. Maarof, and S. Z. M. Shaid, "Crypto-ransomware early detection model using novel incremental bagging with enhanced semi-random subspace selection," *Future Generation Computer Systems*, 2019.

[71] C. Moore, "Detecting ransomware with honeypot techniques," in *Cybersecurity and Cyberforensics Conference (CCC)*. IEEE, 2016.

[72] M. W. Patton, N. Scott, R. R. Gutierrez, and S. Giovannini, "Behavior-based ransomware detection using decoy files," 2019.

[73] N. Scaife, H. Carter, P. Traynor, and K. R. B. Butler, "CryptoLock (and Drop It): Stopping Ransomware Attacks on User Data," in *ICDCS*. IEEE, 2016.

[74] A. Continella, A. Guagnelli, G. Zingaro, G. D. Pasquale, A. Barenghi, S. Zanero, and F. Maggi, "ShieldFS: A Self-healing, Ransomware-aware Filesystem," in *ACSAC*. ACM, 2016.

[75] A. Kharraz and E. Kirda, "Redemption: Real-Time Protection Against Ransomware at End-Hosts," in *RAID 2017*.

[76] A. Kharraz, S. Arshad, C. Mulliner, W. K. Robertson, and E. Kirda, "UNVEIL: A Large-Scale, Automated Approach to Detecting Ransomware," in *USENIX*, 2016.

[77] Microsoft, "Microsoft enhanced cryptographic provider, fips 140-1 documentation security policy," 2005. [Online]. Available: http://csrc.nist.gov/groups/STM/cmvp/documents/140-1/140sp/140sp238.pdf

[78] U.S Departement of commerce / national Institute of standards and Technology, "Data Encryption Standart DES."

[79] V. Kotov and M. S. Rajpal, "Bromium: Understanding cryptoransomware." [Online]. Available: https://www.bromium.com/sites/default/files/bromium-report-ransomware.pdf

[80] G. Hunt and D. Brubacher, "Detours: Binaryinterception ofwin 3 2 functions," in *3rd usenix windows nt symposium*, 1999.

[81] G. R. Blakley and I. Borosh, "Rivest-shamir-adleman public key cryptosystems do not always conceal messages," *Computers & mathematics with applications*, vol. 5, no. 3, pp. 169–178, 1979.

[82] H. Gilbert and H. Handschuh, "Security analysis of sha-256 and sisters," in *International workshop on selected areas in cryptography*. Springer, 2003, pp. 175–193.

[83] P. Syverson, "A taxonomy of replay attacks [cryptographic protocols]," in *Proceedings The Computer Security Foundations Workshop VII*. IEEE, 1994, pp. 187–191.

[84] "Gnu octave: Scientific programming language." [Online]. Available: https://octave.sourceforge.io/octave/function/chi2inv.html

[85] P. J. Huber and V. Strassen, "Minimax tests and the neyman-pearson lemma for capacities," *The Annals of Statistics*, 1973.

[86] M. Kuhkan. Dorma Trading, Est. Publishing Manager, 2016, vol. 8, no. 6.

[87] S. R. Safavian and D. Landgrebe, "A survey of decision tree classifier methodology," *IEEE transactions on systems, man, and cybernetics*, vol. 21, no. 3, pp. 660–674, 1991.

[88] T. K. Ho, "Random decision forests," vol. 1, pp. 278–282, 1995.

[89] Clonezilla, "The free and open source software for disk imaging and cloning."

[90] Viper, "Binary management and analysis framework."

[91] A. Continella, A. Guagnelli, G. Zingaro, G. De Pasquale, A. Barenghi, S. Zanero, and F. Maggi, "Shieldfs: a self-healing, ransomware-aware filesystem," in *Proceedings of the 32nd Annual Conference on Computer Security Applications*, 2016, pp. 336–347.

[92] A. Kharaz, S. Arshad, C. Mulliner, W. Robertson, and E. Kirda, "{UNVEIL}: A large-scale, automated approach to detecting ransomware," in *25th {USENIX} Security Symposium ({USENIX} Security 16)*, 2016, pp. 757–772.

[93] A. Kharraz, W. Robertson, D. Balzarotti, L. Bilge, and E. Kirda, "Cutting the gordian knot: A look under the hood of ransomware attacks," in *International Conference on Detection of Intrusions and Malware, and Vulnerability Assessment*. Springer, 2015, pp. 3–24.

[94] N. Scaife, H. Carter, P. Traynor, and K. R. Butler, "Cryptolock (and drop it): stopping ransomware attacks on user data," in *36th International Conference on Distributed Computing Systems (ICDCS)*. IEEE, 2016, pp. 303–312.

[95] Microsoft, "Microsoft: File system minifilter drivers." [Online]. Available: https://msdn.microsoft.com/en-us/windows/hardware/drivers/ifs/file-system-minifilter-drivers

[96] Z. A. Genç, G. Lenzini, and P. Y. Ryan, "Nocry: No more secure encryption keys for cryptographic ransomware," in *International Workshop on Emerging Technologies for Authorization and Authentication*. Springer, 2019, pp. 69–85.

[97] F. Berti, S. Bhasin, J. Breier, X. Hou, R. Poussier, F.-X. Standaert, and B. Udvarhelyi, "A finer-grain analysis of the leakage (non) resilience of ocb," *IACR Transactions on Cryptographic Hardware and Embedded Systems*, pp. 461–481, 2022.

[98] S. M. Sim, D. Jap, and S. Bhasin, ": Differential analysis aided power attack on (non-) linear feedback shift registers," *IACR Transactions on Cryptographic Hardware and Embedded Systems*, pp. 169–191, 2021.

[99] A. Adomnicai, L. Masson, and J. J. Fournier, "Practical algebraic side-channel attacks against acorn," in *International Conference on Information Security and Cryptology*. Springer, 2019, pp. 325–340.

[100] V. Banciu, E. Oswald, and C. Whitnall, "Exploring the resilience of some lightweight ciphers against profiled single trace attacks," in *International Workshop on Constructive Side-Channel Analysis and Secure Design*. Springer, 2015, pp. 51–63.

[101] P. C. Kocher, J. Jaffe, and B. Jun, "Differential Power Analysis," in *CRYPTO '99*, ser. LNCS, vol. 1666. Springer, pp. 388–397.

[102] Eric Brier, Christophe Clavier and Francis Olivier, "Correlation Power Analysis with a Leakage Model," in *CHES*, 2004, pp. 16–29.

[103] H. Le Bouder, "Un formalisme unifiant les attaques physiques sur circuits cryptographiques et son exploitation afin de comparer et rechercher de nouvelles attaques." Ph.D. dissertation, Ecole Nationale Supérieure des Mines de Saint-Etienne, 2014.

[104] S. Chari, J. R. Rao, and P. Rohatgi, "Template attacks," in *Cryptographic Hardware and Embedded Systems-CHES*. Springer, 2002, pp. 13–28.

[105] C. Archambeau, Eric Peeters, F.-X. Standaert, and J.-J. Quisquater, "Template attacks in principal subspaces," in *Cryptographic Hardware and Embedded Systems-CHES*. Springer, 2006, pp. 1–14.

[106] Y. Linge, C. Dumas, and S. Lambert-Lacroix, "Using the joint distributions of a cryptographic function in side channel analysis," in *International Workshop on Constructive Side-Channel Analysis and Secure Design*. Springer, 2014.

[107] S. Mangard, E. Oswald, and T. Popp, *Power analysis attacks: Revealing the secrets of smart cards*. Springer, 2008, vol. 31.

[108] M. Ouladj and S. Guilley, *Side-Channel Analysis of Embedded Systems*. Springer, 2021.

[109] Paul Kocher, "Timing Attacks on Implementations of Diffie-Hellman, RSA, DSS and Other Systems," in *Advances in Cryptology - Crypto'96*. Springer-Verlag, pp. 104–113.

[110] T. S. Messerges, E. A. Dabbish, and R. H. Sloan, "Power analysis attacks of modular exponentiation in smartcards," in *International Workshop on Cryptographic Hardware and Embedded Systems*. Springer, 1999, pp. 144–157.

[111] S. Mangard, "A simple power-analysis (spa) attack on implementations of the aes key expansion," in *ICISC 2002*. Springer, pp. 343–358.

[112] C. Clavier and M. Joye, "Universal exponentiation algorithm a first step towards provable spa-resistance," in *International Workshop on Cryptographic Hardware and Embedded Systems*. Springer, 2001, pp. 300–308.

[113] C. Rechberger and E. Oswald, "Practical template attacks," in *Information Security Applications*. Springer, 2005, pp. 440–456.

[114] M. A. Elaabid, S. Guilley, and P. Hoogvorst, "Template Attacks with a Power Model." *IACR Cryptology ePrint Archive*, p. 443, 2007.

[115] E. Oswald and S. Mangard, "Template attacks on maskingresistance is futile," in *Topics in Cryptology–CT-RSA*. Springer, 2007, pp. 243–256.

[116] O. Choudary and M. G. Kuhn, "Efficient template attacks," in *International Conference on Smart Card Research and Advanced Applications*. Springer, 2013, pp. 253–270.

[117] A. Loiseau, M. Lecomte, and J. J. Fournier, "Template Attacks against ECC: practical implementation against Curve25519," in *IEEE International Symposium on Hardware Oriented Security and Trust (HOST)*. IEEE, 2020, pp. 13–22.

[118] A. Jovic, D. Jap, L. Papachristodoulou, and A. Heuser, "Traditional machine learning methods for side-channel analysis," in *Security and Artificial Intelligence*. Springer, 2022, pp. 25–47.

[119] L. Lerman, G. Bontempi, and O. Markowitch, "Side channel attack: an approach based on machine learning," *Center for Advanced Security Research Darmstadt*, vol. 29, 2011.

[120] G. Hospodar, B. Gierlichs, E. De Mulder, I. Verbauwhede, and J. Vandewalle, "Machine learning in side-channel analysis: a first study," *Journal of Cryptographic Engineering*, vol. 1, no. 4, pp. 293–302, 2011.

[121] C. Genevey-Metat, A. Heuser, and B. Gérard, "Train or adapt a deeply learned profile?" in *International Conference on Cryptology and Information Security in Latin America*. Springer, 2021, pp. 213–232.

[122] H. Li, M. Krček, and G. Perin, "A comparison of weight initializers in deep learning-based side-channel analysis," in *International Conference on Applied Cryptography and Network Security*. Springer, 2020, pp. 126–143.

[123] L. Wu, G. Perin, and S. Picek, "The best of two worlds: Deep learning-assisted template attack," *IACR Transactions on Cryptographic Hardware and Embedded Systems*, pp. 413–437, 2022.

[124] Ronan Lashermes, Guillaume Reymond, Jean-Max Dutertre, Jacques Fournier, Bruno Robisson and Assia Tria, "A DFA on AES Based on the Entropy of Error Distributions," in *FDTC*, 2012.

[125] B. Gierlichs, L. Batina, P. Tuyls, and B. Preneel, "Mutual information analysis," in *CHES, Proceedings*. Springer, 2008.

[126] Youssef Souissi, Maxime Nassar, Sylvain Guilley, Jean-Luc Danger and Florent Flament, "First Principal Components Analysis: A New Side Channel Distinguisher," in *ICISC*, 2010.

[127] Suresh Balakrishnama and Aravind Ganapathiraju, "Linear Discriminant Analysis - A Brief Tutorial," *Institute for Signal and Information Processing, Mississippi State University*, 1998.

[128] C. Clavier and L. Reynaud, "Improved blind side-channel analysis by exploitation of joint distributions of leakages," in *International Conference on Cryptographic Hardware and Embedded Systems (CHES)*. Springer, 2017, pp. 24–44.

[129] C. Clavier, L. Reynaud, and A. Wurcker, "Quadrivariate improved blind side-channel analysis on Boolean masked AES," in *International Workshop on Constructive Side-Channel Analysis and Secure Design*. Springer, 2018, pp. 153–167.

[130] M. Azouaoui, K. Papagiannopoulos, and D. Zürner, "Blind Side-Channel SIFA," in *Design, Automation & Test in Europe Conference & Exhibition (DATE)*. IEEE, 2021, pp. 555–560.

[131] V. Yli-Mäyry, R. Ueno, N. Miura, M. Nagata, S. Bhasin, Y. Mathieu, T. Graba, J.-L. Danger, and N. Homma, "Diffusional side-channel leakage from unrolled lightweight block ciphers: A case study of power analysis on prince," *Transactions on Information Forensics and Security*, vol. 16, pp. 1351–1364, 2020.

[132] R. Novak, "Side-channel attack on substitution blocks," in *ACNS*, vol. 2846. Springer, 2003, pp. 307–318.

[133] C. Clavier, "An improved SCARE cryptanalysis against a secret A3/A8 GSM algorithm," in *Information Systems Security, Third International Conference, ICISS*, P. D. McDaniel and S. K. Gupta, Eds., vol. 4812. Springer, 2007, pp. 143–155.

[134] R. Daudigny, H. Ledig, F. Muller, and F. Valette, "Scare of the des," in *Applied Cryptography and Network Security*, J. Ioannidis, A. Keromytis, and M. Yung, Eds. Springer, 2005, pp. 393–406.

[135] D. Jap and S. Bhasin, "Practical reverse engineering of secret sboxes by side-channel analysis," in *IEEE International Symposium on Circuits and Systems, ISCAS*, 2020, pp. 1–5.

[136] D. Réal, V. Dubois, A. Guilloux, F. Valette, and M. Drissi, "SCARE of an unknown hardware feistel implementation," in *CARDIS*, G. Grimaud and F. Standaert, Eds., vol. 5189. Springer, 2008, pp. 218–227.

[137] M. Rivain and T. Roche, "SCARE of secret ciphers with SPN structures," in *Advances in Cryptology - ASIACRYPT*, K. Sako and P. Sarkar, Eds., vol. 8269. Springer, 2013, pp. 526–544.

[138] S. Guilley, L. Sauvage, J. Micolod, D. Réal, and F. Valette, "Defeating any secret cryptography with SCARE attacks," in *Progress in Cryptology - LATINCRYPT*, M. Abdalla and P. S. L. M. Barreto, Eds., vol. 6212. Springer, 2010, pp. 273–293.

[139] C. Clavier, Q. Isorez, D. Marion, and A. Wurcker, "Complete reverse-engineering of AES-like block ciphers by SCARE and FIRE attacks," *Cryptogr. Commun.*, vol. 7, no. 1, pp. 121–162, 2015.

[140] T. Eisenbarth, C. Paar, and B. Weghenkel, "Building a side channel based disassembler," *Trans. Comput. Sci.*, vol. 10, pp. 78–99, 2010.

[141] V. Cristiani, M. Lecomte, and T. Hiscock, "A bit-level approach to side channel based disassembling," in *CARDIS*, S. Belaïd and T. Güneysu, Eds., vol. 11833. Springer, 2019, pp. 143–158.

[142] R. Granger, P. Jovanovic, B. Mennink, and S. Neves, "Improved Masking for Tweakable Blockciphers with Applications to Authenticated Encryption," in *Advances in Cryptology - EUROCRYPT*, vol. 9665. Springer, 2016, pp. 263–293.

[143] A. Duc, S. Faust, and F.-X. Standaert, "Making masking security proofs concrete," in *Annual International Conference on the Theory and Applications of Cryptographic Techniques*. Springer, 2015, pp. 401–429.

[144] W. Yu and S. Köse, "A lightweight masked aes implementation for securing iot against cpa attacks," *IEEE Transactions on Circuits and Systems I: Regular Papers*, vol. 64, no. 11, pp. 2934–2944, 2017.

[145] S. Mangard and K. Schramm, "Pinpointing the side-channel leakage of masked AES hardware implementations," in *International Workshop on Cryptographic Hardware and Embedded Systems (CHES)*. Springer, 2006, pp. 76–90.

[146] E. Prouff, M. Rivain, and R. Bevan, "Statistical analysis of second order differential power analysis," *IEEE Transactions on computers*, vol. 58, no. 6, pp. 799–811, 2009.

[147] E. Oswald, S. Mangard, C. Herbst, and S. Tillich, "Practical second-order dpa attacks for masked smart card implementations of block ciphers," in *Cryptographers Track at the RSA Conference*. Springer, 2006, pp. 192–207.

[148] T. S. Messerges, "Using second-order power analysis to attack dpa resistant software," in *International Workshop on Cryptographic Hardware and Embedded Systems*. Springer, 2000, pp. 238–251.

[149] J. D. Golić and C. Tymen, "Multiplicative masking and power analysis of AES," in *International Workshop on Cryptographic Hardware and Embedded Systems(CHES)*. Springer, 2002, pp. 198–212.

[150] E. Oswald, S. Mangard, N. Pramstaller, and V. Rijmen, "A side-channel analysis resistant description of the AES S-box," in *International workshop on fast software encryption*. Springer, 2005, pp. 413–423.

[151] D. Knichel, A. Moradi, N. Müller, and P. Sasdrich, "Automated generation of masked hardware," *Cryptology ePrint Archive*, 2021.

[152] T. Beyne, S. Dhooghe, A. Moradi, and A. R. Shahmirzadi, "Cryptanalysis of efficient masked ciphers: Applications to low latency," *IACR Transactions on Cryptographic Hardware and Embedded Systems*, pp. 679–721, 2022.

[153] Y. Belenky, V. Bugaenko, L. Azriel, H. Chernyshchyk, I. Dushar, O. Karavaev, O. Maksimenko, Y. Ruda, V. Teper, and Y. Kreimer, "Redundancy aes masking basis for attack mitigation (rambam)," *IACR Transactions on Cryptographic Hardware and Embedded Systems*, pp. 69–91, 2022.

[154] M. Montoya, T. Hiscock, S. Bacles-Min, A. Molnos, and J. J. Fournier, "Energy-efficient masking of the trivium stream cipher," in *International Conference on Electronics, Circuits and Systems (ICECS)*. IEEE, 2018, pp. 393–396.

[155] M. Joye and F. Olivier, "Side-channel analysis." 2011.

[156] M. Rivain, E. Prouff, and J. Doget, "Higher-order masking and shuffling for software implementations of block ciphers," in *International Workshop on Cryptographic Hardware and Embedded Systems*. Springer, 2009, pp. 171–188.

[157] R. Benadjila, E. Prouff, R. Strullu, E. Cagli, and C. Dumas, "Deep learning for side-channel analysis and introduction to ascad database," *Journal of Cryptographic Engineering*, vol. 10, no. 2, pp. 163–188, 2020.

[158] F. R. Kschischang, B. J. Frey, and H. Loeliger, "Factor graphs and the sum-product algorithm," *IEEE Trans. on Information Theory*, vol. 47, no. 2, pp. 498–519, 2001.

[159] David Barber, *Bayesian Reasoning and Machine Learning*, 04-2011 ed. Cambridge University Press, 2011.

[160] R. G. Gallager, "Low-density parity-check codes," *IRE Trans. on Information Theory*, vol. 8, no. 1, pp. 21–28, 1962.

[161] Robert G. Gallager, "Low Density Parity check codes," Ph.D. dissertation, MIT, Cambridge, MA, 1963.

[162] R. M. Tanner, "A recursive approach to low complexity codes," *IEEE Trans. on Information Theory*, vol. 27, no. 5, pp. 533–547, 1981.

[163] J. Pearl, "Reverend bayes on inference engines: A distributed hierarchical approach," in *Probabilistic and Causal Inference: The Works of Judea Pearl*, 1982, pp. 129–138.

[164] N. Veyrat-Charvillon, B. Gérard, and F. Standaert, "Soft Analytical Side-Channel Attacks," in *ASIACRYPT 2014*, pp. 282–296.

[165] V. Grosso and F.-X. Standaert, "ASCA, SASCA and DPA with Enumeration: Which One Beats the Other and When?" in *ASIACRYPT 2015*. Springer, pp. 291–312.

[166] V. Cristiani, M. Lecomte, and T. Hiscock, "A bit-level approach to side channel based disassembling," in *International Conference on Smart Card Research and Advanced Applications*. Springer, 2019, pp. 143–158.

[167] D. Strobel, F. Bache, D. Oswald, F. Schellenberg, and C. Paar, "Scandalee: a side-channel-based disassembler using local electromagnetic emanations," in *Design, Automation & Test in Europe Conference & Exhibition (DATE)*. IEEE, 2015, pp. 139–144.

[168] A. Caforio, F. Balli, and S. Banik, "Complete Practical Side-Channel-Assisted Reverse Engineering of AES-Like Ciphers," in *International Conference on Smart Card Research and Advanced Applications*. Springer, 2021, pp. 97–117.

[169] N. Burow, S. A. Carr, J. Nash, P. Larsen, M. Franz, S. Brunthaler, and M. Payer, "Control-flow integrity: Precision, security, and performance," *ACM Computing Surveys (CSUR)*, vol. 50, no. 1, pp. 1–33, 2017.

[170] T. Hiscock, O. Savry, and L. Goubin, "Lightweight instruction-level encryption for embedded processors using stream ciphers," *Microprocessors and Microsystems*, vol. 64, pp. 43–52, 2019.

[171] C. Clavier, J.-S. Coron, and N. Dabbous, "Differential power analysis in the presence of hardware countermeasures," in *International Workshop on Cryptographic Hardware and Embedded Systems*. Springer, 2000, pp. 252–263.

[172] Sébanjila Kevin Bukasa and Ronan Lashermes and Jean-Louis Lanet and Axel Legay, title = Let's shock our IoT's heart: ARMv7-M under (fault) attacks, booktitle = ARES 2018. ACM.

[173] D. Couroussé, B. Robisson, J.-L. Lanet, T. Barry, H. Noura, P. Jaillon, and P. Lalevée, "Cogito: Code polymorphism to secure devices," in *11th International Conference on Security and Cryptography (SECRYPT)*. IEEE, 2014, pp. 1–6.

[174] Dutertre, Jean-Max and Fournier, Jacques JA and Mirbaha, Amir-Pasha and Naccache, David and Rigaud, Jean-Baptiste and Robisson, Bruno and Tria, Assia, "Review of fault injection mechanisms and consequences on countermeasures design," in *6th International Conference on Design & Technology of Integrated Systems in Nanoscale Era (DTIS)*. IEEE, 2011, pp. 1–6.

[175] Barenghi, Alessandro and Breveglieri, Luca and Koren, Israel and Naccache, David, "Fault injection attacks on cryptographic devices: Theory, practice, and countermeasures," *Proceedings of the IEEE*, vol. 100, no. 11, pp. 3056–3076, 2012.

[176] Roscian, Cyril and Dutertre, Jean-Max and Tria, Assia, "Frontside laser fault injection on cryptosystems-Application to the AES'last round," in *International Symposium on Hardware-Oriented Security and Trust (HOST)*. IEEE, 2013, pp. 119–124.

[177] Lacruche, Marc and Borrel, Nicolas and Champeix, Clement and Roscian, Cyril and Sarafianos, Alexandre and Rigaud, Jean-Baptiste and Dutertre, Jean-Max and Kussener, Edith, "Laser fault injection into sram cells: Picosecond versus nanosecond pulses," Ph.D. dissertation, 2015.

[178] Agoyan, Michel and Dutertre, Jean-Max and Mirbaha, Amir-Pasha and Naccache, David and Ribotta, Anne-Lise and Tria, Assia, "Single-bit DFA using multiple-byte laser fault injection," in *International Conference on Technologies for Homeland Security (HST)*. IEEE, 2010, pp. 113–119.

[179] Selmke, Bodo and Pollanka, Maximilian and Duensing, Andreas and Strieder, Emanuele and Wen, Hayden and Mittermair, Michael and Sigl, Georg and others, "On the application of Two-Photon Absorption for Laser Fault Injection attacks: Pushing the physical boundaries for Laser-based Fault Injection," *IACR Transactions on Cryptographic Hardware and Embedded Systems*, pp. 862–885, 2022.

[180] Dumont, Mathieu and Moëllic, Pierre-Alain and Viera, Raphael and Dutertre, Jean-Max and Bernhard, Rémi, "An overview of laser injection against embedded neural network models," in *7th World Forum on Internet of Things (WF-IoT)*. IEEE, 2021, pp. 616–621.

[181] Viera, Raphael and Dutertre, Jean-Max and Dumont, Mathieu and Moëllic, Pierre-Alain, "Permanent Laser Fault Injection into the Flash Memory of a Microcontroller," in *19th International New Circuits and Systems Conference (NEWCAS)*. IEEE, 2021, pp. 1–4.

[182] Dehbaoui, Amine and Dutertre, Jean-Max and Robisson, Bruno and Tria, Assia, "Electromagnetic transient faults injection on a hardware and a software implementations of AES," in *Workshop on Fault Diagnosis and Tolerance in Cryptography (FDTC)*. IEEE, 2012, pp. 7–15.

[183] P. Bayon, L. Bossuet, A. Aubert, V. Fischer, F. Poucheret, B. Robisson, and P. Maurine, "Contactless electromagnetic active attack on ring oscillator based true random number generator," in *Constructive Side-Channel Analysis and Secure Design (COSADE)*. Springer, 2012, pp. 151–166.

[184] Khuat, Vanthanh and Trabelsi, Oualid and Sauvage, Laurent and Danger, Jean-Luc, "Multiple and reproducible fault models on micro-controller using electromagnetic fault injection," in *International Joint EMC/SI/PI and EMC Europe Symposium*. IEEE, 2021, pp. 667–672.

[185] Lanet, Jean-Louis, "When Fault Injection Collides with Hardware Complexity," in *Foundations and Practice of Security (FPS)*, vol. 11358. Springer, 2019, p. 243.

[186] Balasch, Josep and Gierlichs, Benedikt and Verbauwhede, Ingrid, "An In-depth and Black-box Characterization of the Effects of Clock Glitches on 8-bit MCUs," in *Workshop on Fault Diagnosis and Tolerance in Cryptography (FDTC)*. IEEE, 2011, pp. 105–114.

[187] Zussa, Loic and Dutertre, Jean-Max and Clediere, Jessy and Tria, Assia, "Power supply glitch induced faults on FPGA: An in-depth analysis of the injection mechanism," in *19th International On-Line Testing Symposium (IOLTS)*. IEEE, 2013, pp. 110–115.

[188] Claudepierre, Ludovic and Péneau, Pierre-Yves and Hardy, Damien and Rohou, Erven, "TRAITOR: a low-cost evaluation platform for multifault injection," in *International Symposium on Advanced Security on Software and Systems*, 2021, pp. 51–56.

[189] Eli Biham and Adi Shamir., "Differential Fault Analysis of Secret Key Cryptosystems," in *CRYPTO*, 1997.

[190] P. Dusart, G. Letourneux, and O. Vivolo, "Differential fault analysis on aes," in *International Conference on Applied Cryptography and Network Security*. Springer, 2003, pp. 293–306.

[191] G. Piret and J.-J. Quisquater, "A differential fault attack technique against spn structures, with application to the aes and khazad," in *International workshop on cryptographic hardware and embedded systems*. Springer, 2003, pp. 77–88.

[192] Christophe Giraud, "DFA on AES," in *Advanced Encryption Standard - AES*, ser. Lecture Notes in Computer Science, vol. 3373. Springer, 2005, pp. 27–41.

[193] M. Minier, "Improving impossible-differential attacks against rijndael-160 and rijndael-224," *Designs, Codes and Cryptography*, vol. 82, no. 1, pp. 117–129, 2017.

[194] Trouchkine, Thomas and Bukasa, Sébanjila Kevin and Escouteloup, Mathieu and Lashermes, Ronan and Bouffard, Guillaume, "Electromagnetic fault injection against a complex CPU, toward new micro-architectural fault models," *Journal of Cryptographic Engineering*, vol. 11, no. 4, pp. 353–367, 2021.

[195] Moro, Nicolas and Dehbaoui, Amine and Heydemann, Karine and Robisson, Bruno and Encrenaz, Emmanuelle, "Electromagnetic fault injection: towards a fault model on a 32-bit microcontroller," in *Workshop on Fault Diagnosis and Tolerance in Cryptography (FDTC)*. IEEE, 2013, pp. 77–88.

[196] Breier, Jakub and Jap, Dirmanto and Chen, Chien-Ning, "Laser profiling for the back-side fault attacks: with a practical laser skip instruction attack on AES," in *Proceedings of the 1st ACM Workshop on Cyber-Physical System Security*, 2015, pp. 99–103.

[197] Dutertre, Jean-Max and Riom, Timothé and Potin, Olivier and Rigaud, Jean-Baptiste, "Experimental analysis of the laser-induced instruction skip fault model," in *Nordic Conference on Secure IT Systems*. Springer, 2019, pp. 221–237.

[198] Menu, Alexandre and Dutertre, Jean-Max and Potin, Olivier and Rigaud, Jean-Baptiste and Danger, Jean-Luc, "Experimental analysis of the electromagnetic instruction skip fault model," in *15th Design & Technology of Integrated Systems in Nanoscale Era (DTIS)*. IEEE, 2020, pp. 1–7.

[199] Riviere, Lionel and Najm, Zakaria and Rauzy, Pablo and Danger, Jean-Luc and Bringer, Julien and Sauvage, Laurent, "High precision fault injections on the instruction cache of ARMv7-M architectures," in *International Symposium on Hardware Oriented Security and Trust (HOST)*. IEEE, 2015, pp. 62–67.

[200] Khuat, Vanthanh and Dutertre, Jean-Max and Danger, Jean-Luc, "Analysis of a laser-induced instructions replay fault model in a 32-bit microcontroller," in *24th Euromicro Conference on Digital System Design (DSD)*. IEEE, 2021, pp. 363–370.

[201] Ordas, Sébastien and Guillaume-Sage, Ludovic and Maurine, Philippe, "Electromagnetic fault injection: the curse of flip-flops," *Journal of Cryptographic Engineering*, vol. 7, pp. 183–197, 2017.

[202] N. Joshi, K. Wu, and R. Karri, "Concurrent error detection schemes for involution ciphers," in *International Workshop on Cryptographic Hardware and Embedded Systems*. Springer, 2004, pp. 400–412.

[203] M. Karpovsky, K. J. Kulikowski, and A. Taubin, "Robust protection against fault-injection attacks on smart cards implementing the advanced encryption standard," in *International Conference on Dependable Systems and Networks*. IEEE, 2004, pp. 93–101.

[204] N. A. Manssour, V. Lapotre, G. Gogniat, and A. Tisserand, "Processor extensions for hardware instruction replay against fault injection attacks," in *25th International Symposium on Design and Diagnostics of Electronic Circuits and Systems (DDECS)*. IEEE, 2022, pp. 26–31.

[205] T. Bartkewitz, S. Bettendorf, T. Moos, A. Moradi, and F. Schellenberg, "Beware of Insufficient Redundancy: An Experimental Evaluation of Code-based FI Countermeasures," *IACR Transactions on Cryptographic Hardware and Embedded Systems*, pp. 438–462, 2022.

[206] G. Di Natale, M. Doulcier, M.-L. Flottes, and B. Rouzeyre, "A reliable architecture for parallel implementations of the advanced encryption standard," *Journal of Electronic Testing*, vol. 25, no. 4-5, pp. 269–278, 2009.

[207] Z. Wang, M. Karpovsky, and A. Joshi, "Secure multipliers resilient to strong fault-injection attacks using multilinear arithmetic codes," *IEEE transactions on very large scale integration (VLSI) systems*, vol. 20, no. 6, pp. 1036–1048, 2011.

[208] M. Montoya, S. Bacles-Min, A. Molnos, and J. J. Fournier, "Dynamic encoding, a lightweight combined countermeasure against hardware attacks," in *23rd Euromicro Conference on Digital System Design (DSD)*. IEEE, 2020, pp. 185–192.

[209] Dutertre, Jean-Max and Fournier, Jacques JA and Mirbaha, Amir-Pasha and Naccache, David and Rigaud, Jean-Baptiste and Robisson, Bruno and Tria, Assia, "Review of fault injection mechanisms and consequences on countermeasures design," in *6th International Conference on Design & Technology of Integrated Systems in Nanoscale Era (DTIS)*. IEEE, 2011, pp. 1–6.

[210] Gaine, Clément and Nikolovski, Jean-Pierre and Aboulkassimi, Driss and Dutertre, Jean-Max, "Active Shielding Against Physical Attacks by Observation and Fault Injection: ChaXa," *Journal of Hardware and Systems Security*, vol. 7, no. 1, pp. 1–10, 2023.

[211] R. De Clercq, J. Götzfried, D. Übler, P. Maene, and I. Verbauwhede, "SOFIA: software and control flow integrity architecture," *Computers & Security*, vol. 68, pp. 16–35, 2017.

[212] T. Barry, D. Couroussé, and B. Robisson, "Compilation of a countermeasure against instruction-skip fault attacks," in *Proceedings of the Third Workshop on Cryptography and Security in Computing Systems*, 2016, pp. 1–6.

[213] D. Couroussé, T. Barry, B. Robisson, P. Jaillon, O. Potin, and J.-L. Lanet, "Runtime code polymorphism as a protection against side channel attacks," in *IFIP International Conference on Information Security Theory and Practice*. Springer, 2016, pp. 136–152.

[214] H.-P. Charles, D. Couroussé, V. Lomüller, F. A. Endo, and R. Gauguey, "degoal a tool to embed dynamic code generators into applications," in *International Conference on Compiler Construction*. Springer, 2014, pp. 107–112.

[215] T. Kotzmann, C. Wimmer, H. Mössenböck, T. Rodriguez, K. Russell, and D. Cox, "Design of the java hotspot client compiler for java 6," *ACM Transactions on Architecture and Code Optimization (TACO)*, vol. 5, no. 1, pp. 1–32, 2008.

[216] O. Savry, M. El-Majihi, and T. Hiscock, "Confidaent: Control flow protection with instruction and data authenticated encryption," in *23rd Euromicro Conference on Digital System Design (DSD)*. IEEE, 2020, pp. 246–253.

[217] J. Borghoff, A. Canteaut, T. Güneysu, E. B. Kavun, M. Knezevic, L. R. Knudsen, G. Leander, V. Nikov, C. Paar, C. Rechberger, P. Rombouts, S. S. Thomsen, and T. Yalçin, "PRINCE - A low-latency block cipher for pervasive computing applications - extended abstract," in *ASIACRYPT 2012*, ser. Lecture Notes in Computer Science, vol. 7658. Springer, 2012, pp. 208–225.

[218] S. Banik, A. Bogdanov, T. Isobe, K. Shibutani, H. Hiwatari, T. Akishita, and F. Regazzoni, "Midori: A block cipher for low energy," in *ASIACRYPT 2015 - Proceedings, Part II*, ser. Lecture Notes in Computer Science, T. Iwata and J. H. Cheon, Eds., vol. 9453. Springer, 2015, pp. 411–436.

[219] C. Beierle, J. Jean, S. Kölbl, G. Leander, A. Moradi, T. Peyrin, Y. Sasaki, P. Sasdrich, and S. M. Sim, "The SKINNY family of block ciphers and its low-latency variant MANTIS," in *CRYPTO 2016 - Proceedings, Part II*, ser. Lecture Notes in Computer Science, M. Robshaw and J. Katz, Eds., vol. 9815. Springer, 2016, pp. 123–153.

[220] S. Banik, T. Isobe, F. Liu, K. Minematsu, and K. Sakamoto, "Orthros: A low-latency PRF," *IACR Trans. Symmetric Cryptol.*, vol. 2021, no. 1, pp. 37–77, 2021.

[221] R. Avanzi, "The QARMA block cipher family," *IACR Trans. Symmetric Cryptol.*, vol. 2017, no. 1, pp. 4–44, 2017.

[222] G. Leander, T. Moos, A. Moradi, and S. Rasoolzadeh, "The SPEEDY family of block ciphers engineering an ultra low-latency cipher from gate level for secure processor architectures," *IACR Trans. Cryptogr. Hardw. Embed. Syst.*, vol. 2021, no. 4, pp. 510–545, 2021.

[223] Y. Koizumi, Y. Kawaguchi, K. Imoto, T. Nakamura, Y. Nikaido, R. Tanabe, H. Purohit, K. Suefusa, T. Endo, M. Yasuda *et al.*, "Description and discussion on DCASE2020 challenge task2: Unsupervised anomalous sound detection for machine condition monitoring," *arXiv*, 2020.

[224] D.-P. Pham, D. Marion, M. Mastio, and A. Heuser, "Obfuscation revealed: Leveraging electromagnetic signals for obfuscated malware classification," in *Annual Computer Security Applications Conference*, 2021, pp. 706–719.

[225] X. T. Ngo, I. Exurville, S. Bhasin, J.-L. Danger, S. Guilley, Z. Najm, J.-B. Rigaud, and B. Robisson, "Hardware trojan detection by delay and electromagnetic measurements," in *DATE*. IEEE, 2015, pp. 782–787.

[226] S. Narasimhan, D. Du, R. S. Chakraborty, S. Paul, F. G. Wolff, C. A. Papachristou, K. Roy, and S. Bhunia, "Hardware trojan detection by multiple-parameter side-channel analysis," *IEEE Transactions on computers*, vol. 62, no. 11, pp. 2183–2195, 2012.

[227] S. L. Thomas and A. Francillon, "Backdoors: Definition, deniability and detection," in *International Symposium on Research in Attacks, Intrusions, and Defenses*. Springer, 2018, pp. 92–113.

IMT Atlantique
Bretagne-Pays de la Loire
École Mines-Télécom

IMT Atlantique Bretagne - Pays de la Loire - www.imt-atlantique.fr

Campus de Brest
Technopôle Brest-Iroise
CS 83818
29238 Brest Cedex 03
T +33 (0)2 29 00 11 11
F +33 (0)2 29 00 10 00

Campus de Nantes
4, rue Alfred Kastler - La Chantrerie
CS 20722
44307 Nantes Cedex 03
T +33 (0)2 51 85 81 00
F +33 (0)2 51 85 81 99

Campus de Rennes
2, rue de la Châtaigneraie
CS 17607
35576 Cesson Sévigné Cedex
T +33 (0)2 99 12 70 00
F +33 (0)2 99 12 70 08